Oracle® Communications Converged Application Server

Administrator's Guide

Release 5.1

E27704-01

December 2012

Oracle Communications Converged Application Server Administrator's Guide, Release 5.1

E27704-01

Contents

Preface

This document gives an overview of Oracle Communications Converged Application Server architecture and management and provides configuration information for the data tier, engine tier, geographic redundancy, and performance. It also provides information on upgrading from previous releases of Converged Application Server.

Audience

This document is intended for those who set up Converged Application Server and its domains and who upgrade from previous versions of Converged Application Server.

Related Documents

For more information, see the following documents in the Oracle Communications Converged Application Server Release 5.1 documentation set:

- *Converged Application Server Installation Guide*
- *Converged Application Server Release Notes*
- *Converged Application Server Concepts*
- *Converged Application Server Security Guide*
- *Converged Application Server Developer's Guide*
- *Converged Application Server Diameter Application Development Guide*

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.

Part I

Configuring Converged Application Server

This part provides an overview of Oracle Communications Converged Application Server architecture and management and provides configuration information for the data tier, engine tier, geographic redundancy, and performance. It also provides information on upgrading from previous releases of Converged Application Server.

This part contains the following chapters:

1 Converged Application Server Configuration Overview

This chapter introduces Oracle Communications Converged Application Server configuration and administration. It covers the following topics:

- About the Oracle WebLogic Platform
- Overview of Configuration and Administration Tools
- Common Configuration Tasks

About the Oracle WebLogic Platform

Converged Application Server is based on Oracle WebLogic Server 11*g*, and many system-level configuration tasks are the same for both products. This guide addresses only those system-level configuration tasks that are unique to Converged Application Server, such as tasks related to network and security configuration and cluster configuration for the engine and SIP data tiers.

HTTP server configuration and other basic configuration tasks such as server logging are addressed in the Oracle WebLogic Server documentation. See *Oracle Fusion Middleware Getting Started With Installation for Oracle WebLogic Server* to get started.

Overview of Configuration and Administration Tools

You can apply configuration changes using the Administration Console or from the command line using the WLST utility. Changes to certain SIP Servlet container properties require a restart of the engine tier server for the change to take affect. In such cases, a *Restart may be required* icon appears in the console. Configuration for SIP data tier nodes cannot be changed dynamically, so you must reboot SIP data tier servers in order to change the number of partitions or replicas.

The following sections contain more information about the configuration tools:

- Administration Console
- WebLogic Scripting Tool (WLST)
- Additional Configuration Methods

Administration Console

The Converged Application Server extends the WebLogic Administration Console user interface with its own configuration and monitoring pages. The Administration

Console interface for Converged Application Server settings are similar to the core console available in Oracle WebLogic Server 11*g*.

All Converged Application Server configuration and monitoring is provided through these nodes in the left pane of the console:

- SipServer: presents SIP Servlet container properties and other engine tier functionality. This extension also enables you to view (but not modify) SIP data tier partitions and replicas.
- Converged Load Balancer: presents configuration settings and monitoring pages for the activities of the converged load balancers in the implementation.
- Diameter: presents Diameter nodes and application configuration settings.

See "Accessing the Administration Console" for more information about using the console in the Oracle WebLogic Server documentation.

WebLogic Scripting Tool (WLST)

The WebLogic Scripting Tool (WLST) enables you to perform interactive or automated (batch) configuration operations using a command-line interface. WLST is a JMX tool that can view or manipulate the MBeans available in a running Converged Application Server domain.

See "Using WLST (JMX) to Configure Converged Application Server" for more information about modifying SIP Servlet container properties using WLST.

For general WebLogic Server information, see the following documents:

- For information about WLST, see *Oracle WebLogic Scripting Tool*.
- For information about WLST commands, see *WebLogic Scripting Tool Command Reference*.

Additional Configuration Methods

Most Converged Application Server configuration is performed using either the Administration Console or WLST. The methods described in the following sections may also be used for certain configuration tasks.

Editing Configuration Files

You may also modify the configuration by editing configuration files.

The Converged Application Server custom resources utilize the basic domain resources defined in **config.xml**, such as network channels, cluster and server configuration, and Java EE resources. The **config.xml** file applies to all managed servers in the domain. However, standalone Converged Application Server components are configured in separate configuration files based on functionality:

- **sipserver.xml** contains general SIP container properties and engine tier configuration settings.
- **datatier.xml** identifies servers that participate as replicas in the SIP data tier, and also defines the number and layout of SIP data tier partitions.
- **diameter.xml** defines Diameter nodes and Diameter protocol applications used in the domain.
- **approuter.xml** contains the configuration for the Default Application Router (DAR) and the Custom Application Router (CAR).

- **clb-local-config.xml** contains the Converged Load Balancer configuration.

The component configuration files determine the role of each server instance, such as whether they behave as SIP data tier replicas, engine tier nodes, or Diameter client nodes.

See Part IV, "Reference" for more information on the configuration files.

If you edit configuration files manually, you must reboot all servers to apply the configuration changes.

Custom JMX Applications

Converged Application Server properties are represented by JMX-compliant MBeans. You can therefore program JMX applications to configure SIP container properties using the appropriate Converged Application Server MBeans.

The general procedure for modifying Converged Application Server MBean properties using JMX is described in "Using WLST (JMX) to Configure Converged Application Server". For more information about the individual MBeans used to manage SIP container properties, see the Converged Application Server JavaDocs.

Common Configuration Tasks

General administration and maintenance of Converged Application Server requires that you manage both WebLogic Server configuration properties and Converged Application Server container properties.

Common configuration tasks include:

- Configure SIP Container Properties using the Administration Console or using WLST to perform batch configuration. See "Configuring Converged Application Container Properties" for more information.
- Assign Converged Application Server instances to the SIP data tier partitions and setting up call state replication using data tier instances. See "Configuring SIP Data Tier Partitions and Replicas" for more information.
- Configure WebLogic Server network channels to handle SIP and HTTP traffic. See "Configuring Network Connection Settings" for more information.
- Configure load balancers, proxy registrar, diameter components, or other infrastructure elements to support the Converged Application Server deployment. See "Configuring Infrastructure Components" for more information.
- Deploy applications to the Converged Application Servers. See *Oracle Communications Converged Application Server Developer's Guide* for more information.
- Create and deploy logging Servlets to record SIP requests and responses and manage log records. See "Logging SIP Requests and Responses" for more information.

2 Getting Started

This chapter describes how to start and stop servers in an Oracle Communications Converged Application Server domain:

- Accessing the Administration Console
- Using WLST (JMX) to Configure Converged Application Server
- Setting Logging Levels
- Startup Sequence for a Converged Application Server Domain
- Startup Command Options
- Reverting to the Original Boot Configuration

Accessing the Administration Console

The Administration Console enables you to configure and monitor core WebLogic Server functionality as well as the SIP Servlet container functionality provided with Converged Application Server. To configure or monitor SIP Servlet features using the Administration Console:

1. Use your browser to access the URL:

   `http://`*address*`:`*port*`/console`

   where *address* is the Administration Server's listen address and *port* is the listen port.

2. Select the **SipServer** node in the left pane.

   The right pane of the console provides two levels of tabbed pages that are used for configuring and monitoring Converged Application Server. Table 2–1 summarizes the available pages and provides links to additional information about configuring SIP container properties.

Table 2–1 Converged Application Server Configuration and Monitoring Pages

Tab	SubTab	Function
Configuration	General	Configure SIP timer values, session timeout duration, default Converged Application Server behavior (proxy or user agent), server header format, call state caching, DNS name resolution, timer affinity, domain aliases, rport support, and diagnostic image format.
Configuration	Application Router	Configure custom Application Router (AR) class name, configuration, or default application. See "Composing SIP Applications" in *Converged Application Server Developer's Guide*.
Configuration	Proxy	Configure proxy routing URIs and proxy policies.

Table 2–1 (Cont.) Converged Application Server Configuration and Monitoring Pages

Tab	SubTab	Function
Configuration	Overload Protection	Configure the conditions for enabling and disabling automatic overload controls.
Configuration	Message Debug	Enable or disable SIP message logging on a development system.
Configuration	SIP Security	Identify trusted hosts for which authentication is not performed.
Configuration	Persistence	Configure persistence options for storing long-lived session data in an RDBMS, or for replicating long-lived session data to a remote, geographically-redundant site.
Configuration	Data Tier	View the current configuration of SIP data tier servers. You can also add, delete and configure partitions here.
Configuration	LoadBalancer Map	Configure the mapping of multiple clusters to internal virtual IP addresses during a software upgrade.
Configuration	Targets	Configure the list of servers or clusters that receive the engine tier configuration. The target server list determines which servers and/or clusters provide SIP Servlet container functionality.
Configuration	Connection Pools	Configure connection reuse pools to minimize communication overhead with a Session Border Control (SBC) function or Serving Call Session Control Function (S-CSCF).
Monitoring	General	View runtime information about messages and sessions processed in engine tier servers.
Monitoring	SIP Performance	View runtime performance information on SIP traffic throughput and number of successful and failed transactions.
Monitoring	SIP Applications	View runtime session information for deployed SIP applications.
Monitoring	Data Tier Information	View runtime information about the current status and the work performed by servers in the SIP data tier.

Locking and Persisting the Configuration

The Administration Console Change Center provides a way to lock a domain configuration so you can make configuration changes while preventing other administrators from making changes during your edit session. You can enable or disable this feature in development domains. It is disabled by default when you create a new development domain. See "Enable and disable the domain configuration lock" in the Administration Console Online Help for more information.

Some changes you make in the Administration Console take place immediately when you activate them. Other changes require you to restart the server or module affected by the change. These latter changes are called non-dynamic changes. Non-dynamic changes are indicated in the Administration Console with a warning icon containing an exclamation point. If an edit is made to a non-dynamic configuration setting, no edits to dynamic configuration settings will take effect until after you restart the server.

For more information on using Oracle WebLogic Server Administration Console, see *Administrator's Guide* in the Oracle WebLogic Server 11*g* documentation.

To make changes:

1. Locate the Change Center in the upper left corner of the Administration Console.
2. Click **Lock & Edit** to lock the editable configuration hierarchy for the domain. This enables you to make changes using the Administration Console.

3. Make the changes you desire on the relevant page of the console and click **Save** on each page where you make a change.
4. When you have finished making all the desired changes, click **Activate Changes** in the Change Center.

Note:

- You can instead discard your current changes by clicking **Undo All Changes**. This deletes any temporary configuration files that were written with previous **Save** operations.
- If you need to discard all configuration changes made since the server was started, you can revert to original boot configuration file. See "Reverting to the Original Boot Configuration" for more information.

Using WLST (JMX) to Configure Converged Application Server

The WebLogic Scripting Tool (WLST) is a utility that you can use to observe or modify JMX MBeans available on a WebLogic Server or Converged Application Server instance. To learn how to use WLST, refer to the *Oracle WebLogic Scripting Tool* documentation.

Before using WLST to configure a Converged Application Server domain, set your environment to add required Converged Application Server classes to your classpath. Use either a domain environment script or the `setWLSEnv.sh` script located in *WL_home*/**server/bin** where *WL_home* is the directory where WebLogic Server is installed. The default WebLogic Server home directory is named **wlserver_10.3**.

Managing Configuration Locks

Table 2–2 summarizes the WLST methods used to lock a configuration and apply changes.

Table 2–2 ConfigManagerRuntimeMBean Method Summary

Method	Description
`activate()`	Writes the current configuration MBean attributes (the current SIP Servlet container configuration) to the **sipserver.xml** configuration file and applies changes to the running servers.
`cancelEdit()`	Cancels an edit session, releasing the edit lock, and discarding all unsaved changes. This operation can be called by any user with administrator privileges, even if the user did not start the edit session.
`cd`	Navigate the hierarchy of configuration or runtime beans.
`connect`	Connect WLST to a WebLogic Server instance.
`edit`	Starts an edit session.
`save()`	Writes the current configuration MBean attributes (the current SIP Servlet container configuration) to a temporary configuration file.
`startEdit()`	Locks changes to the SIP Servlet container configuration. Other JMX applications cannot alter the configuration until you explicitly call `stopEdit()`, or until your edit session is terminated. If you attempt to call `startEdit()` when another user has obtained the lock, you receive an error message that states the user who owns the lock.

Table 2–2 (Cont.) ConfigManagerRuntimeMBean Method Summary

Method	Description
`set`	Set the specified attribute value for the current configuration bean.
`stopEdit()`	Releases the lock obtained for modifying SIP container properties and rolls back any pending MBean changes, discarding any temporary files.

A typical configuration session involves the following tasks:

1. Call `startEdit()` to obtain a lock on the active configuration.
2. Modify existing SIP Servlet container configuration MBean attributes (or create or delete configuration MBeans) to modify the active configuration. See "Configuration MBeans for the SIP Servlet Container" for a summary of the configuration MBeans.
3. Call `save()` to persist all changes to a temporary configuration file named **sipserver.xml.saved**, or
4. Call `activate()` to persist changes to the **sipserver.xml.saved** file, rename **sipserver.xml.saved** to **sipserver.xml** (copying over the existing file), and apply changes to the running engine tier server nodes.

Note: When you boot the Administration Server for a Converged Application Server domain, the server parses the current container configuration in **sipserver.xml** and creates a copy of the initial configuration in a file named **sipserver.xml.booted**. You can use this copy to revert to the booted configuration, as described in "Reverting to the Original Boot Configuration".

Configuration MBeans for the SIP Servlet Container

`ConfigManagerRuntimeMBean` manages access to and persists the configuration MBean attributes described in Table 2–3. Although you can modify other configuration MBeans, such as WebLogic Server MBeans that manage resources such as network channels and other server properties, those MBeans are not managed by `ConfigManagerRuntimeMBean`.

Table 2–3 SIP Container Configuration MBeans

MBean Type	MBean Attributes	Description

`ClusterToLoadBalancerMap`	`ClusterName,`
	`LoadBalancerSipURI`
		Manages the mapping of multiple clusters to internal virtual IP addresses during a software upgrade. This attribute is not used during normal operations.
		
		See also "Define the Cluster-to-Load Balancer Mapping" in Chapter 10, "Upgrading Production Converged Application Server Software."

`OverloadProtection`	`RegulationPolicy,`
	`ThresholdValue,`
	`ReleaseValue`
		Manages overload settings for throttling incoming SIP requests.
		
		See also "overload" in Chapter 21, "Engine Tier Configuration Reference (sipserver.xml)."

`Proxy`	`ProxyURIs,`
	`RoutingPolicy`
		Manages the URIs routing policies for proxy servers. See also "proxy—Setting Up an Outbound Proxy Server" in *Chapter 21, "Engine Tier Configuration Reference (sipserver.xml)."*

`SipSecurity`	`TrustedAuthenticationHosts`	Defines trusted hosts for which authentication is not performed. See also "sip-security" in *Chapter 21, "Engine Tier Configuration Reference (sipserver.xml)."*

`SipServer`	`DefaultBehavior,`
	`EnableLocalDispatch,`
	`MaxApplicationSessionLifeTime,`
	`OverloadProtectionMBean,`
	`ProxyMBean,`
	`T1TimeoutInterval,`
	`T2TimeoutInterval,`
	`T4TimeoutInterval,`
	`TimerBTimeoutInterval,`
	`TimerFTimeoutInterval`
	
	SipServer also has several helper methods:
	
	`createProxy(),`
	`destroyProxy(),`
	`createOverloadProtection(),`
	`destroyOverloadProtection(),`
	`createClusterToLoadBalancerMap()`
	`destroyClusterToLoadBalancerMap(`
	`)`
		Configuration MBean that represents the entire **sipserver.xml** configuration file. You can use this MBean to obtain and manage each of the individual MBeans described in this table, or to set SIP timer or SIP Session timeout values. See also:
		■ Creating and Deleting MBeans
		■ default-behavior,
		■ enable-local-dispatch
		■ max-application-session-lifetime
		■ t1-timeout-interval
		■ t2-timeout-interval
		■ t4-timeout-interval
		■ timer-b-timeout-interval
		■ timer-f-timeout-interval

Locating the Converged Application Server MBeans

All SIP Servlet container configuration MBeans are located in the "serverConfig" MBean tree, accessed using the `serverConfig()` command in WLST. Within this bean tree, individual configuration MBeans can be accessed using the path:

```
CustomResources/sipserver/Resource/sipserver
```

For example, to browse the default Proxy MBean for a Converged Application Server domain you would enter these WLST commands:

```
serverConfig()
cd('CustomResources/sipserver/Resource/sipserver/Proxy')
ls()
```

Runtime MBeans, such as `ConfigManagerRuntime`, are accessed in the "custom" MBean tree, accessed using the `custom()` command in WLST. Runtime MBeans use the path:

```
mydomain:Location=myserver,Name=myserver,Type=mbeantype
```

Certain configuration settings, such as proxy and overload protection settings, are defined by default in **sipserver.xml**. Configuration MBeans are generated for these settings when you boot the associated server, so you can immediately browse the `Proxy` and `OverloadProtection` MBeans. Other configuration settings are not configured by default and you will need to create the associated MBeans before they can be accessed. See "Creating and Deleting MBeans".

WLST Configuration Examples

The following sections provide example WLST scripts and commands for configuring SIP Servlet container properties.

Invoking WLST

To use WLST with Converged Application Server, you must ensure that all Converged Application Server JAR files are included in your classpath. Follow these steps:

1. Set your Converged Application Server environment:

   ```
   cd ~/domain_home/bin
   ./setDomainEnv.sh
   ```

   where *domain_home* is the path to the domain's home directory.

2. Start WLST:

   ```
   java weblogic.WLST
   ```

3. Connect to the Administration Server for your Converged Application Server domain:

   ```
   connect('system','weblogic','t3://myadminserver:port_number')
   ```

WLST Template for Configuring Container Attributes

Because a typical configuration session involves accessing `ConfigManagerRuntimeMBean` twice—once for obtaining a lock on the configuration, and once for persisting the configuration and/or applying changes—JMX applications that manage container attributes generally have a similar structure. Example 2–1 shows a WLST script that contains the common commands needed to access `ConfigManagerRuntimeMBean`. The example script modifies the proxy `RoutingPolicy` attribute, which is set to `supplemental` by default in new Converged Application Server domains. You can use this listing as a basic template, modifying commands to access and modify the configuration MBeans as necessary.

Example 2–1 Template WLST Script for Accessing ConfigManagerRuntimeMBean

```
# Connect to the Administration Server
connect('weblogic','weblogic','t3://localhost:7001')
# Navigate to ConfigManagerRuntimeMBean and start an edit session.
custom()
cd('mydomain:Location=myserver,Name=sipserver,ServerRuntime=myserver,Type=ConfigMa
```

```
nagerRuntime')
cmo.startEdit()
# --MODIFY THIS SECTION AS NECESSARY--
# Edit SIP Servlet container configuration MBeans
cd('mydomain:DomainConfig=mydomain,Location=myserver,Name=myserver,SipServer=myser
ver,Type=Proxy')
set('RoutingPolicy','domain')
# Navigate to ConfigManagerRuntimeMBean and persist the configuration
# to sipserver.xml
cd('mydomain:Location=myserver,Name=sipserver,ServerRuntime=myserver,Type=ConfigMa
nagerRuntime')
cmo.activate()
```

Creating and Deleting MBeans

The `SipServer` MBean represents the entire contents of the **sipserver.xml** configuration file. In addition to having several attributes for configuring SIP timers and SIP application session timeouts, `SipServer` provides helper methods to help you create or delete MBeans representing proxy settings and overload protection controls.

Example 2–2 shows an example of how to use the helper commands to create and delete configuration MBeans that configuration elements in **sipserver.xml**. See also Example 2–3, "Invoking Helper Methods for Setting URI Attributes" for a listing of other helper methods in `SipServer`, or refer to the Converged Application Server JavaDocs.

Example 2–2 WLST Commands for Creating and Deleting MBeans

```
connect()
custom()
cd('mydomain:Location=myserver,Name=sipserver,ServerRuntime=myserver,Type=ConfigMa
nagerRuntime')
cmo.startEdit()
cd('mydomain:DomainConfig=mydomain,Location=myserver,Name=sipserver,ServerRuntime=
myserver,Type=SipServer')
cmo.destroyOverloadProtection()
cmo.createProxy()
cd('mydomain:Location=myserver,Name=sipserver,ServerRuntime=myserver,Type=ConfigMa
nagerRuntime')
cmo.save()
```

Working with URI Values

Configuration MBeans such as `Proxy` require URI objects passed as attribute values. Oracle provides a helper class, `com.bea.wcp.sip.util.URIHelper`, to help you easily generate URI objects from an array of Strings. Example 2–3 modifies the sample shown in Example 2–2 to add a new URI attribute to the `LoadBalancer` MBean. See also the Converged Application Server JavaDocs for a full reference to the `URIHelper` class.

Example 2–3 Invoking Helper Methods for Setting URI Attributes

```
# Import helper method for converting strings to URIs.
from com.bea.wcp.sip.util.URIHelper import stringToSipURIs
connect()
custom()
cd('mydomain:Location=myserver,Name=sipserver,ServerRuntime=myserver,Type=ConfigMa
nagerRuntime')
cmo.startEdit()
cd('mydomain:DomainConfig=mydomain,Location=myserver,Name=sipserver,Type=SipServer
')
```

```
cmo.createProxy()
cd('mydomain:DomainConfig=mydomain,Location=myserver,Name=sipserver,SipServer=sips
erver,Type=Proxy')
stringarg =
jarray.array([java.lang.String("sip://siplb.bea.com:5060")],java.lang.String)
uriarg = stringToSipURIs(stringarg)
set('ProxyURIs',uriarg)
cd('mydomain:Location=myserver,Name=sipserver,ServerRuntime=myserver,Type=ConfigMa
nagerRuntime')
cmo.save()
```

Setting Logging Levels

The Converged Application Server is subject to the common configuration settings defined for WebLogic servers. To modify the logging settings for a Converged Application Server in the Administration Console, access the logging configuration settings page as follows:

1. Expand the **Environment** node in the Domain Structure tree.
2. Click **Servers**.
3. In the right pane, click the **Logging** tab.
4. Modify the default logging settings and then click **Save** to commit your changes.

Alternatively, use the **logging.xml** WebLogic file to manually configure logging properties for the servers.

Converged Application Server supports additional logging features that provide for SIP message logging. SIP message logging should be enabled in development environments only. It is not intended for production environments.

Configure SIP message logging as follows:

1. Expand the **SipServer** node in the Domain Structure tree.
2. In the Configuration tab, click the **Message Debug** subtab.
3. Select the **Enable Debug** checkbox.
4. Configure other message logging settings as needed. Other settings include the logging verbosity level, the log entry pattern, and the target log file name. See the onscreen field description for more information.
5. Click **Save** to commit your changes.

See "Logging SIP Requests and Responses" for information about creating custom log listeners and more information about logging settings.

Startup Sequence for a Converged Application Server Domain

Converged Application Server start scripts use default values for many 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. See "Modifying JVM Parameters in Server Start Scripts" in the chapter "Tuning JVM Garbage Collection for Production Deployments" for suggestions about maximizing JVM performance in a production domain.

Caution: When you configure a domain with multiple engine and SIP data tier servers, you must accurately synchronize all system clocks to a common time source (to within one or two milliseconds) in order for the SIP protocol stack to function properly. See "Configuring NTP for Accurate SIP Timers" in Chapter 3, "Configuring Converged Application Container Properties" for more information.

Because a typical Converged Application Server domain contains numerous engine and SIP data tier servers, with dependencies between the different server types, you should generally follow this sequence when starting up a domain:

1. **Start the Administration Server for the domain.** Start the Administration Server in order to provide the initial configuration to engine and SIP data 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 the `startWebLogic.sh` or `startWebLogic.cmd` script (depending on your OS) installed with the Configuration Wizard, or a custom startup script.
2. **Start SIP data tier servers in each partition.** The engine tier cannot function until servers in the SIP data tier are available to manage call state data. Although all replicas in each partition need not be available to begin processing requests, at least one replica in each configured partition must be available in order to manage the concurrent call state. All replicas should be started and available before opening the system to production network traffic.

   You generally start each SIP data tier server by using either the `startManagedWebLogic.cmd` script installed with the Configuration Wizard, or a custom startup script. `startManagedWebLogic.cmd` requires that you specify the name of the server to startup, as well as the URL of the Administration Server for the domain, as in:

   ```
   startManagedWebLogic.cmd datanode0-0 t3://adminhost:7001
   ```

3. **Start engine tier servers.** After the SIP data tier servers have started, you can start servers in the engine tier and begin processing client requests. As with SIP data tier servers, engine tier servers are generally started using the `startManagedWebLogic.cmd` script or a custom startup script.

Following the above startup sequence ensures that all Managed Servers use the latest SIP Servlet container and SIP data tier configuration. This sequence also avoids engine tier error messages that are generated when servers in the SIP data tier are unavailable.

Startup Command Options

Table 2–4 lists startup options available to Oracle Communications Converged Application Server and other Converged Application Server utilities. For more information about these and other options, see *Command Reference for Oracle WebLogic Server* in the Oracle WebLogic Server 11*g* documentation.

Table 2–4 Startup Command Options

Application	Startup Option	For More Information
SIP Servlet Application Router	-Djavax.servlet.sip.dar.configuration	See "Using the Default Application Router" in "Composing SIP Applications" in *Converged Application Server Developer's Guide*.
SIP Servlet Application Router	-Djavax.servlet.sip.ar.spi.SipApplicationRouterProvider	See "Configuring a Custom Application Router" in "Composing SIP Applications" in *Oracle Communications Converged Application Server Developer's Guide*.
Converged Application Server	–Dwlss.dialog.index.enabled	See "Join and Replaces Header Support" in *Oracle Communications Converged Application Server Developer's Guide*.
Converged Application Server	-Dwlss.local.serialization	See "Optimizing Memory Utilization and Performance with Serialization" in *Oracle Communications Converged Application Server Developer's Guide*.
Converged Application Server	-Dwlss.udp.listen.on.ephemeral	See information about single-NIC configurations with TCP and UDP channels in *Oracle Communications Converged Application Server Concepts*.
Converged Application Server	-Dwlss.udp.lb.masquerade	See information about network address translation (NAT) options in *Oracle Communications Converged Application Server Concepts*.
Converged Application Server	-Dweblogic.management.discover	See "Restarting an Administration Server on the Same Machine" in Chapter 18, "Avoiding and Recovering From Server Failures."
Converged Application Server	-Dweblogic.RootDirectory	See "Restarting an Administration Server on Another Machine" in Chapter 18, "Avoiding and Recovering From Server Failures."
Installer	-Djava.io.tmpdir	See the discussion on temporary disk space requirements in *Installation Guide for Oracle WebLogic Server* in the WebLogic Server 11*g* documentation.
WlssEchoServer	-Dwlss.ha.echoserver.port	See "Starting WlssEchoServer on SIP Data Tier Server Machines" in Chapter 6, "Configuring Server Failure Detection."
WlssEchoServer	-Dwlss.ha.echoserver.logfile	See "Starting WlssEchoServer on SIP Data Tier Server Machines" in Chapter 6, "Configuring Server Failure Detection."
WlssEchoServer	-Dreplica.host.monitor.enabled	See "Enabling and Configuring the Heartbeat Mechanism on Servers" in Chapter 6, "Configuring Server Failure Detection."

Table 2–4 (Cont.) Startup Command Options

Application	Startup Option	For More Information
WlssEchoServer	-Dwlss.ha.heartbeat.interval	See "Enabling and Configuring the Heartbeat Mechanism on Servers" in Chapter 6, "Configuring Server Failure Detection."
WlssEchoServer	-Dwlss.ha.heartbeat.count	See "Enabling and Configuring the Heartbeat Mechanism on Servers" in Chapter 6, "Configuring Server Failure Detection."
WlssEchoServer	-Dwlss.ha.heartbeat.SoTimeout	See "Enabling and Configuring the Heartbeat Mechanism on Servers" in Chapter 6, "Configuring Server Failure Detection."

Reverting to the Original Boot Configuration

When you boot the Administration Server for a Converged Application Server domain, the server creates parses the current container configuration in **sipserver.xml**, and generates a copy of the initial configuration in a file named **sipserver.xml.booted** in the **config/custom** subdirectory of the domain directory. This backup copy of the initial configuration is preserved until you next boot the server; modifying the configuration using JMX does not affect the backup copy.

If you modify the SIP Servlet container configuration and later decide to roll back the changes, copy the **sipserver.xml.booted** file over the current **sipserver.xml** file. Then reboot the server to apply the new configuration.

3 Configuring Converged Application Container Properties

This chapter describes how to configure SIP container features in the engine tier of an Oracle Communications Converged Application Server deployment:

Creating the Engine Tier Cluster

In most implementations, the Converged Application Server instances are grouped into clusters. You create clusters and servers for the Converged Application Server in the same manner as other WebLogic servers. In the Administration Console, access the Environment page in the Domain Structure tree.

When configuring clusters for production deployments of the Converged Application Server, Oracle recommends that you use multicast cluster communication instead of unicast communication. While unicast may be easier to configure, using multicast reduces the likelihood of performance degradation that may result from excessive work load on group leaders and retransmissions during periods of high traffic.

After defining servers and clusters, you associate the cluster with the Converged Application Server as follows:

1. In a browser, open the Administration Console for your domain.
2. Click the **SipServer** link in the Domain Structure pane.

   The General configuration page appears, which shows two top-level tabs, Configuration and Monitoring.
3. In the Configuration tab, click the **Targets** subtab.
4. In the **SIP Server Targets** field, enter the name of the cluster that contains the servers you want to act as Converged Application Server instances.
5. Click **Save** to save your configuration changes.

For more information on clustering, see *Oracle Fusion Middleware Using Clusters for Oracle WebLogic Server*.

Configure General SIP Application Server Properties

Loading SIP applications to the Converged Application Server in the Administration Console is similar to loading any application to WebLogic server. You use the Deployments page in the Administration Console to load, update, or remove an application or module.

The Converged Application Server defines general settings that apply to all SIP applications. Before deploying applications to the Converged Application Server, you should verify and modify the default values for the general settings. You can configure the general settings in the SIP Server page of the Administration Console.

To configure general SIP application server properties:

1. Open the Administration Console for your domain.
2. Click the **SipServer** link in the Domain Structure pane.

   The right pane of the console provides two levels of tabbed pages that are used for configuring and monitoring Converged Application Server. By default, the General configuration page appears.
3. Use the fields in the **General** subtab of the Configuration tab to configure the general settings applicable to serving SIP applications.

   Among the settings that determine common application handling are:

   - The default servlet invoked if a specific servlet is not identified for a request based on the servlet mapping rules.
   - Timer values. See "Configuring Timer Processing" for more information.
   - Header handling settings.
   - Application session settings.

   For details, see the onscreen field descriptions in the Administration Console.
4. Click **Save** to save your configuration changes.
5. Click **Activate Changes** to apply your changes to the engine tier servers.

Configuring Timer Processing

As engine tier servers add new call state data to the SIP data tier, SIP data tier instances queue and maintain the complete list of SIP protocol timers and application timers associated with each call. Engine tier servers periodically poll partitions in the SIP data tier to determine which timers have expired, given the current time. By default, multiple engine tier polls to the SIP data tier are staggered to avoid contention on the timer tables. Engine tier servers then process all expired timers using threads allocated in the `sip.timer.Default` execute queue.

Configuring Timer Affinity (Optional)

With the default timer processing mechanism, a given engine tier server processes all timers that are currently due to fire, regardless of whether or not that engine was involved in processing the calls associated with those timers. However, some deployment scenarios require that a timer is processed on the same engine server that last modified the call associated with that timer. One example of this scenario is a hot standby system that maintains a secondary engine that should not process any call data until another engine fails. Converged Application Server enables you to configure timer affinity in such scenarios.

When you enable timer affinity, replicas request that each engine tier server periodically poll the SIP data tier for processed timers. When polling the SIP data tier, an engine processes only those timers associated with calls that were last modified by that engine, or timers for calls that have no owner.

Note: When an engine tier server fails, any call states that were last modified by that engine no longer have an owner. Expired timers that have no owner are processed by the next engine server that polls the SIP data tier.

To enable timer affinity:

1. Access the Administration Console for your domain.
2. Select the SipServer node in the left pane. The right pane of the console provides two levels of tabbed pages that are used for configuring and monitoring Converged Application Server.
3. Select the Configuration > General tab in the right pane.
4. Select Enable Timer Affinity.
5. Click Save to save your configuration changes.
6. Click Activate Changes to apply your changes to the engine tier servers.

The Enable Timer Affinity setting is persisted in **sipserver.xml** in the `enable-timer-affinity` element.

Configuring NTP for Accurate SIP Timers

In order for the SIP protocol stack to function properly, all engine and SIP data tier servers must accurately synchronize their system clocks to a common time source, to within one or two milliseconds. Large differences in system clocks cause a number of severe problems such as:

- SIP timers firing prematurely on servers with fast clock settings.
- Poor distribution of timer processing in the engine tier. For example, one engine tier server may process all expired timers, whereas other engine tier servers process no timers.

Oracle recommends using a Network Time Protocol (NTP) client or daemon on each Converged Application Server instance and synchronizing to a common NTP server.

Caution: You must accurately synchronize server system clocks to a common time source (to within one or two milliseconds) in order for the SIP protocol stack to function properly. Because the initial T1 timer value of 500 milliseconds controls the retransmission interval for INVITE request and responses, and also sets the initial values of other timers, even small differences in system clock settings can cause improper SIP protocol behavior. For example, an engine tier server with a system clock 250 milliseconds faster than other servers will process more expired timers than other engine tier servers, will cause retransmits to begin in half the allotted time, and may force messages to timeout prematurely.

Configuring the JSR 309 Media Server Control Driver

Converged Application Server allows you to control third-party media servers using a Media Server Control API based on JSR 309, a standard Java interface. The JSR 309 specification defines a protocol agnostic API for controlling media servers. JSR 309 provides a portable interface to create media rich applications with interactive voice response (IVR), conferencing, speech recognition, and similar features. Vendors of IP media servers provide JSR 309 based driver implementations that work with their IP media servers.

See "Media Server Control" in the *Converged Application Server Technical Product Description* for more information about the Media Server Control feature. For information about JSR 309, see the JSR 309 specification at:

`http://jcp.org/en/jsr/detail?id=309`

To enable your converged SipServlet-Java EE applications deployed on the Converged Application Server to communicate with the media server using the Media Server Control API, you must configure Converged Application Server to recognize the custom, JSR 309 resource; install the Media Server Control driver; and configure the driver using the Administration Console.

Install the Media Server Control Driver

For a standalone deployment of Converged Application Server, you must to install the Media Server Control driver on the Administration Server. For a replicated deployment, you must to install the Media Server Control driver on each engine in the Engine tier cluster.

Follow these steps to install the Media Server Control driver:

1. Shutdown all servers in the domain.
2. Copy the Media Server Control JAR file(s) to share into a `lib` subdirectory of the domain directory. For example:

   CAS_home`\user_projects\domains\`*domain_name*`\lib\`

   where *CAS_home* is the root directory where you installed Converged Application Server.

   Note: The Administration Server does not automatically distribute Media Server Control drivers to Managed Servers on remote machines. If you have Managed Servers that do not share the same physical domain directory as the Administration Server, you must manually install Media Server Control drivers to the CLASSPATH of each Managed Server.

   Alternatively, you can add the JAR file to the system or Converged Application Server classpath.

   See the discussion on adding JARs to the system classpath in *Developing Applications with WebLogic Server* for more information about adding JAR files to the system or Converged Application Server classpath.

1. Start the Administration Server and all Managed Servers in the domain. Converged Application Server appends JAR files found in the `lib` directory to the system classpath.

2. When the Converged Application Server starts, you can configure the installed driver(s) using the Converged Application Server Administration Console. See "Configure the Media Server Control Factory" for more information.

Configure the JSR 309 Resource for the Media Server Control Driver

The Media Server Control driver is not configured with a JSR 309 resource during the initial installation of the software. You must manually configure a custom JSR 309 resource for all servers in your Converged Application Server domain that will use the Media Server Control driver.

Configure the JSR 309 Resource for Proxy Registrar Domains

To configure the JSR 309 resource for deployments using Proxy Registrar domains:

1. Shutdown all servers in the domain.
2. Add the custom resource shown in the example below to the domain's **config.xml** file on the Administration Server. This example uses ***AdminServer*** as the target for the JSR309 resource. Specify the target parameters applicable to your Converged Application Server deployment.

```
<custom-resource>
<name>jsr309driveroam</name>
<target>AdminServer</target>
<descriptor-file-name>custom/jsr309driver.xml</descriptor-file-name>
<resource-class>
oracle.sdp.jsr309.configuration.resources.Jsr309Resource
</resource-class>
<descriptor-bean-class>
oracle.sdp.jsr309.configuration.beans.Jsr309ConfigurationBean
</descriptor-bean-class>
</custom-resource>
```

3. Start the Administration Server, and all instances of Managed Servers on which you will install a Media Server Control driver.
4. When the Converged Application Server starts, you can configure the installed driver(s) using the Converged Application Server Administration Console. See "Configure the Media Server Control Factory" for more information.

Configure the JSR 309 Resource for Administration and Replicated Domains

The **mscontrol.jar** and **jsr309-descriptor-binding.jar** JAR files are not added to the WebLogic CLASSPATH during installation. For this reason, you must manually add these JAR files to the CLASSPATH for your deployment. If you do not add these files to the CLASSPATH, the following exception is thrown during server start-up:

```
<Error> <netuix> <BEA-423168>
<An exception or error occurred in the backing file

[oracle.sdp.jsr309.console.util.JSR309NavTreeBac
kingFile] while executing its init method. It was

java.lang.NoClassDefFoundError
: oracle/sdp/jsr309/configuration/resources/Jsr309Resource
```

To add the **mscontrol.jar** and **jsr309-descriptor-binding.jar** JAR files to the WebLogic CLASSPATH:

1. Shutdown all servers in the domain.

2. Modify the CLASSPATH variable of the `startWebLogic` script as follows:

   For Windows-based operating systems:

   Modify the `startWebLogic.cmd` script located at *CAS_home*`\user_projects\domains\`*domain_name*`\bin` as follows:

   ```
   set
   CLASSPATH=%SAVE_CLASSPATH%;
   %ORCL_HOME%\server\modules\mscontrol.jar;%
   %ORCL_HOME%\server\lib\jsr309-descriptor-binding.jar
   ```

   For UNIX-based operating systems:

   Modify the `startWebLogic.sh` script located at *CAS_home*`/user_projects/domains/`*domain_name*`/bin` as follows:

   ```
   CLASSPATH="${SAVE_CLASSPATH}:
   ${ORCL_HOME}/server/modules/mscontrol.jar:
   ${ORCL_HOME}/server/lib/jsr309-descriptor-binding.jar"
   ```

3. Add the custom resource shown in the example below to the domain's **config.xml** file on the Administration Server. This example uses ***`AdminServer`*** as the target for the JSR309 custom resource. Specify the target parameters applicable to your Converged Application Server deployment.

   ```
   <custom-resource>
   <name>jsr309driveroam</name>
   <target>AdminServer</target>
   <descriptor-file-name>custom/jsr309driver.xml</descriptor-file-name>
   <resource-class>
   oracle.sdp.jsr309.configuration.resources.Jsr309Resource
   </resource-class>
   <descriptor-bean-class>
   oracle.sdp.jsr309.configuration.beans.Jsr309ConfigurationBean
   </descriptor-bean-class>
   </custom-resource>
   ```

4. Create an XML file called **jsr309driver.xml** with the following contents:

   ```
   <?xml version='1.0' encoding='UTF-8'?>
   <jsr309-configuration xmlns="http://www.oracle.com/ns/occas/sdp/jsr309/100"
   xmlns:sec="http://xmlns.oracle.com/weblogic/security"
   xmlns:wls="http://xmlns.oracle.com/weblogic/security/wls"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   </jsr309-configuration>
   ```

5. Save the **jsr309driver.xml** file to the directory *CAS_home*`/user_projects/domains/`*domain_name*`/config/custom` (UNIX) or *CAS_home*`\user_projects\domains\`*domain_name*`\config\custom` (Windows) on your domain's Administration Server.

6. Start the Administration Server, and all instances of Managed Servers on which you will install a Media Server Control driver.

7. When the Converged Application Server starts, you can configure the installed driver(s) using the Converged Application Server Administration Console. See "Configure the Media Server Control Factory" for more information.

Configure the Media Server Control Factory

Follow these steps to configure the Media Server Control factory:

1. Access the WebLogic Server Administration Console. Use your browser to access the URL:

   `http://`*address*`:`*port*`/console`

   where *address* is the Administration Server's listen address and *port* is the listen port.

2. Select **Media Server Control** in the Domain Structure navigation tree.

   The Media Server Control Factory page displays any previously configured Media Server Control drivers.

3. Click **New** to create a new Media Server Control Factory.

4. Fill in the fields of the Create a Media Server Control Factory page as follows:

   - **Factory name**: The name to assign to the Media Server Control factory.
   - **JNDI name**: The JNDI name under which to create the Media Server Control factory. The JNDI name uniquely identifies this instance of the driver.

     Observe the conventions of the Java Naming and Directory Interface (JNDI) service when you create JNDI factory names. The following naming conventions are supported for the JNDI name:

     – Only a-z, A-Z, 0-9,'_', '@' and '/' are valid characters for the JNDI name.

       All other characters are treated as invalid, and will cause the configuration to fail.

     – '/' is treated as a subcontext character. For example, the name `test1/test2/factory` will be treated as a JNDI hierarchy.

     When registering a JNDI name for the Media Server Control factory, Converged Application Server prepends the term `mscontrol` to the name. For example: `mscontrol/mediasrv1`

   - **Driver name**: Select a Media Server Control driver from the Driver list.

   Click **Next**.

5. The properties of the Media Server Control configuration are displayed in an editable text field. You can edit this field to add new properties, or you can modify or delete an existing property.

   The only property specified by the Media Server Control API is `MEDIA_SERVER_URI`, which defines the Uniform Resource Identifier (URI) of the media server. The URI is a string of characters used to identify a resource on the Internet. You can edit the `MEDIA_SERVER_URI` property with the URI of the media server to use in conjunction with Converged Application Server. While the `MEDIA_SERVER_URI` is an optional property, and may not be required by all media server vendors, Oracle recommends specifying a URI value whenever possible.

   Note: Certain properties defined by the `MsControlFactory` class are required to create a Media Server Control factory. If you delete a required property, the creation of the factory will fail.

6. Click **Finish**.

   A new Media Server Control factory instance that can be injected or looked up from inside your converged applications to communicate with the media server is created.

About Monitoring the Count of Application Sessions

The current count of active application sessions in a domain is displayed in the Administration Console under the SIP Server **Monitoring** tab, in the **General** subtab. The counters for created sessions and destroyed sessions on each engine-tier server are maintained in runtime MBeans on those servers locally and do not survive server restarts. To minimize the performance impact of data storage and serialization, a server's counters are not replicated to the data tier and hence are reset to zero each time the server is restarted.

The count of active sessions in a domain at any given time is the sum of sessions created on all engine-tier servers, less the sum of sessions destroyed on all engine-tier servers, from the time that the domain servers are started. This count is a best estimate and may not reflect the actual count of active sessions if servers are started at different times or when one or more servers are shut down.

For example, an engine-tier cluster contains server 1 and server 2. Server 2 is shut down. This causes the created- and destroyed-session counters on server 2 to be zero. Because sessions created on server 2 will eventually invalidate on server 1, the sum of destroyed sessions for the cluster-wide count will be greater than the sum of the created sessions for the cluster-wide count, resulting in a negative value for the active-session counter in the cluster. The total count will continue to be negative even if server 2 is restarted.

If you need to ensure that the domain-wide count of active application sessions is accurate at all times, you can use a custom, external JMX-based agent that maintains counters outside of the engine-tier servers.

4 Configuring SIP Data Tier Partitions and Replicas

This chapter describes how to configure Oracle Communications Converged Application Server instances that make up the SIP data tier cluster of a deployment:

Overview of SIP Data Tier Configuration

The Converged Application Server SIP data tier is a cluster of server instances that manages the application call state for concurrent SIP calls. The SIP data tier may manage a single copy of the call state or multiple copies as needed to ensure that call state data is not lost if a server machine fails or network connections are interrupted.

The SIP data tier cluster is arranged into one or more *partitions*. A partition consists of one or more SIP data tier server instances that manage the same portion of concurrent call state data. In a single-server Converged Application Server installation, or in a two-server installation where one server resides in the engine tier and one resides in the SIP data tier, all call state data is maintained in a single partition. Multiple partitions are required when the size of the concurrent call state exceeds the maximum size that can be managed by a single server instance. When more than one partition is used, the concurrent call state is split among the partitions, and each partition manages an separate portion of the data. For example, with a two-partition SIP data tier, one partition manages the call state for half of the concurrent calls (for example, calls A through M) while the second partition manages the remaining calls (N through Z).

In most cases, the maximum call state size that can be managed by an individual server corresponds to the Java Virtual Machine limit of approximately 1.6GB per server.

Additional servers can be added within the same partition to manage copies of the call state data. When multiple servers are members of the same partition, each server manages a copy of the same portion of the call data, referred to as a *replica* of the call state. If a server in a partition fails or cannot be contacted due to a network failure, another replica in the partition supplies the call state data to the engine tier. Oracle recommends configuring two servers in each partition for production installations, to guard against machine or network failures. A partition can have a maximum of three replicas for providing additional redundancy.

datatier.xml Configuration File

The **datatier.xml** configuration file, located in the **config/custom** subdirectory of the domain directory, identifies SIP data tier servers and also defines the partitions and replicas used to manage the call state. If a server's name is present in **datatier.xml**, that server loads Converged Application Server SIP data tier functionality at boot time. (Server names that do not appear in **datatier.xml** act as engine tier nodes, and instead provide SIP Servlet container functionality configured by the **sipserver.xml** configuration file.)

The sections that follow show examples of the **datatier.xml** contents for common SIP data tier configurations. See Chapter 22, "SIP Data Tier Configuration Reference (datatier.xml)" for complete information about the XML Schema and its elements.

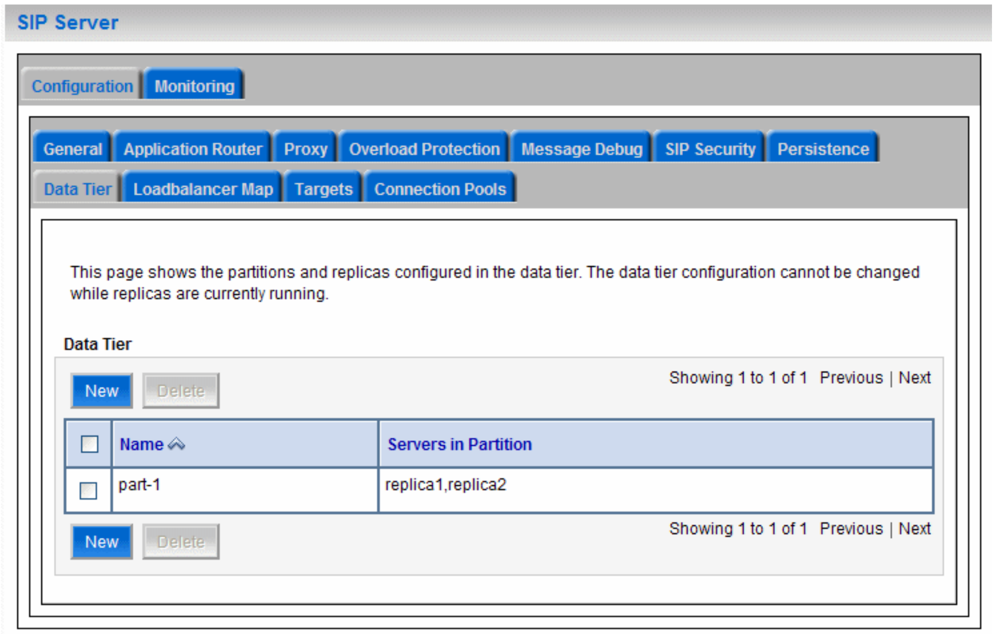
Configuration Requirements and Restrictions

All servers that participate in the SIP data tier should be members of the same WebLogic Server cluster. The cluster configuration enables each server to monitor the status of other servers. Using a cluster also enables you to easily target the **sipserver** and **datatier** custom resources to all servers for deployment.

For high reliability, you can configure up to three replicas within a partition.

You cannot change the SIP data tier configuration while replicas or engine tier nodes are running. You must restart servers in the domain in order to change SIP data tier membership or reconfigure partitions or replicas.

To view and configure the current SIP data tier configuration in the Administration Console, click the **SipServer** node, click the **Configuration** tab, and then click the **Data Tier** tab. The SIP data tier configuration page is shown in the following figure.



This page shows the partitions and replicas configured in the data tier. The data tier configuration cannot be changed while replicas are currently running.

Data Tier

Showing 1 to 1 of 1 Previous | Next

☐	Name	Servers in Partition
☐	part-1	replica1,replica2

Showing 1 to 1 of 1 Previous | Next

Best Practices for Configuring and Managing SIP Data Tier Servers

Adding replicas can increase reliability for the system as a whole, however, each additional server in a partition requires additional network bandwidth to manage the replicated data. With three replicas in a partition, each transaction that modifies the call state updates data on three different servers.

To ensure high reliability when using replicas, always ensure that server instances in the same partition reside on different machines. Hosting two or more replicas on the same machine leaves all of the hosted replicas vulnerable to a machine or network failure.

SIP data tier servers can have one of three different statuses:

- **ONLINE**: indicates that the server is available for managing call state transactions.
- **OFFLINE**: indicates that the server is shut down or unavailable.
- **ONLINE_LOCK_AUTHORITY_ONLY**: indicates that the server was rebooted and is currently being updated (from other replicas) with the current call state data. A recovering server cannot yet process call state transactions, because it does not maintain a full copy of the call state managed by the partition.

If you need to take a SIP data tier server instance offline for scheduled maintenance, make sure that at least one other server in the same partition is active. If you shut down an active server and all other servers in the partition are offline or recovering, you will lose a portion of the active call state.

In a Converged Application Server installation with a set of engine tier nodes and tier nodes in a partition, if the connection to the SIP data tier becomes “split” such that each engine tier server can only reach a different SIP data tier node, one of the replicas is forced offline.

To recover from this situation, always configure the Node Manager utility to restart SIP data tier replicas automatically when a replica fails. This enables the replica to rejoin its associated partition and update its copy of the call state data without having to manually restart the server.

Converged Application Server automatically divides the call state evenly over all configured partitions.

When configuring the data tier cluster, you have the option of choosing unicast or multicast as the mechanism by which servers in a cluster communicate with each other. While unicast may be easier to configure, for most production deployments, Oracle recommends use of multicast communication. Using multicast reduces the likelihood of performance degradation that may result from excessive work load on group leaders and retransmissions during periods of high traffic. For more information on clustering, see *Oracle Fusion Middleware Using Clusters for Oracle WebLogic Server*.

If you configure two or more SIP data tier replicas using the default WebLogic Server Listen Address configuration (which specifies no listen address), multiple SIP data tier instances on the same machine cannot connect to one another. This occurs because, using the default Listen Address configuration, JNDI objects in the first booted server bind to all local IP addresses.

To avoid this problem, always enter a valid IP address for each configured SIP data tier server instance.

Example SIP Data Tier Configurations and Configuration Files

The sections that follow describe some common Converged Application Server installations that utilize a separate SIP data tier.

SIP Data Tier with One Partition

A single-partition, single-server SIP data tier represents the simplest data tier configuration. Example 4–1 shows a SIP data tier configuration for a single-server deployment.

Example 4–1 SIP Data Tier Configuration for Small Deployment

```
<?xml version="1.0" encoding="UTF-8"?>
  <data-tier xmlns="http://www.bea.com/ns/wlcp/wlss/300">
    <partition>
      <name>part-1</name>
      <server-name>replica1</server-name>
    </partition>
  </data-tier>
```

To add a replica to an existing partition, define a second **server-name** entry in the same partition. For example, the **datatier.xml** configuration file shown in Example 4–2 recreates a two-replica configuration.

Example 4–2 SIP Data Tier Configuration for Small Deployment with Replication

```
<?xml version="1.0" encoding="UTF-8"?>
  <data-tier xmlns="http://www.bea.com/ns/wlcp/wlss/300">
    <partition>
      <name>Partition0</name>
      <server-name>DataNode0-0</server-name>
      <server-name>DataNode0-1</server-name>
    </partition>
  </data-tier>
```

SIP Data Tier with Two Partitions

Multiple partitions can be easily created by defining multiple **partition** entries in **datatier.xml**, as shown in Example 4–3.

Example 4–3 Two-Partition SIP Data Tier Configuration

```
<?xml version="1.0" encoding="UTF-8"?>
  <data-tier xmlns="http://www.bea.com/ns/wlcp/wlss/300">
    <partition>
      <name>Partition0</name>
      <server-name>DataNode0-0</server-name>
    </partition>
    <partition>
      <name>Partition1</name>
      <server-name>DataNode1-0</server-name>
    </partition>
  </data-tier>
```

SIP Data Tier with Two Partitions and Two Replicas

Replicas of the call state can be added by defining multiple SIP data tier servers in each partition. Example 4–4 shows the **datatier.xml** configuration file used to define a system having two partitions with two servers (replicas) in each partition.

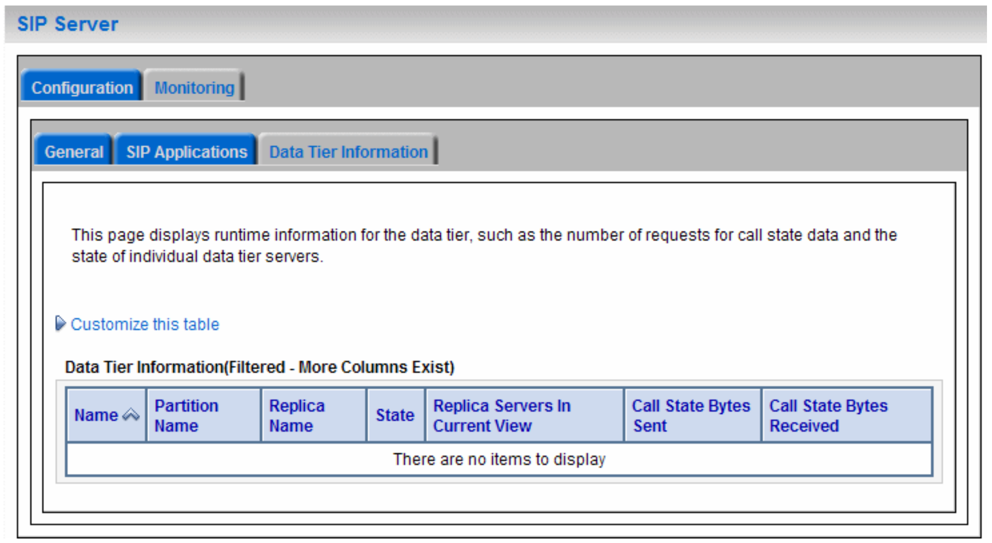
Example 4–4 SIP Data Tier Configuration for Small Deployment

```
<?xml version="1.0" encoding="UTF-8"?>
  <data-tier xmlns="http://www.bea.com/ns/wlcp/wlss/300">
    <partition>
      <name>Partition0</name>
      <server-name>DataNode0-0</server-name>
      <server-name>DataNode0-1</server-name>
    </partition>
    <partition>
      <name>Partition1</name>
      <server-name>DataNode1-0</server-name>
      <server-name>DataNode1-1</server-name>
    </partition>
  </data-tier>
```

Monitoring and Troubleshooting SIP Data Tier Servers

A runtime MBean, **ReplicaRuntimeMBean**, provides valuable information about the current state and configuration of the SIP data tier. See the Converged Application Server JavaDocs for a description of the attributes provided in this MBean.

Many of the attributes can be viewed in the Administration Console by navigating to the **SipServer** node, clicking the **Monitoring** tab, and then clicking the **Data Tier Information** tab, as shown in the following figure.



This page displays runtime information for the data tier, such as the number of requests for call state data and the state of individual data tier servers.

Data Tier Information(Filtered - More Columns Exist)

Name	Partition Name	Replica Name	State	Replica Servers In Current View	Call State Bytes Sent	Call State Bytes Received
There are no items to display						

Example 4–5 shows a simple WLST session that queries the current attributes of a single Managed Server instance in a SIP data tier partition. Table 4–1 (following the example) describes the MBean services in more detail.

Example 4–5 Displaying ReplicaRuntimeMBean Attributes

```
connect('weblogic','weblogic','t3://datahost1:7001')
custom()
cd('com.bea')
cd('com.bea:ServerRuntime=replica1,Name=replica1,Type=ReplicaRuntime')
ls()
-rw-   BackupStoreInboundStatistics                 null
-rw-   BackupStoreOutboundStatistics                null
-rw-   BytesReceived                                0
-rw-   BytesSent                                    0
-rw-   CurrentViewId                                2
-rw-   DataItemCount                                0
-rw-   DataItemsToRecover                           0
-rw-   DatabaseStoreStatistics                      null
-rw-   HighKeyCount                                 0
-rw-   HighTotalBytes                               0
-rw-   KeyCount                                     0
-rw-   Name                                         replica1
-rw-   Parent                                       com.bea:Name=replica1,Type=S
erverRuntime
-rw-   PartitionId                                  0
-rw-   PartitionName                                part-1
-rw-   ReplicaId                                    0
-rw-   ReplicaName                                  replica1
-rw-   ReplicaServersInCurrentView                  java.lang.String[replica1,
replica2]
-rw-   ReplicasInCurrentView                        [I@75378c
-rw-   State                                        ONLINE
-rw-   TimerQueueSize                               0
-rw-   TotalBytes                                   0
-rw-   Type                                         ReplicaRuntime
```

Table 4–1 ReplicaRuntimeMBean Method and Attribute Summary

Method/Attribute	Description
`dumpState()`	Records the entire state of the selected SIP data tier server instance to the Converged Application Server log file. You may want to use the `dumpState()` method to provide additional diagnostic information to a Technical Support representative in the event of a problem.
`BackupStoreInboundStatistics`	Provides statistics about call state data replicated from a remote geographical site.
`BackupStoreOutboundStatistics`	Provides statistics about call state data replicated to a remote geographical site.
`BytesReceived`	The total number of bytes received by this SIP data tier server. Bytes are received as servers in the engine tier provide call state data to be stored.
`BytesSent`	The total number of bytes sent from this SIP data tier server. Bytes are sent to engine tier servers when requested to provide the stored call state.
`CurrentViewId`	The current view ID. Each time the layout of the SIP data tier changes, the view ID is incremented. For example, as multiple servers in a SIP data tier cluster are started for the first time, the view ID is incremented when each server begins participating in the SIP data tier. Similarly, the view is incremented if a server is removed from the SIP data tier, either intentionally or due to a failure.
`DataItemCount`	The total number of stored call state keys for which this server has data. This attribute may be lower than the `KeyCount` attribute if the server is currently recovering data.

Table 4–1 (Cont.) ReplicaRuntimeMBean Method and Attribute Summary

Method/Attribute	Description
`DataItemsToRecover`	The total number of call state keys that must still be recovered from other replicas in the partition. A SIP data tier server may recover keys when it has been taken offline for maintenance and is then restarted to join the partition.
`HighKeyCount`	The highest total number of call state keys that have been managed by this server since the server was started.
`HighTotalBytes`	The highest total number of bytes occupied by call state data that this server has managed since the server was started.
`KeyCount`	The number of call data keys that are stored on the replica.
`PartitionId`	The numerical partition ID (from 0 to 7) of this server's partition.
`PartitionName`	The name of this server's partition.
`ReplicaId`	The numerical replica ID (from 0 to 2) of this server's replica.
`ReplicaName`	The name of this server's replica.
`ReplicaServersInCurrentView`	The names of other Converged Application Server instances that are participating in the partition.
`State`	The current state of the replica. SIP data tier servers can have one of three statuses: ■ **ONLINE**: indicates that the server is available for managing call state transactions. ■ **OFFLINE**: indicates that the server is shut down or unavailable. ■ **ONLINE_LOCK_AUTHORITY_ONLY**: indicates that the server was rebooted and is currently being updated (from other replicas) with the current call state data. A recovering server cannot yet process call state transactions, because it does not maintain a full copy of the call state managed by the partition.
`TimerQueueSize`	The current number of timers queued on the SIP data tier server. This generally corresponds to the KeyCount value, but may be less if new call states are being added but their associated timers have not yet been queued. **Note:** Engine tier servers periodically check with SIP data tier instances to determine if timers associated with a call have expired. In order for SIP timers to function properly, all engine tier servers must actively synchronize their system clocks to a common time source. Oracle recommends using a Network Time Protocol (NTP) client or daemon on each engine tier instance and synchronizing to a selected NTP server. See "Configuring Timer Processing".
`TotalBytes`	The total number of bytes consumed by the call state managed in this server.

5

Configuring Network Connection Settings

This chapter describes how to configure network resources for use with Oracle Communications Converged Application Server.

Overview of Network Configuration

The default HTTP network configuration for each Converged Application Server instance is determined from the Listen Address and Listen Port setting for each server. However, Converged Application Server does not support the SIP protocol over HTTP. The SIP protocol is supported over the UDP and TCP transport protocols. SIPS is also supported using the TLS transport protocol.

To enable UDP, TCP, or TLS transports, you configure one or more *network channels* for a Converged Application Server instance. A network channel is a configurable Oracle WebLogic Server resource that defines the attributes of a specific network connection to the server instance. Basic channel attributes include:

- The protocols supported by the connection
- The listen address (DNS name or IP address) of the connection
- The port number used by the connection
- (optional) The port number used by outgoing UDP packets
- The public listen address to embed in SIP headers when the channel is used for an outbound connection. This is typically the IP address presented by the IP sprayer or external load balancer as the virtual IP (VIP) for the telecommunication services.

You can assign multiple channels to a single Converged Application Server instance to support multiple protocols or to utilize multiple interfaces available with multihomed server hardware. You cannot assign the same channel to multiple server instances.

When you configure a new network channel for the SIP protocol, both the UDP and TCP transport protocols are enabled on the specified port. You cannot create a SIP channel that supports only UDP transport or only TCP transport. When you configure a network channel for the SIPS protocol, the server uses the TLS transport protocol for the connection.

As you configure a new SIP Server domain, you will generally create multiple SIP channels for communication to each engine tier server in your system. Engine tier servers can communicate to SIP data tier replicas using the configured Listen Address attributes for the replicas. Note, however, that replicas must use unique Listen Addressees in order to communicate with one another.

Note: If you configure multiple replicas in a SIP data tier cluster, you must configure a unique Listen Address for each server (a unique DNS name or IP address). If you do not specify a unique Listen Address, the replica service binds to the default *localhost* address and multiple replicas cannot locate one another.

Configuring External IP Addresses in Network Channels

When you set up a network channel for your Converged Application Server instance, you must specify the public IP address that external clients use to address the instance. In most cases, this address is presented by an IP sprayer or external load balancer or other network element capable of exposing a virtual IP (VIP) on behalf of the Converged Application Server to the external network.

You configure the client-facing address as the external listen address. When a SIP channel has an external listen address that differs from the channel's primary listen address, Converged Application Server embeds the host and port number of the external address in SIP headers, such as in the Response header. This causes subsequent messages from external clients to be directed to the public address rather than the local engine tier server address (which may not be accessible to clients).

If an external listen address is not specified for the network channel, the Converged Application Server embeds the primary listen address for the channel in the headers.

If you have more than one IP sprayer or load balancer that may receive external traffic addressed to the Converged Application Servers, you must define a channel on each engine tier server for each one. When a particular network interface on the engine tier server is selected for outbound traffic, the network channel associated with the network interface card’s (NIC's) address is examined to determine the external listen address to embed in SIP headers.

If your system uses a multihomed IP sprayer or load balancer having two public addresses, you must also define a pair of channels to configure both public addresses. If the engine tier server has only one NIC, you must define a second, logical address on the NIC to configure a dedicated channel for the second public address. In addition, you must configure your IP routing policies to define which logical address is associated with each public address.

About IPv4 and IPv6 Support

If your operating system and hardware support IPv6, you can also configure Converged Application Server to use IPv6 for network communication. IPv6 for SIP traffic is enabled by configuring a network channel with an IPv6 address. You must configure an IPv6 SIP channel on each engine tier server that will support IPv6 traffic.

Note that each SIP network channel configured on an engine supports either IPv6 or IPv4 traffic. You cannot mix IPv4 and IPv6 traffic on a single channel. A single engine can be configured with both an IPv4 and IPv6 channel to support multiple, separate networks.

It is also possible for Converged Application Server engine and SIP data tier nodes to communicate on IPv4 (or IPv6) while supporting the other protocol version for external SIP traffic. To configure engine and SIP data tier nodes on an IPv6 network, simply specify IPv6 listen addresses for each server instance.

Enabling DNS Support

Converged Application Server supports DNS for resolving the transport, IP address and port number of a proxy required to send a SIP message. This matches the behavior described in RFC 3263 (`http://www.ietf.org/rfc/rfc3263.txt`). DNS may also be used when routing responses in order to resolve the IP address and port number of a destination.

Caution: Because multihome resolution is performed within the context of SIP message processing, any multihome performance problems result in increased latency performance. Oracle recommends using a caching multihome server in a production environment to minimize potential performance problems.

To configure DNS support:

1. Log in to the Administration Console for the Converged Application Server domain you want to configure.
2. Select the **SipServer** node in the left pane of the Console.
3. Select the **Configuration**, and then select the **General** tab in the right pane.
4. Select **Enable DNS Server Lookup**.
5. Click **Save** to save your changes.

When you enable DNS lookup, the server can use DNS to:

- Discover a proxy server's transport, IP address, and port number when a request is sent to a SIP URI.
- Resolve an IP address and/or port number during response routing, depending on the contents of the Sent-by field.

For proxy discovery, Converged Application Server uses DNS resolution only once per SIP transaction to determine transport, IP, and port number information. All retransmissions, ACKs, or CANCEL requests are delivered to the same address and port using the same transport. For details about how DNS resolution takes place, see RFC 3263 (`http://www.ietf.org/rfc/rfc3263.txt`).

When a proxy is required to send a response message, Converged Application Server uses DNS lookup to determine the IP address and/or port number of the destination, depending on the information provided in the `sent-by` field and Via header.

Configuring Network Channels for SIP or SIPS

When you create a new domain using the Configuration Wizard, Converged Application Server instances are configured with a default network channel supporting the SIP protocol over UDP and TCP. This default channel is configured to use Listen Port 5060, but specifies no Listen Address. Follow the instructions in "Reconfiguring an Existing Channel" to change the default channel's listen address or listen port settings. See "Creating a New SIP or SIPS Channel" for information on creating a new channel resource to support additional protocols or additional network interfaces.

Reconfiguring an Existing Channel

You cannot change the protocol supported by an existing channel. To reconfigure an existing listen address/port combination to use a different network protocol, you must delete the existing channel and create a new channel using the instructions in "Creating a New SIP or SIPS Channel".

To configure a channel:

1. Log in to the Administration Console for the Converged Application Server domain you want to configure.
2. In the left pane, select the **Environment** entry to display its contents. Select **Servers** from the displayed entries.
3. In the right pane, select the name of the server you want to configure.
4. Select **Protocols**, then select the **Channels** tab to display the configured channels.
5. To delete an existing channel, select it in the table and click Delete.
6. To reconfigure an existing channel:
   a. Select the channel's name from the channel list (for example, the default `sip` channel).
   b. Edit the Listen Address or Listen Port fields to correspond to the address of a NIC or logical address on the associated engine tier machine.

      Note: The channel must be disabled before you can modify the listen address or listen port.

   c. Set the External Listen Address or External Listen Port fields to the destination address and port addressed by external clients. This is typically the VIP address presented by an external load balancer or IP sprayer in your system.
   d. Edit the advanced channel attributes as necessary (see "Creating a New SIP or SIPS Channel" for details.)
7. Click Save.

Creating a New SIP or SIPS Channel

To add a new SIP or SIPS channel to the configuration of a Converged Application Server instance:

1. Log in to the Administration Console for the Converged Application Server domain you want to configure.
2. In the left pane, select the **Environment** node, and then select the **Servers** tab.
3. In the right pane, select the name of the server you want to configure.
4. Select the **Protocols** tab, then select the **Channels** tab to display the configured channels.
5. Click New to configure a new channel.
6. Fill in the new channel fields as follows:
   - **Name:** Enter an administrative name for this channel, such as *SIPS-Channel-eth0*.
   - **Protocol:** Select either *SIP* to support UDP and TCP transport, or *SIPS* to support TLS transport. Note that a SIP channel cannot support only UDP or only TCP transport on the configured port.
7. Click **Next**
8. Fill in the new channel's addressing fields as follows:
   - **Listen Address:** Enter the IP address or DNS name for this channel. On a DNS machine, enter the exact IP address of the interface you want to configure, or a multihome name that maps to the exact IP address.
   - **Listen Port:** Enter the port number used to communication through this channel. The combination of Listen Address and Listen Port must be unique across all channels configured for the server. SIP channels support both UDP and TCP transport on the configured port.
   - **External Listen Address** and **External Listen Port**: Edit these fields to match the external address and port used by clients to address the system. This is typically a virtual IP address presented by an external load balancer or IP sprayer.

     If this value differs from the **Listen Address** value, the Converged Application Server embeds this value in SIP message headers for further call traffic.
9. Click **Next**.
10. Optionally click **Show** to display and edit advanced channel properties, such as connection timeout values. Be aware of the following restrictions and suggestions for advanced channel properties:
    - **Outbound Enabled**: This attribute cannot be unchecked, because all SIP and SIPS channels can originate network connections.
    - **HTTP Enabled for This Protocol**: This attribute cannot be selected for SIP and SIPS channels, because Converged Application Server does not support HTTP transport SIP protocols.
    - **Maximum Message Size**: This attribute specifies the maximum TCP message size that the server allows on a connection from this channel. Converged Application Server shuts off any connection where the messages size exceeds the configured value. The default size of 10,000,000 bytes is large. If you are concerned about preventing Denial Of Service (DOS) attacks against the

server, reduce this attribute to a value that is compatible with your deployed services.

11. Click **Finish**.

Configuring Custom Timeout, MTU, and Other Properties

SIP channels can be further configured using one or more custom channel properties. The custom properties cannot be set using the Administration Console. Instead, you must use a text editor to add the properties to a single, `custom-property` stanza in the channel configuration portion of the **config.xml** file for the domain.

Converged Application Server provides the following custom properties that affect the transport protocol of SIP channels:

- `TcpConnectTimeoutMillis`: Specifies the amount of time Converged Application Server waits before it declares a destination address (for an outbound TCP connection) as unreachable. The property is applicable only to SIP channels; Converged Application Server ignores this attribute value for SIPS channels. A value of 0 disables the timeout completely. A default value of 3000 milliseconds is used if you do not specify the custom property.
- `SctpConnectTimeoutMillis`: Specifies the amount of time Converged Application Server waits before it declares a destination address (for an outbound SCTP connection) as unreachable. The property is applicable only to SCTP channels (for Diameter traffic). A value of 0 disables the timeout completely. A default value of 3000 milliseconds is used if you do not specify the custom property. See "Configuring Static Source Port for Outbound UDP Packets" for information about creating SCTP channels for Diameter.
- `SourcePorts`: Configures one or more static port numbers that a server uses for originating UDP packets.

> **Caution:** Oracle does not recommend using the SourcePorts custom property in most configurations because it degrades performance. Configure the property only in cases where you must specify the exact ports that Converged Application Server uses to originate UDP packets.

- `Mtu`: Specifies the Maximum Transmission Unit (MTU) value for this channel. A value of -1 uses the default MTU size for the transport.
- EnabledProtocolVersions: Specifies the version of the SSL protocol to use with this channel when Converged Application Server acts as an SSL client. When acting as an SSL client, by default the channel requires TLS V1.0 as the supported protocol. The server can be configured to use SSL V3.0 as well, if that is the highest version that the SSL peer servers support. You can set one of the following values for this property:
  - `TLS1`, the default, configures the channel to send and accept only TLS V1.0 messages. Peers must respond with a TLS V1.0 message, or the SSL connection is dropped.
  - `SSL3` configures the channel to send and accept only SSL V3.0 messages. Peers must respond with an SSL V3.0 message, or the SSL connection is dropped.
  - `ALL` supports either TLS V1.0 or SSL V3.0 messages. Peers must respond with a TLS V1.0 or SSL V3.0 message, or the SSL connection is dropped.

To configure a custom property, use a text editor to modify the **config.xml** file directly, or use a JMX client such as WLST to add the custom property. When editing **config.xml** directly, ensure that you add only one `custom-properties` element to the end of a channel's configuration stanza. Separate multiple custom properties within the same element using semicolons (;) as shown in Example 5–1.

Example 5–1 Setting Custom Properties

```
<network-access-point>
  <name>sip</name>
  <protocol>sip</protocol>
  <listen-port>5060</listen-port>
  <public-port>5060</public-port>
  <http-enabled-for-this-protocol>false</http-enabled-for-this-protocol>
  <tunneling-enabled>false</tunneling-enabled>
  <outbound-enabled>true</outbound-enabled>
  <enabled>true</enabled>
  <two-way-ssl-enabled>false</two-way-ssl-enabled>
  <client-certificate-enforced>false</client-certificate-enforced>

<custom-properties>EnabledProtocolVersions=ALL;Mtu=1000;SourcePorts=5060</custom-p
roperties>
</network-access-point>
```

Configuring SIP Channels for Multihomed Machines

If you are configuring a server that has multiple network interfaces (a "multihomed" server), you must configure a separate network channel for each IP address used by Converged Application Server. Converged Application Server uses the listen address and listen port values for each channel when embedding routing information into SIP message system headers.

Note: If you do not configure a channel for a particular IP address on a multihomed machine, that IP address cannot be used when populating Via, Contact, and Record-Route headers.

Configuring TCP and TLS Channels for Diameter Support

The Converged Application Server Diameter implementation supports the Diameter protocol over the TCP or TLS transport protocols. To enable incoming Diameter connections on a server, you configure a dedicated network channel using the protocol type *diameter* for TCP transport, or diameters for both TCP and TLS transport. The Diameter implementation application may automatically upgrade Diameter connections to use TLS as described in the Diameter specification (RFC 3558).

See "Steps for Configuring Diameter Client Nodes and Relay Agents" for more information about configuring network channels for Diameter protocol support.

Configuring Engine Servers to Listen on Any IP Interface

To configure Converged Application Server to listen for UDP traffic on any available IP interface, create a new SIP channel and specify `0.0.0.0` (or `::` for IPv6 networks) as the listen address. Note that you must still configure at least one additional channel with an explicit IP address to use for outgoing SIP messages. (For multihomed machines, each interface used for outgoing messages must have a configured channel.)

Note: You must configure the 0.0.0.0 address directly on the server's network channel. If you configure a SIP channel without specifying the channel listen address, but you do configure a listen address for the server itself, then the SIP channel inherits the server listen address. In this case the SIP channel *does not* listen on IP_ANY.

Note: Using the `0.0.0.0` configuration affects only UDP traffic on Linux platforms. Converged Application Server only creates TCP and HTTP listen threads corresponding to the configured hostname of the server, and localhost. If multiple addresses are mapped to the hostname, Converged Application Server displays warning messages upon startup. To avoid this problem and listen on all addresses, specify the `::` address, which encompasses all available addresses for both IPv6 and IPv4 for HTTP and TCP traffic as well.

Configuring Unique Listen Address Attributes for SIP Data Tier Replicas

Each replica in the SIP data tier must bind to a unique Listen Address attribute (a unique DNS name or IP address) in order to contact one another as peers. Follow these instructions for each replica to assign a unique Listen Address:

1. Access the Administration Console for the Converged Application Server domain.
2. Select **Environment**, then select **Servers** from the left pane.
3. In the right pane, select the name of the server to configure.
4. Select **Configuration**, then select the **General** tab.
5. Enter a unique DNS name or IP address in the Listen Address field.
6. Click Save.

Configuring Static Source Port for Outbound UDP Packets

You can optionally use a static port rather than a dynamically assigned ephemeral port as the source port for outgoing UDP datagrams. Converged Application Server network channels provide a `SourcePorts` attribute that you can use to configure one or more static ports that a server uses for originating UDP packets.

You can identify the ephemeral port currently used by the Converged Application Server by examining the server log file. A log entry appears as follows:

```
<Nov 30, 2005 12:00:00 AM PDT> <Notice> <WebLogicServer> <BEA-000202> <Thread "SIP
Message Processor (Transport UDP)" listening on port 35993.>
```

Caution: Oracle does not recommend using the SourcePorts custom property in most configurations because it degrades performance. Configure the property only in cases where you must specify the exact ports that Converged Application Server uses to originate UDP packets.

To use a static port for outgoing UDP datagrams, first disable use of the ephemeral port by specifying the following server start-up option:

```
-Dwlss.udp.listen.on.ephemeral=false
```

To configure the `SourcePorts` property, use a JMX client such as WLST or directly modify a network channel configuration in **config.xml** to include the custom property. `SourcePorts` defines an array of port numbers or port number ranges. Do not include spaces in the `SourcePorts` definition; use only port numbers, hyphens ("-") to designate ranges of ports, and commas (",") to separate ranges or individual ports. See Example 5–2 for an example configuration.

Example 5–2   Static Port Configuration for Outgoing UDP Packets

```
<network-access-point>
  <name>sip</name>
  <protocol>sip</protocol>
  <listen-port>5060</listen-port>
  <public-port>5060</public-port>
  <http-enabled-for-this-protocol>false</http-enabled-for-this-protocol>
  <tunneling-enabled>false</tunneling-enabled>
  <outbound-enabled>true</outbound-enabled>
  <enabled>true</enabled>
  <two-way-ssl-enabled>false</two-way-ssl-enabled>
  <client-certificate-enforced>false</client-certificate-enforced>
  <custom-properties>SourcePorts=5060</custom-properties>
</network-access-point>
```

6

Configuring Server Failure Detection

This chapter describes how to use the Oracle Communications Converged Application Server "echo server" process to improve SIP data tier failover performance when a server becomes physically disconnected from the network:

Overview of Failover Detection

In a production system, engine tier servers continually access SIP data tier replicas in order to retrieve and write call state data. The Converged Application Server architecture depends on engine tier nodes to detect when a SIP data tier server has failed or become disconnected. When an engine cannot access or write call state data because a replica is unavailable, the engine connects to another replica in the same partition and reports the offline server. The replica updates the current view of the SIP data tier to account for the offline server, and other engines are then notified of the updated view as they access and retrieve call state data.

By default, an engine tier server uses its RMI connection to the replica to determine if the replica has failed or become disconnected. The algorithms used to determine a failure of an RMI connection are reliable, but ultimately they depend on the TCP protocol's retransmission timers to diagnose a disconnection (for example, if the network cable to the replica is removed). Because the TCP retransmission timer generally lasts a full minute or longer, Converged Application Server provides an alternate method of detecting failures that can diagnose a disconnected replica in a matter of a few seconds.

WlssEchoServer Failure Detection

`WlssEchoServer` is a separate process that you can run on the same server hardware as a SIP data tier replica. The purpose of `WlssEchoServer` is to provide a simple UDP echo service to engine tier nodes to be used for determining when a SIP data tier server goes offline, for example in the event that the network cable is disconnected. The algorithm for detecting failures with `WlssEchoServer` is as follows:

1. For all normal traffic, engine tier servers communicate with SIP data tier replicas using TCP. TCP is used as the basic transport between the engine tier and SIP data tier regardless of whether or not `WlssEchoServer` is used.

2. Engine tier servers send a periodic heartbeat message to each configured `WlssEchoServer` over UDP. During normal operation, `WlssEchoServer` responds to the heartbeats so that the connection between the engine node and replica is verified.
3. Should there be a complete failure of the SIP data tier stack, or the network cable is disconnected, the heartbeat messages are not returned to the engine node. In this case, the engine node can mark the replica as being offline *without* having to wait for the normal TCP connection timeout.
4. After identifying the offline server, the engine node reports the failure to an available SIP data tier replica, and the SIP data tier view is updated as described in the previous section.

Also, should a SIP data tier server notice that its local `WlssEchoServer` process has died, it automatically shuts down. This behavior ensures even quicker failover because avoids the time it takes engine nodes to notice and report the failure as described in "Overview of Failover Detection".

You can configure the heartbeat mechanism on engine tier servers to increase the performance of failover detection as necessary. You can also configure the listen port and log file that `WlssEchoServer` uses on SIP data tier servers.

Forced Shutdown for Failed Replicas

If any engine tier server cannot communicate with a particular replica, the engine access another, available replica in the SIP data tier to report the offline server. The replica updates its view of the affected partition to remove the offline server. The updated view is then distributed to all engine tier servers that later access the partition. Propagating the view in this manner helps to ensure that engine servers do not attempt to access the offline replica.

The replica that updates the view also issues a one-time request to the offline replica to ask it to shut down. This is done to try to shut-down running replica servers that cannot be accessed by one or more engine servers due to a network outage. If an active replica can reach the replica marked as "offline," the offline replica shuts down.

WlssEchoServer Requirements and Restrictions

Note: Using `WlssEchoServer` is not required in all Converged Application Server installations. Enable the echo server only when your system requires detection of a network or replica failure faster than the configured TCP timeout interval.

Observe the following requirements and restrictions when using `WlssEchoServer` to detect replica failures:

- If you use the heartbeat mechanism to detect failures, you must ensure that the `WlssEchoServer` process is always running on each replica server machine. If the `WlssEchoServer` process fails or is stopped, the replica will be treated as being "offline" even if the server process is unaffected.
- Note that `WlssEchoServer` listens on all IP addresses available on the server machine.
- `WlssEchoServer` requires a dedicated port number to listen for heartbeat messages.

Starting WlssEchoServer on SIP Data Tier Server Machines

`WlssEchoServer` is a Java program that you can start directly from a shell or command prompt. The basic syntax for starting `WlssEchoServer` is:

```
java -classpath WLSS_HOME/server/lib/wlssechosvr.jar options
com.bea.wcp.util.WlssEchoServer
```

Where `WLSS_HOME` is the path to the WebLogic Server SIP directory and `options` may include one of the options described in Table 6–1.

Table 6–1 WlssEchoServer Options

Option	Description
`-Dwlss.ha.echoserver.ipaddress`	Specifies the IP address on which the `WlssEchoServer` instance listens for heartbeat messages. If you do not specify an IP address, the instance listens on any available IP address (0.0.0.0).
`-Dwlss.ha.echoserver.port`	Specifies the port number used to listen for heartbeat messages. Ensure that the port number you specify is not used by any other process on the server machine. By default `WlssEchoServer` uses port 6734.
`-Dwlss.ha.echoserver.logfile`	Specifies the log file location and name. By default, log messages are written to `./echo_servertime.log` where `time` is the time expressed in milliseconds.

Oracle recommends that you include the command to start `WlssEchoServer` in the same script you use to start each Converged Application Server SIP data tier instance. If you use the `startManagedWebLogic.sh` script to start an engine or SIP data tier server instance, add a command to start `WlssEchoServer` before the final command used to start the server. For example, change the lines:

```
"$JAVA_HOME/bin/java" ${JAVA_VM} ${MEM_ARGS} ${JAVA_OPTIONS}     \
  -Dweblogic.Name=${SERVER_NAME}                                 \
  -Dweblogic.management.username=${WLS_USER}                     \
  -Dweblogic.management.password=${WLS_PW}                       \
  -Dweblogic.management.server=${ADMIN_URL}                      \
  -Djava.security.policy="${WL_HOME}/server/lib/weblogic.policy" \
   weblogic.Server
```

to read:

```
"$JAVA_HOME/bin/java" -classpath WLSS_HOME/server/lib/wlssechosvr.jar    \
  -Dwlss.ha.echoserver.ipaddress=192.168.1.4                   \
  -Dwlss.ha.echoserver.port=6734 com.bea.wcp.util.WlssEchoServer &
"$JAVA_HOME/bin/java" ${JAVA_VM} ${MEM_ARGS} ${JAVA_OPTIONS}     \
  -Dweblogic.Name=${SERVER_NAME}                                 \
  -Dweblogic.management.username=${WLS_USER}                     \
  -Dweblogic.management.password=${WLS_PW}                       \
  -Dweblogic.management.server=${ADMIN_URL}                      \
  -Djava.security.policy="${WL_HOME}/server/lib/weblogic.policy" \
   weblogic.Server
```

Enabling and Configuring the Heartbeat Mechanism on Servers

To enable the `WlssEchoServer` heartbeat mechanism, you must include the `-Dreplica.host.monitor.enabled` JVM argument in the command you use to start all engine and SIP data tier servers. Oracle recommends adding this option directly to the script used to start Managed Servers in your system. For example, in the `startManagedWebLogic.sh` script, change the line:

```
# JAVA_OPTIONS="-Dweblogic.attribute=value -Djava.attribute=value"
```

to read:

```
JAVA_OPTIONS="-Dreplica.host.monitor.enabled=true"
```

Several additional JVM options configure the functioning of the heartbeat mechanism. Table 6–2 describes the options used to configure failure detection.

Table 6–2 WlssEchoServer Options

Option	Description
`-Dreplica.host.monitor.enabled`	This system property is required on both engine and SIP data tier servers to enable the heartbeat mechanism.
`-Dwlss.ha.heartbeat.interval`	Specifies the number of milliseconds between heartbeat messages. By default heartbeats are sent every 1,000 milliseconds.
`-Dwlss.ha.heartbeat.count`	Specifies the number of consecutive, missed heartbeats that are permitted before a replica is determined to be offline. By default, a replica is marked offline if the `WlssEchoServer` process on the server fails to respond to 3 heartbeat messages.
`-Dwlss.ha.heartbeat.SoTimeout`	Specifies the UDP socket timeout value.

7 Using the Engine Tier Cache

This chapter describes how to enable the engine tier cache for improved performance with SIP-aware load balancers:

Overview of Engine Tier Caching

The default Oracle Communications Converged Application Server configuration the engine tier cluster is stateless. A separate SIP data tier cluster manages call state data in one or more partitions, and engine tier servers fetch and write data in the SIP data tier as necessary. Engines can write call state data to multiple replicas in each partition to provide automatic failover should a SIP data tier replica going offline.

Converged Application Server also provides the option for engine tier servers to cache a portion of the call state data locally, as well as in the SIP data tier. When a local cache is used, an engine tier server first checks its local cache for existing call state data. If the cache contains the required data, and the local copy of the data is up-to-date (compared to the SIP data tier copy), the engine locks the call state in the SIP data tier but reads directly from its cache. This improves response time performance for the request, because the engine does not have to retrieve the call state data from a SIP data tier server.

The engine tier cache stores only the call state data that has been most recently used by engine tier servers. Call state data is moved into an engine's local cache as necessary in order to respond to client requests or to refresh out-of-date data. If the cache is full when a new call state must be written to the cache, the least-recently accessed call state entry is first removed from the cache. The size of the engine tier cache is not configurable.

Using a local cache is most beneficial when a SIP-aware load balancer manages requests to the engine tier cluster. With a SIP-aware load balancer, all of the requests for an established call are directed to the same engine tier server, which improves the effectiveness of the cache. If you do not use a SIP-aware load balancer, the effectiveness of the cache is limited, because subsequent requests for the same call may be distributed to different engine tier severs (having different cache contents).

Configuring Engine Tier Caching

Engine tier caching is enabled by default. To disable partial caching of call state data in the engine tier, specify the `engine-call-state-cache-enabled` element in **sipserver.xml**:

```
<engine-call-state-cache-enabled>false</engine-call-state-cache-enabled>
```

When enabled, the cache size is fixed at a maximum of 250 call states. The size of the engine tier cache is not configurable.

Monitoring and Tuning Cache Performance

`SipPerformanceRuntime` monitors the behavior of the engine tier cache. Table 7–1 describes the MBean attributes.

Table 7–1 SipPerformanceRuntime Attribute Summary

Attribute	Description
`cacheRequests`	Tracks the total number of requests for session data items.
`cacheHits`	The server increments this attribute each time a request for session data results in a version of that data being found in the engine tier server's local cache. Note that this counter is incremented even if the cached data is out-of-date and needs to be updated with data from the SIP data tier.
`cacheValidHits`	This attribute is incremented each time a request for session data is fully satisfied by a cached version of the data.

When enabled, the size of the cache is fixed at 250 call states. Because the cache consumes memory, you may need to modify the JVM settings used to run engine tier servers to meet your performance goals. Cached call states are maintained in the tenured store of the garbage collector. Try reducing the fixed "NewSize" value when the cache is enabled (for example, `-XX:MaxNewSize=32m -XX:NewSize=32m`). Note that the actual value depends on the call state size used by applications, as well as the size of the applications themselves.

8 Storing Long-Lived Call State Data in an RDBMS

This chapter describes how to configure a Oracle Communications Converged Application Server domain to use an Oracle or MySQL RDBMS with the SIP data tier cluster, in order to conserve RAM:

Overview of Long-Lived Call State Storage

Converged Application Server enables you to store long-lived call state data in an Oracle or MySQL RDBMS in order to conserve RAM. When you enable RDBMS persistence, by default the SIP data tier persists a call state's data to the RDBMS after the call dialog has been established, and at subsequent dialog boundaries, retrieving or deleting the persisted call state data as necessary to modify or remove the call state.

Oracle also provides an API for application designers to provide "hints" as to when the SIP data tier should persist call state data. These hints can be used to persist call state data to the RDBMS more frequently, or to disable persistence for certain calls.

Note that Converged Application Server only uses the RDBMS to supplement the SIP data tier's in-memory replication functionality. To improve latency performance when using an RDBMS, the SIP data tier maintains SIP timers in memory, along with call states being actively modified (for example, in response to a new call being set up). Call states are automatically persisted only after a dialog has been established and a call is in progress, at subsequent dialog boundaries, or in response to persistence hints added by the application developer.

When used in conjunction with an RDBMS, the SIP data tier selects one replica server instance to process all call state writes (or deletes) to the database. Any available replica can be used to retrieve call states from the persistent store as necessary for subsequent reads.

RDBMS call state storage can be used in combination with an engine tier cache, if your domain uses a SIP-aware load balancer to manage connections to the engine tier. See "Using the Engine Tier Cache".

Requirements and Restrictions

Enable RDBMS call state storage only when all of the following criteria are met:

- The call states managed by your system are typically long-lived.
- The size of the call state to be stored is large. Very large call states may require a significant amount of RAM in order to store the call state.
- Latency performance is not critical to your deployed applications.

The latency requirement, in particular, must be well understood before choosing to store call state data in an RDBMS. The RDBMS call state storage option measurably increases latency for SIP message processing, as compared to using a SIP data tier cluster. If your system must handle a large number of short-lived SIP transactions with brief response times, Oracle recommends storing all call state data in the SIP data tier.

Note: RDBMS persistence is designed only to reduce the RAM requirements in the SIP data tier for large, long-lived call states. The persisted data cannot be used to restore a failed SIP data tier partition or replica.

Steps for Enabling RDBMS Call State Storage

In order to use the RDBMS call state storage feature, your Converged Application Server domain must include the necessary JDBC configuration, SIP Servlet container configuration, and a database having the schema required to store the call state. You can automate much of the required configuration by using the Configuration Wizard to set up a new domain with the RDBMS call state template. See "Using the Configuration Wizard RDBMS Store Template".

If you have an existing Converged Application Server domain, or you want to configure the RDBMS store on your own, see "Configuring RDBMS Call State Storage" for instructions to configure JDBC and Converged Application Server to use an RDBMS store.

Using the Configuration Wizard RDBMS Store Template

The Configuration Wizard provides a simple template that helps you easily begin using and testing the RDBMS call state store. Follow these steps to create a new domain from the template:

1. Start the Configuration Wizard application (config.sh):

   ```
   cd ~/WL_HOME/common/bin
   ./config.sh
   ```
   where WL_HOME is the path to the directory where the WebLogic Server component of Converged Application Server is installed.
2. Accept the default selection, *Create a new WebLogic domain*, and click **Next**.
3. Select *Base this domain on an existing template*, and click **Browse** to display the Select a Template dialog.
4. Select the template named `replicateddomain.jar`, and click **OK**.
5. Click **Next**.
6. Enter the username and password for the Administrator of the new domain, and click Next.

7. Select a JDK to use, and click Next.
8. Select No to keep the settings defined in the source template file, and click **Next**.
9. Click **Create** to create the domain.

   The template creates a new domain with two engine tier servers in a cluster, two SIP data tier servers in a cluster, and an Administration Server (AdminServer). The engine tier cluster includes the following resources and configuration:

   - A JDBC datasource, `wlss.callstate.datasource`, required for storing long-lived call state data. Note that you must modify this configuration to configure the datasource for your own RDBMS server. See "Modify the JDBC Datasource Connection Information".
   - A persistence configuration (shown in the SipServer node, Configuration > Persistence tab of the Administration Console) that defines default handling of persistence hints for both RDBMS and geographical redundancy.
10. Click **Done** to exit the configuration wizard.
11. Follow the steps under "Modify the JDBC Datasource Connection Information" to create the necessary tables in your RDBMS.
12. Follow the steps under "Create the Database Schema" to create the necessary tables in your RDBMS.

Modify the JDBC Datasource Connection Information

After installing the new domain, modify the template JDBC datasource to include connection information for your RDBMS server:

1. Use your browser to access the URL http://*address*:*port*/console where *address* is the Administration Server's listen address and *port* is the listen port.
2. Select **Services**, then select **JDBC**, and then select the **Data Sources** tab in the left pane.
3. Select the data source named **wlss.callstate.datasource** in the right pane.
4. Select **Configuration**, and then select the **Connection Pool** tab in the right pane.
5. Modify the following connection pool properties:
   - **URL**: Modify the URL to specify the host name and port number of your RDBMS server.
   - **Properties**: Modify the value of the user, portNumber, SID, and serverName properties to match the connection information for your RDBMS.
   - **Password** and **Confirm Password**: Enter the password of the RDBMS user you specified.
6. Click **Save** to save your changes.
7. Select the **Targets** tab in the right pane.
8. On the Select Targets page, select the name of your SIP data tier cluster (for example, CAS_DATA_TIER_CLUST), then click **Save**.
9. Click **Save**.
10. Follow the steps under "Create the Database Schema" to create the necessary tables in your RDBMS.

Configuring RDBMS Call State Storage

To change an existing Converged Application Server domain to store call state data in an Oracle or MySQL RDBMS, you must configure the required JDBC datasource, edit the Converged Application Server configuration, and add the required schema to your database. Follow the instructions in the sections below to configure an Oracle Database.

Configure JDBC Resources

Follow these steps to create the required JDBC resources in your domain:

1. Boot the Administration Server for the domain if it is not already running.
2. Access the Administration Console for the domain.
3. Select **Services**, then select **JDBC**, and then select the **Data Sources** tab in the left pane.
4. Click **New** to create a new data source.
5. Fill in the fields of the Create a New JDBC Data Source page as follows:
   - **Name:** Enter wlss.callstate.datasource
   - **JNDI Name**: Enter wlss.callstate.datasource.
   - **Database Type**: Select **Oracle**.
   - **Database Driver**: Select an appropriate JDBC driver from the **Database Driver** list. Note that some of the drivers listed in this field may not be installed by default on your system. Install third-party drivers as necessary using the instructions from your RDBMS vendor.
6. Click **Next**:
7. Fill in the fields of the **Connection Properties** tab using connection information for the database you wan to use. Click **Next** to continue.
8. Click **Test Configuration** to test your connection to the RDBMS, or click **Next** to continue.
9. On the Select Targets page, select the name of your SIP data tier cluster (for example, CAS_DATA_TIER_CLUST).
10. Click **Finish** to save your changes.

Configure Converged Application Server Persistence Options

Follow these steps to configure the Converged Application Server persistence options to use an RDBMS call state store:

1. Boot the Administration Server for the domain if it is not already running.
2. Access the Administration Console for the domain.
3. Select the **SipServer** node in the left pane.
4. Select **Configuration**, then select the **Persistence** tab in the right pane.
5. In the **Default Handling** drop-down menu, select either **db** or **all**. It is acceptable to select **all** because geographically-redundant replication is only performed if the **Geo Site ID** and **Geo Remote T3 URL** fields have been configured.
6. Click **Save** to save your changes.

Create the Database Schema

Converged Application Server includes a SQL script, `callstate.sql`, that you can use to create the tables necessary for storing call state information. The script is installed to the `user_staged_config` subdirectory of the domain directory when you configure a replicated domain using the Configuration Wizard. The script is also available in the *WL_HOME*`/common/templates/scripts/db/oracle` directory.

The contents of the `callstate.sql` SQL script are shown in Example 8–1.

Example 8–1 callstate.sql Script for Call State Storage Schema

```
drop table callstate;

create table callstate (
  key1 int,
  key2 int,
  bytes blob default empty_blob(),
  constraint pk_callstate primary key (key1, key2)
);
```

Follow these steps to execute the script commands using SQL*Plus:

1. Move to the Converged Application Server `utils` directory, in which the SQL Script is stored:

   ```
   cd ~/WL_HOME/common/templates/scripts/db/oracle
   ```

   where WL_HOME is the path to the directory where the WebLogic Server component of Converged Application Server is installed.

2. Start the SQL*Plus application, connecting to the Oracle database in which you will create the required tables. Use the same username, password, and connect to the same database that you specified when configuring the JDBC driver in "Configure JDBC Resources". For example:

   ```
   sqlplus username/password@connect_identifier
   ```

   where `connect_identifier` connects to the database identified in the JDBC connection pool.

3. Execute the Converged Application Server SQL script, `callstate.sql`:

   ```
   START callstate.sql
   ```

4. Exit SQL*Plus:

   ```
   EXIT
   ```

Using Persistence Hints in SIP Applications

Converged Application Server provides a simple API to provide "hints" as to when the SIP data tier should persist call state data. You can use the API to disable persistence for specific calls or SIP requests, or to persist data more frequently than the default setting (at SIP dialog boundaries).

To use the API, simply obtain a `WlssSipApplicationSession` instance and use the `setPersist` method to enable or disable persistence. Note that you can enable or disable persistence either to an RDBMS store, or to as geographically-redundant Converged Application Server installation (see "Configuring Geographically-Redundant Installations").

For example, some SIP-aware load balancing products use the SIP OPTIONS message to determine if a SIP Server is active. To avoid persisting these messages to an RDBMS and to a geographically-redundant site, a Servlet might implement a `doOptions` method to echo the request and turn off persistence for the message, as shown in Example 8–2.

Example 8–2 Disabling RDBMS Persistence for Option Methods

```
protected void doOptions(SipServletRequest req) throws IOException {
    WlssSipApplicationSession session =
      (WlssSipApplicationSession) req.getApplicationSession();
    session.setPersist(WlssSipApplicationSession.PersistenceType.DATABASE,
      false);
    session.setPersist(WlssSipApplicationSession.PersistenceType.GEO_REDUNDANCY,
false);
    req.createResponse(200).send();
}
```

9 Configuring Geographically-Redundant Installations

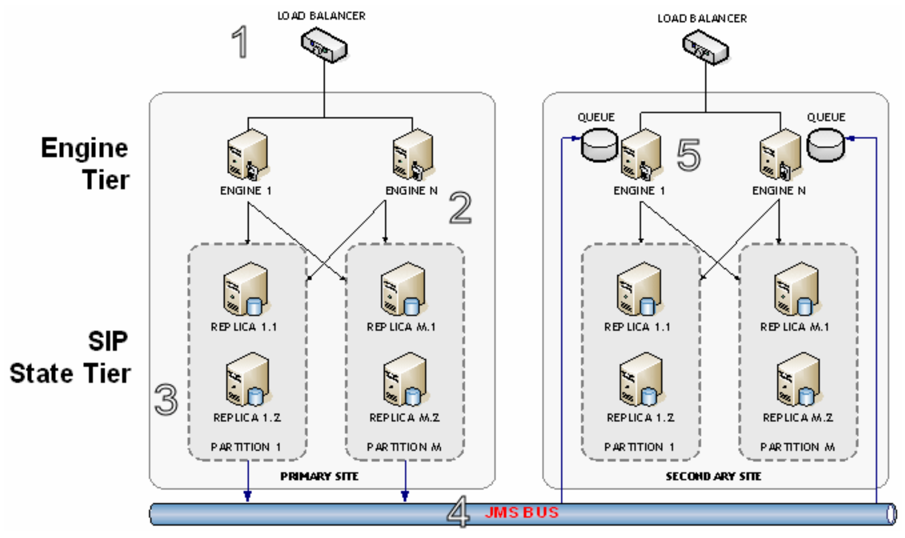
This chapter describes how to replicate call state transactions across multiple, regional Oracle Communications Converged Application Server installations ("sites"):

Introducing Geographic Redundancy

Geo-Redundancy ensures uninterrupted transactions and communications for providers, using geographically-separated SIP server deployments.

A primary site can process various SIP transactions and communications and upon determining a transaction boundary, replicate the state data associated with the transaction being processed, to a secondary site. Upon failure of the primary site, calls are routed from the failed primary site to a secondary site for processing. Similarly, upon recovery, the calls are re-routed back to the primary site.

Figure 9–1 Geo-Redundancy



In the preceding figure, Geo-Redundancy is portrayed. The process proceeds as follows:

1. Call is initiated on a primary Converged Application Server Cluster site, call setup and processing occurs normally.
2. Call is replicated as usual to the site's SIP State Tier, and becomes eligible for replication to a secondary site.
3. A single replica in the SIP State Tier then places the call state data to be replicated on a JMS queue configured.
4. Call is transmitted to one of the available engines using JMS over WAN.
5. Engines at the secondary site monitor their local queue for new messages. Upon receiving a message, an Engine in the secondary site Converged Application Server Cluster persists the call state data and assigns it the site ID value of the primary site.

Table 9–1 Geographic Redundancy flow

Normal Operation	Failover
When a session is initiated on a primary Converged Application Server site, call setup and processing occurs normally.	Global LB policy updated to begin routing calls - primary site to secondary site.
When a SIP transaction boundary is reached, the call is replicated (in-memory) to the site's data tier, and becomes eligible for replication to a secondary site.	Once complete, the secondary site begins processing requests for the backed-up call state data.
A single replica in the data tier then places the call state data to be replicated on a JMS queue configured on the replica site.	When a requests hit secondary site engine retrieves the data and activates the call state, taking ownership for the call.
Data is transmitted to one of the available engines round-robin fashion.	Sets the site ID associated with the call to zero (making it appear local).
Engines at the secondary site monitor their local queue for new messages.	Activates all dormant timers present in the call state.

Table 9–1 (Cont.) Geographic Redundancy flow

Normal Operation	Failover
Upon receiving a message, an engine on the secondary site persists the call state data and assigns it the site ID value of the primary site.	By default, call states are activated only for individual calls, and only after those calls are requested on the backup site.
The site ID distinguishes replicated call state data on the secondary site from any other call state data actively managed by the secondary site.	Servlets can use the WlssSipApplicationSession.getGeoSiteId() method to examine the site ID associated with a call.
Timers in replicated call state data remain dormant on the secondary site, so that timer processing does not become a bottleneck to performance.	Any non-zero value for the site ID indicates that the Servlet is working with call state data that was replicated from another site.

Situations Best Suited to Use Geo-Redundancy

The following situations are best suited to take advantage of Geo-Redundancy:

- Your application uses SIP dialog states that are long-lived (dialog states that typically last 30 seconds or longer, such as SUBSCRIBE dialogs or conferences)
- Your application would reasonably be able to reconstruct the session (re-INVITE, expire SUBSCRIBE dialogs to trigger re-subscriptions, and so on) from the state that has been replicated
- The link between two Converged Application Server clusters or sites is low-bandwidth (<1Gb/s each direction) or high (or variable) latency (>5ms 95%)

Situations Not Suited to Use Geo-Redundancy

Geo-Redundancy should not be used in these situations:

- A high-capacity link between sites is available
- Your application does not reach SIP dialog steady-states that are likely to last longer than the time it would take to re-route all traffic to the secondary site in the event of catastrophic failure (15-30 seconds)
- If the application session is likely to be terminated by the user before the application could re-construct the session (most users will disconnect their calls before the session can be re-established from the secondary site)
- The volume of session state objects created by the application is greater than the site interconnect can support

Geo-Redundancy Considerations

Consider the following issues when planning for Geo-Redundancy:

- Dimension the system for the site link.
- Each dialog state is ~25KB on the wire (25600 bits).
- A typical B2BUA is two (2) dialogs.
- Aim for 25% utilization (or less, depending on the specific equipment and topology of the site) to accommodate “jitter” and sustained latency on the link.

  For example, a 100 Mb/s link can handle approximately1000 call states per second, and a typical B2BUA (in the default configuration) generates 4 states during the call (two for each dialog). So, a 100 Mb/s link will support a single Converged

Application Server cluster dimensioned for a peak arrival rate (call rate) of 250 CPS.

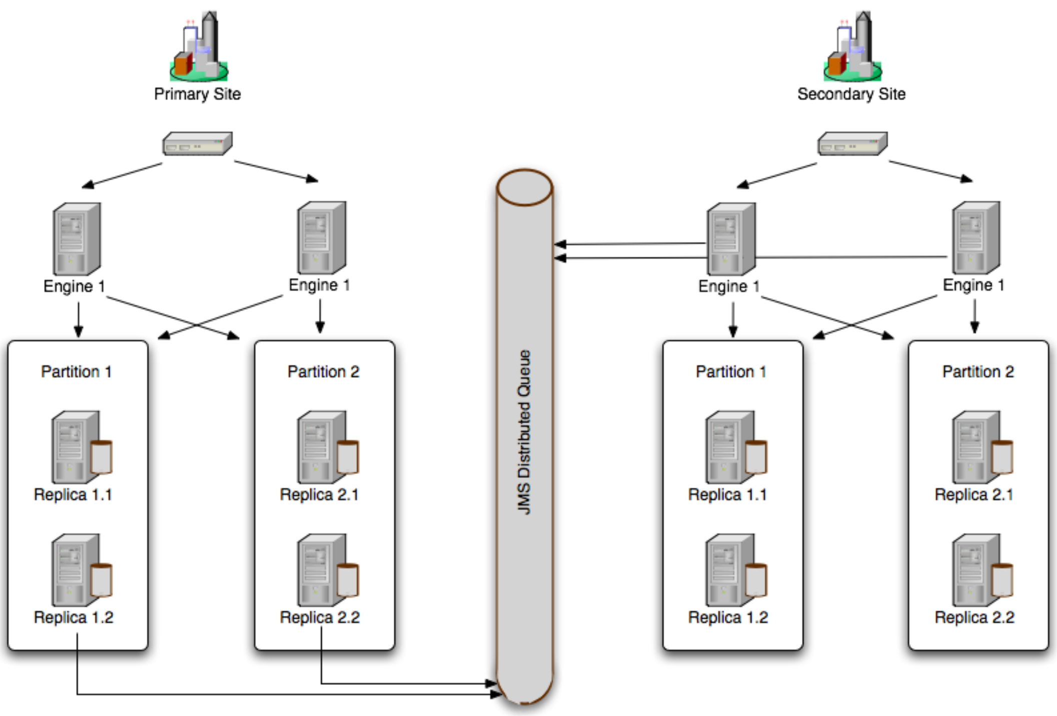
- Geo-Redundancy is not *transparent* to the application; in most cases the application must be designed to use `SetPersist()` appropriately, and the developer must consider the volume of state that the application will queue for replication between sites.
- `SetPersist()` should be used within the application code to selectively identify dialog states that will be long-lived.
- Given the time it generally takes to route traffic to a secondary site, any application that replicates state more frequently will unnecessarily saturate the JMS queue and site interconnect.
- Tuning of JMS to the specific application environment is required: Serialization options, message batching, reliable delivery options and queue size are all variable, depending on the specific application and site characteristics
- Geo-Redundancy default behavior is to replicate all dialog state changes when Geo-Redundancy is enabled for the container (this is *not* recommended for production deployments).
- Given the time it generally takes to route traffic to a secondary site, any application that replicates state more frequently will unnecessarily saturate the site interconnect.
- `SetPersist()` should be used within the application code to selectively identify dialog states that will be long-lived (longer than ~20-30 seconds would be a reasonable threshold).

Using Geographically-Redundant SIP Data Tiers

The basic call state replication functionality available in the Converged Application Server SIP data tier provides excellent failover capabilities for a single site installation. However, the active replication performed within the SIP data tier requires high network bandwidth in order to meet the latency performance needs of most production networks. This bandwidth requirement makes a single SIP data tier cluster unsuitable for replicating data over large distances, such as from one regional data center to another.

The Converged Application Server geographic persistence feature enables you to replica call state transactions across multiple Converged Application Server installations (multiple Administrative domains or "sites"). A geographically-redundant configuration minimizes dropped calls in the event of a catastrophic failure of an entire site, for example due to an extended, regional power outage.

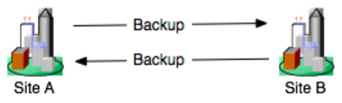
Figure 9–2 Oracle Communications Converged Application Server Geographic Persistence



Example Domain Configurations

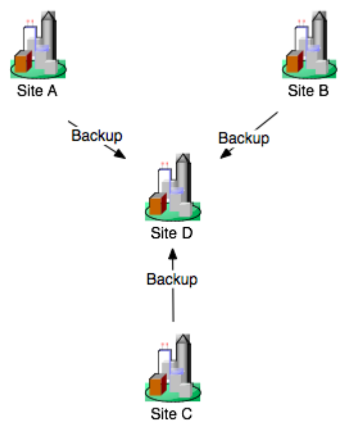
A secondary Converged Application Server domain that persists data from another domain may itself process SIP traffic, or it may exist solely as an active standby domain. In the most common configuration, two sites are configured to replicate each other's call state data, with each site processing its own local SIP traffic. The administrator can then use either domain as the "secondary" site should one of domains fail.

Figure 9–3 Common Geographically-Redundant Configuration



An alternate configuration utilizes a single domain that persists data from multiple, other sites, acting as the secondary for those sites. Although the secondary site in this configuration can also process its own, local SIP traffic, be aware that the resource requirements of the site may be considerable because of the need to persist active traffic from several other installations.

Figure 9–4 Alternate Geographically-Redundant Configuration



When using geographic persistence, a single replica in the primary site places modified call state data on a distributed JMS queue. By default, data is placed on the queue only at SIP dialog boundaries. (A custom API is provided for application developers who want to replicate data using a finer granularity, as described in "Using Persistence Hints in SIP Applications".) In a secondary site, engine tier servers use a message listener to monitor the distributed queue to receive messages and write the data to its own SIP data tier cluster. If the secondary site uses an RDBMS to store long-lived call states (recommended), then all data writes from the distribute queue go directly to the RDBMS, rather than to the in-memory storage of the SIP data tier.

Requirements and Limitations

The Converged Application Server geographically-redundant persistence feature is most useful for sites that manage long-lived call state data in an RDBMS. Short-lived calls may be lost in the transition to a secondary site, because Converged Application Server may choose to collect data for multiple call states before replicating between sites.

You must have a reliable, site-aware load balancing solution that can partition calls between geographic locations, as well as monitor the health of a given regional site. Converged Application Server provides no automated functionality for detecting the failure of an entire domain, or for failing over to a secondary site. It is the responsibility of the Administrator to determine when a given site has "failed," and to redirect that site's calls to the correct secondary site. Furthermore, the site-aware load balancer must direct all messages for a given callId to a single home site (the "active" site). If, after a failover, the failed site is restored, the load balancer must continue directing calls to the active site and not partition calls between the two sites.

During a failover to a secondary site, some calls may be dropped. This can occur because Converged Application Server generally queues call state data for site replication only at SIP dialog boundaries. Failures that occur before the data is written to the queue result in the loss of the queued data.

Also, Converged Application Server replicates call state data across sites only when a SIP dialog boundary changes the call state. If a long-running call exists on the primary site before the secondary site is started, and the call state remains unmodified, that call's data is not replicated to the secondary site. Should a failure occur before a long-running call state has been replicated, the call is lost during failover.

When planning for the capacity of a Converged Application Server installation, be aware that, after a failover, a given site must be able to support all of the calls from the failed site as well as from its own geographic location. This means that all sites that are involved in a geographically-redundant configuration will operate at less than maximum capacity until a failover occurs.

Steps for Configuring Geographic Persistence

In order to use the Converged Application Server geographic persistence features, you must perform certain configuration tasks on both the primary "home" site and on the secondary replication site.

Table 9–2 Steps for Configuring Geographic Persistence

Steps for Primary "Home" Site	Steps for Secondary "Replication" Site:
1. Install Converged Application Server software and create replicated domain. **2.** Enable RDBMS storage for long-lived call states (recommended). **3.** Configure persistence options to: define the unique regional site ID; identify the secondary site's URL; and enable replication hints.	**1.** Install Converged Application Server software and create replicated domain. **2.** Enable RDBMS storage for long-lived call states (recommended). **3.** Configure JMS Servers and modules required for replicating data. **4.** Configure persistence options to define the unique regional site ID.

Note: In most production deployments, two sites will perform replication services for each other, so you will generally configure each installation as both a primary and secondary site.

Converged Application Server provides domain templates to automate the configuration of most of the resources described in Table 9–2. See "Using the Configuration Wizard Templates for Geographic Persistence" for information about using the templates.

If you have an existing Converged Application Server domain and want to use geographic persistence, follow the instructions in "Manually Configuring Geographic Redundancy" to create the resources.

Using the Configuration Wizard Templates for Geographic Persistence

Converged Application Server provides two Configuration Wizard templates for using geographic persistence features:

- `WL_HOME/common/templates/domains/geo1domain.jar` configures a primary site having a site ID of 1. The domain replicates data to the engine tier servers created in `geo2domain.jar`.
- `WL_HOME/common/templates/domains/geo2domain.jar` configures a secondary site that replicates call state data from the domain created with `geo1domain.jar`. This installation has site ID of 2.

The server port numbers in both domain templates are unique, so you can test geographic persistence features on a single machine if necessary. Follow the instructions in the sections that follow to install and configure each domain.

Installing and Configuring the Primary Site

Follow these steps to create a new primary domain from the template:

1. Start the Configuration Wizard application:

   ```
   cd ~/cas_home/wlserver_10.3/common/bin
   ./config.sh
   ```

   where `cas_home` is the directory where you installed the Converged Application Server software.

2. Accept the default selection, Create a new WebLogic domain, and click **Next**.
3. Select **Base this domain on an existing template**, and click **Browse** to display the **Select a Template** dialog.
4. Select the template named `geo1domain.jar`, and click OK.
5. Click **Next**.
6. Enter the username and password for the Administrator of the new domain, and click **Next**.
7. Select a JDK to use, and click **Next**.
8. Select **No** to keep the settings defined in the source template file, and click **Next**.
9. Click **Create** to create the domain.

   The template creates a new domain with two engine tier servers in a cluster, two SIP data tier servers in a cluster, and an Administration Server (AdminServer). The engine tier cluster includes the following resources and configuration:

   - A JDBC datasource, `wlss.callstate.datasource`, required for storing long-lived call state data. If you want to use this functionality, edit the datasource to include your RDBMS connection information as described in "Modify the JDBC Datasource Connection Information".
   - A persistence configuration (shown in the SipServer node, Configuration > Persistence tab of the Administration Console) that defines:
     - Default handling of persistence hints for both RDBMS and geographic persistence.
     - A Geo Site ID of 1.
     - A Geo Remote T3 URL of `t3://localhost:8011,localhost:8061`, which identifies the engine tier servers in the "geo2" domain as the replication site for geographic redundancy.

10. Click **Done** to exit the configuration wizard.
11. Follow the steps under "Installing the Secondary Site" to create the domain that performs the replication.

Installing the Secondary Site

Follow these steps to use a template to create a secondary site from replicating call state data from the "geo1" domain:

1. Start the Configuration Wizard application:

   ```
   cd ~/cas_home_home/wlserver_10.3/common/bin
   ./config.sh
   ```

where `cas_home` is the directory where you installed the Converged Application Server software.

2. Accept the default selection, **Create a new WebLogic domain**, and click **Next**.
3. Select **Base this domain on an existing template**, and click **Browse** to display the **Select a Template** dialog.
4. Select the template named `geo2domain.jar`, and click **OK**.
5. Click **Next**.
6. Enter the username and password for the Administrator of the new domain, and click **Next**.
7. Select a JDK to use, and click **Next**.
8. Select **No** to keep the settings defined in the source template file, and click **Next**.
9. Click **Create** to create the domain.

   The template creates a new domain with two engine tier servers in a cluster, two SIP data tier servers in a cluster, and an Administration Server (AdminServer). The engine tier cluster includes the following resources and configuration:

   - A JDBC datasource, `wlss.callstate.datasource`, required for storing long-lived call state data. If you want to use this functionality, edit the datasource to include your RDBMS connection information as described in "Modify the JDBC Datasource Connection Information".
   - A persistence configuration (shown in the SipServer node, Configuration > Persistence tab of the Administration Console) that defines:
     - Default handling of persistence hints for both RDBMS and geographic redundancy.
     - A Geo Site ID of 2.
   - A JMS system module, `SystemModule-Callstate`, that includes:
     - `ConnectionFactory-Callstate`, a connection factory required for backing up call state data from a primary site.
     - `DistributedQueue-Callstate`, a uniform distributed queue required for backing up call state data from a primary site.

     The JMS system module is targeted to the site's engine tier cluster
   - Two JMS Servers, `JMSServer-1` and `JMSServer-2`, are deployed to `engine1-site2` and `engine2-site2`, respectively.
10. Click **Done** to exit the configuration wizard.

Manually Configuring Geographic Redundancy

If you have an existing replicated Converged Application Server installation, or pair of installations, you must manually create the JMS and JDBC resources required for enabling geographic redundancy. You must also configure each site to perform replication. The steps to enable geographicic redundancy are:

1. Configure JDBC Resources. Oracle recommends configuring both the primary and secondary sites to store long-lived call state data in an RDBMS.

2. Configure Persistence Options. Persistence options must be configured on both the primary and secondary sites to enable engine tier hints to write to an RDBMS or to replicate data to a geographically-redundant installation.
3. Configure JMS Resources. A secondary site must have available JMS Servers and specific JMS module resources in order to replicate call state data from another site.

The sections that follow describe each step in detail.

Configuring JDBC Resources (Primary and Secondary Sites)

Follow the instructions in "Storing Long-Lived Call State Data in an RDBMS" to configure the JDBC resources required for storing long-lived call states in an RDBMS.

Configuring Persistence Options (Primary and Secondary Sites)

Both the primary and secondary sites must configure the correct persistence settings in order to enable replication for geographic redundancy. Follow these steps to configure persistence:

1. Use your browser to access the URL http://*address*:*port*/console where *address* is the Administration Server's listen address and *port* is the listen port.
2. Select the **SipServer** node in the left pane. The right pane of the console provides two levels of tabbed pages that are used for configuring and monitoring Converged Application Server.
3. Select **Configuration**, then select the **Persistence** tab in the right pane.
4. Configure the Persistence attributes as follows:
   - **Default Handling**: Select "all" to persist long-lived call state data to an RDBMS and to replicate data to an external site for geographic redundancy (recommended). If your installation does not store call state data in an RDBMS, select "geo" instead of "all."
   - **Geo Site ID**: Enter a unique number from 1 to 9 to distinguish this site from all other configured sites. Note that the site ID of 0 is reserved to indicate call states that are local to the site in question (call states not replicated from another site).
   - **Geo Remote T3 URL**: For primary sites (or for secondary sites that replicate their own data to another site), enter the T3 URL or URLs of the engine tier servers that will replicate this site's call state data. If the secondary engine tier cluster uses a cluster address, you can enter a single T3 URL, such as t3://mycluster:7001. If the secondary engine tier cluster does not use a cluster address, enter the URLs for each individual engine tier server separated by a comma, such as t3://engine1-east-coast:7001,t3://engine2-east-coast:7002,t3://engine3-east-coast:7001,t4://engine4-east-coast:7002.
5. Click **Save** to save your configuration changes.
6. Click **Activate Changes** to apply your changes to the engine tier servers.

Configuring JMS Resources (Secondary Site Only)

Any site that replicates call state data from another site must configure certain required JMS resources. The resources are not required for sites that do not replicate data from another site.

Follow these steps to configure JMS resources:

1. Use your browser to access the URL http://*address*:*port*/console where *address* is the Administration Server's listen address and *port* is the listen port.
2. Select **Services**, then select **Messaging**, and then select the **JMS Servers** tab in the left pane.
3. Click **New** in the right pane.
4. Enter a unique name for the JMS Server or accept the default name. Click **Next** to continue.
5. In the **Target** list, select the name of a single engine tier server node in the installation. Click **Finish** to create the new Server.
6. Repeat Steps 3 through 6 to create a dedicated JMS Server for each engine tier server node in your installation.
7. Select **Services**, then select **Messaging**, and then select the **JMS Modules** node in the left pane.
8. Click **New** in the right pane.
9. Fill in the fields of the Create JMS System Module page as follows:
   - **Name**: Enter a name for the new module, or accept the default name.
   - **Descriptor File Name**: Enter the prefix a configuration file name in which to store the JMS module configuration (for example, `systemmodule-callstate`).
10. Click **Next** to continue.
11. Select the name of the engine tier cluster, and choose the option **All servers in the cluster**.
12. Click **Next** to continue.
13. Select **Would you like to add resources to this JMS system module** and click Finish to create the module.
14. Click **New** to add a new resource to the module.
15. Select the **Connection Factory** option and click Next.
16. Fill in the fields of the Create a new JMS System Module Resource as follows:
    - **Name**: Enter a descriptive name for the resource, such as ConnectionFactory-Callstate.
    - **JNDI Name**: Enter the name `wlss.callstate.backup.site.connection.factory`.
17. Click **Next** to continue.
18. Click **Finish** to save the new resource.
19. Select the name of the connection factory resource you just created.
20. Select **Configuration**, then select the **Load Balance** tab in the right pane.
21. De-select the **Server Affinity Enabled** option, and click **Save**.
22. Re-select **Services**, then select **Messaging**, and then select the **JMS Modules** node in the left pane.
23. Select the name of the JMS module you created in the right pane.
24. Click **New** to create another JMS resource.

25. Select the **Distributed Queue** option and click Next.
26. Fill in the **Name** field of the Create a new JMS System Module Resource by entering a descriptive name for the resource, such as DistributedQueue-Callstate.
27. **JNDI Name**: Enter the name Fill in the fields of the Create a new JMS System Module Resource as follows:
    - **Name**: Enter a descriptive name for the resource, such as ConnectionFactory-Callstate.
    - **JNDI Name**: Enter the name `wlss.callstate.backup.site.queue`.
28. Click **Next** to continue.
29. Click **Finish** to save the new resource.
30. Click **Save** to save your configuration changes.
31. Click **Activate Changes** to apply your changes to the engine tier servers.

Understanding Geo-Redundant Replication Behavior

This section provides more detail into how multiple sites replicate call state data. Administrators can use this information to better understand the mechanics of geo-redundant replication and to better troubleshoot any problems that may occur in such a configuration. Note, however, that the internal workings of replication across Converged Application Server installations is subject to change in future releases of the product.

Call State Replication Process

When a call is initiated on a primary Converged Application Server site, call setup and processing occurs normally. When a SIP dialog boundary is reached, the call is replicated (in-memory) to the site's SIP data tier, and becomes eligible for replication to a secondary site. Converged Application Server may choose to aggregate multiple call states for replication in order to optimize network usage.

A single replica in the SIP data tier then places the call state data to be replicated on a JMS queue configured on the replica site. Data is transmitted to one of the available engines (specified in the `geo-remote-t3-url` element in **sipserver.xml**) in a round-robin fashion. Engines at the secondary site monitor their local queue for new messages.

Upon receiving a message, an engine on the secondary site persists the call state data and assigns it the site ID value of the primary site. The site ID distinguishes replicated call state data on the secondary site from any other call state data actively managed by the secondary site. Timers in replicated call state data remain dormant on the secondary site, so that timer processing does not become a bottleneck to performance.

Call State Processing After Failover

To perform a failover, the Administrator must change a global load balancer policy to begin routing calls from the primary, failed site to the secondary site. After this process is completed, the secondary site begins processing requests for the backed-up call state data. When a request is made for data that has been replicated from the failed site, the engine retrieves the data and activates the call state, taking ownership for the call. The activation process involves:

- Setting the site ID associated with the call to zero (making it appear local).

- Activating all dormant timers present in the call state.

By default, call states are activated only for individual calls, and only after those calls are requested on the backup site. `SipServerRuntimeMBean` includes a method, `activateBackup(byte site)`, that can be used to force a site to take over all call state data that it has replicated from another site. The Administrator can execute this method using a WLST configuration script. Alternatively, an application deployed on the server can detect when a request for replicated site data occurs, and then execute the method. Example 9–1 shows sample code from a JSP that activates a secondary site, changing ownership of all call state data replicated from site 1. Similar code could be used within a deployed Servlet. Note that either a JSP or Servlet must run as a privileged user in order to execute the `activateBackup` method.

In order to detect whether a particular call state request, Servlets can use the `WlssSipApplicationSession.getGeoSiteId()` method to examine the site ID associated with a call. Any non-zero value for the site ID indicates that the Servlet is working with call state data that was replicated from another site.

Example 9–1 Activating a Secondary Site Using JMX

```
<%
    byte site = 1;

    InitialContext ctx = new InitialContext();
    MBeanServer server = (MBeanServer) ctx.lookup("java:comp/env/jmx/runtime");
    Set set = server.queryMBeans(new ObjectName("*:*,Type=SipServerRuntime"),
null);
    if (set.size() == 0) {
      throw new IllegalStateException("No MBeans Found!!!");
    }

    ObjectInstance oi = (ObjectInstance) set.iterator().next();
    SipServerRuntimeMBean bean = (SipServerRuntimeMBean)
      MBeanServerInvocationHandler.newProxyInstance(server,
        oi.getObjectName());

    bean.activateBackup(site);
  %>
```

Note that after a failover, the load balancer must route all calls having the same callId to the newly-activated site. Even if the original, failed site is restored to service, the load balancer must not partition calls between the two geographical sites.

Removing Backup Call States

You may also choose to stop replicating call states to a remote site in order to perform maintenance on the remote site or to change the backup site entirely. Replication can be stopped by setting the **Site Handling** attribute to "none" on the primary site as described in "Configuring Persistence Options (Primary and Secondary Sites)".

After disabling geographic replication on the primary site, you also may want to remove backup call states on the secondary site. `SipServerRuntimeMBean` includes a method, `deleteBackup(byte site)`, that can be used to force a site to remove all call state data that it has replicated from another site. The Administrator can execute this method using a WLST configuration script or via an application deployed on the secondary site. The steps for executing this method are similar to those for using the `activateBackup` method, described in "Call State Processing After Failover".

Monitoring Replication Across Regional Sites

The `ReplicaRuntimeMBean` includes two new methods to retrieve data about geographically-redundant replication:

- `getBackupStoreOutboundStatistics()` provides information about the number of calls queued to a secondary site's JMS queue.
- `getBackupStoreInboundStatistics()` provides information about the call state data that a secondary site replicates from another site.

See the Converged Application Server JavaDoc for more information about `ReplicaRuntimeMBean`.

Troubleshooting Replication

In addition to using the `ReplicaRuntimeMBean` methods described in "Monitoring Replication Across Regional Sites", Administrators should monitor any SNMP traps that indicate failed database writes on a secondary site installation.

Administrators must also ensure that all sites participating in geographically-redundant configurations use unique site IDs.

10 Upgrading Production Converged Application Server Software

This chapter provides general instructions for upgrading Oracle Communications Converged Application Server software to a new Service Pack release. The service pack may contain additional tools or instructions for performing the upgrade which may supersede the information in this chapter. Refer to those instructions if they are available.

This chapter does not cover how to upgrade software for a major release or how to upgrade deployed applications. See "Steps for Upgrading a Deployed SIP Application" for information about updating deployed applications.

This chapter covers the following topics:

Overview of System Upgrades

Because a typical production Converged Application Server installation uses multiple server instances in both the engine and SIP data tiers, upgrading the Converged Application Server software requires that you follow very specific practices. These practices ensure that:

- Existing clients of deployed SIP Servlets are not interrupted or lost during the upgrade procedure.
- The upgrade procedure can be "rolled back" to a previous state if any problems occur.

The sections that follow describe how to use a configured load balancer to perform a "live" upgrade of the Converged Application Server software on a production installation.

When upgrading the Converged Application Server software (for example, in response to a Service Pack), a new engine tier cluster is created to host newly-upgraded engine tier instances. One-by-one, servers in the engine tier are shut-down, upgraded, and then restarted in the new target cluster. While servers are being upgraded, Converged Application Server automatically forwards requests from one engine tier cluster to the other as necessary to ensure that SIP data tier requests are always initiated by a compatible engine tier server. After all servers have been upgraded, the older cluster is removed and no longer used. After upgrading the engine tier cluster, servers in the SIP data tier may also be upgraded, one-by-one.

Requirements for Upgrading a Production System

To upgrade a production Converged Application Server installation you require:

- A compatible load balancer product and administrator privileges for reconfiguring the load balancer virtual IP addresses and pools.
- Adequate disk space on the Administration Server machine and on each Managed Server machine for installing a copy of the new Converged Application Server software.
- Privileges for modifying configuration files on the Converged Application Server Administration Server machine.
- Privileges for shutting down and starting up individual Managed Server instances.
- Three or more replicas in each partition of the SIP data tier, in order to upgrade the Converged Application Server software to a new version. With fewer than three replicas in each partition, it is not possible to safely upgrade a production SIP data tier deployment as no backup replica would be available during the upgrade procedure.

Caution: Before modifying any production installation, thoroughly test your proposed changes in a controlled, "stage" environment to ensure software compatibility and verify expected behavior.

Upgrading to a New Version of Converged Application Server

Follow these steps to upgrade a production installation of Converged Application Server to a newer Service Pack version of the Converged Application Server software. These instructions upgrade both the SIP Servlet container implementation and the SIP data tier replication and failover implementation.

The steps for performing a software upgrade are divided into several high-level procedures:

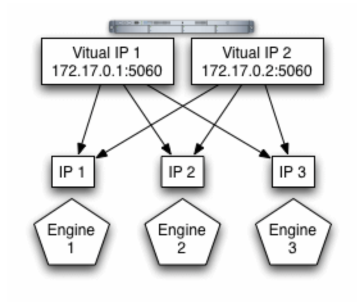
1. Configure the Load Balancer: Define a new, internal Virtual IP address for the new engine tier cluster you will configure.
2. Configure the New Engine Tier Cluster: Create and configure a new, empty engine tier cluster that will host upgraded engine tier servers and your converged applications.
3. Define the Cluster-to-Load Balancer Mapping: Modify the SIP Servlet container configuration to indicate the virtual IP address of each engine tier cluster.
4. Duplicate the SIP Servlet Configuration: Copy the active **sipserver.xml** configuration file to duplicate your production container configuration.
5. Upgrade Engine Tier Servers and Target Applications to the New Cluster: Shut down individual engine tier server instances, restarting them in the new engine tier cluster.
6. Upgrade SIP Data Tier Servers: Shut down individual SIP data tier servers, restarting them with the new SIP data tier software implementation.

Each procedure is described in the sections that follow.

Configure the Load Balancer

Begin the software upgrade procedure by defining a new internal virtual IP address for the new engine tier cluster you will create in "Configure the New Engine Tier Cluster". The individual server IP addresses (the pool definition) for the new virtual IP address should be identical to the pool definition of your currently-active engine tier cluster. Figure 10–1 shows a sample configuration for a cluster having three engine tier server instances; both virtual IP addresses define the same servers.

Figure 10–1 Virtual IP Address Configuration for Parallel Clusters



See your load balancer documentation for more information about defining virtual IP addresses.

In the next section, you will configure the Converged Application Server domain to identify the virtual IP addresses that map to each engine tier cluster. Converged Application Server uses this mapping during the upgrade procedure to automatically forward requests to the appropriate "version" of the cluster. This ensures that SIP data tier requests always originate from a compatible version of the Converged Application Server engine tier.

Configure the New Engine Tier Cluster

Follow these steps to create a new Engine Tier cluster to host upgraded containers, and to configure both clusters in preparation for a software upgrade:

1. On the Administration Server machine, install the new Converged Application Server software into a new ORACLE_HOME\Middleware home directory. The steps that follow refer to `c:\Oraclenew` as the ORACLE_HOME\Middleware home directory in which the new software was installed. `c:\Oracle` refers to the software implementation that is being upgraded.
2. Log in to the Administration Console for the active Converged Application Server domain.
3. In the Administration Console, create a new, empty engine tier cluster for hosting the upgraded engine tier servers:
   a. In the left pane, select the **Environment** node, and then select the **Clusters** node.
   b. Click New to configure a new cluster.
   c. Enter a name for the new cluster. For example, "NewEngineCluster."
   d. Enter a unicast channel or multicast address and port number for the cluster.

   e. Click OK to create the cluster.

4. Proceed to "Define the Cluster-to-Load Balancer Mapping".

Define the Cluster-to-Load Balancer Mapping

In this procedure, you configure the cluster-to-load balancer mapping to define the internal, virtual IP address that is assigned to the older and newer engine tier clusters.

To define the cluster-to-load balancer mapping:

1. Log in to the Administration Console for the active Converged Application Server domain.
2. In the Administration Console, create a new load balancer map entry for the new cluster you created:
   a. In the left pane, select the **SipServer** node.
   b. Select **Configuration**, then select the **Loadbalancer Map** tab.
   c. Click New to create a new mapping.
   d. Enter the Cluster Name and SIP URI of the new cluster you created in Configure the New Engine Tier Cluster. For example, enter "NewEngineCluster" and "sip:172.17.0.2:5060".
   e. Click Finish to create the new mapping.
3. Repeat Step 2 to enter a mapping for the older cluster. For example, create an entry for "EngineCluster" and "sip:172.17.0.1:5060".

   Note that you must include a mapping for both the older and the newer engine tier cluster. A mapping consists of the internal virtual IP address of the cluster configured on the load balancer, as well as the cluster name defined in the Converged Application Server domain. Example 10–1 shows an entry in the **sipserver.xml** file for the sample clusters described earlier.

Example 10–1 Sample cluster-loadbalancer-map Definition

```
   <cluster-loadbalancer-map>
      <cluster-name>EngineCluster</cluster-name>
      <sip-uri>sip:172.17.0.1:5060</sip-uri>
   </cluster-loadbalancer-map>
   <cluster-loadbalancer-map>
      <cluster-name>NewEngineCluster</cluster-name>
      <sip-uri>sip:172.17.0.2:5060</sip-uri>
   </cluster-loadbalancer-map>
</sip-server>
```

Duplicate the SIP Servlet Configuration

Before upgrading individual engine tier servers, you must ensure that the SIP Servlet container configuration and SIP data tier configuration in the new engine tier cluster matches your current production configuration. To duplicate the container configuration, copy your production **sipserver.xml** and **datatier.xml** configuration files to the new installation. For example:

```
cp c:\Oracle\Middleware\user_projects\domains\mydomain\config\custom\*.xml
c:\Oraclenew\Middleware\user_projects\domains\mydomain\config\custom
```

As engine tier servers are restarted in the new engine tier cluster in the next procedure, they will have the same SIP container configuration but will use the new container implementation.

Upgrade Engine Tier Servers and Target Applications to the New Cluster

To upgrade individual engine tier servers, you gracefully shut each server down, change its cluster membership, and then restart it. Follow these steps:

1. Access the Administration Console for your production domain.
2. Select the first running engine tier server that you want to upgrade:
   a. Select **Environment**, then select the **Servers** tab in the left pane.
   b. Select the **Control** tab in the right pane.
   c. Select the name of the server you want to upgrade.
3. Select **Shutdown**, then select **When work completes from the table**.

   The server remains active while clients are still accessing the server, but no new connection requests are accepted. After all existing client connections have ended or timed out, the server shuts down. Other server instances in the engine tier process client requests during the shutdown procedure.
4. Select **Environment**, then select the **Servers** tab in the left pane and verify that the Managed Server has shut down.
5. Change the stopped server's cluster membership so that it is a member of the new engine tier cluster:
   a. Expand the **Environment** node, and the select the **Clusters** tab in the left pane.
   b. Select the name of the active engine tier cluster.
   c. Select **Configuration**, and then select the **Servers** tab in the right pane.
   d. Select the checkbox next to the name of the stopped server, and click Remove.
   e. Click Yes to remove the server from the cluster.
   f. Next, expand the **Environment** node, then select the **Clusters** tab and select the newly-created engine tier cluster ("NewEngineCluster").
   g. Select **Configuration**, then select the **Servers** tab in the right pane.
   h. Click **Add** to add a new server.
   i. Select the name of the stopped server from the drop-down list.
   j. Click Finish.
6. After adding the first engine tier server to the new cluster, you can now target your own applications to the new cluster.

   Note: Perform this step only once, after adding the first engine tier server to the new cluster:

   ---

   a. In the left pane, select the Deployments node.
   b. Click Install.
   c. Use the links to select an application to deploy, and click Next.

d. Select Install this deployment as an application, and click Next.

e. Select the name of the new engine tier cluster ("NewEngineCluster"). Also ensure that All servers in the cluster is selected.

f. Click Finish.

g. Select the checkbox next to the name of the application in the Deployments table.

h. Click **Start**, then select **Servicing all requests**.

i. Repeat this step to deploy all of your applications to the new cluster. Both the new and old engine tier clusters should be configured similarly.

7. Restart the stopped managed server to bring it up in the new engine tier cluster:

   a. Access the machine on which the stopped engine tier server runs (for example, use a remote desktop on Windows, or secure shell (SSH) on Linux).

   b. Use the available Managed Server start script (startManagedWebLogic.cmd or startManagedWebLogic.sh) to boot the Managed Server. For example:

   ```
   startManagedWebLogic.cmd engine-server1 t3://adminhost:7001
   ```

8. In the Administration Console, select **Environment**, then select the **Servers** node and verify that the Managed Server has started.

9. Repeat these steps to upgrade the remaining engine tier servers.

At this point, all running Managed Servers are using the new SIP Container implementation and are hosting your production SIP Servlets with the same SIP Servlet container settings as your old configuration. SIP data tier servers can now be upgraded using the instructions in "Upgrade SIP Data Tier Servers".

Upgrade SIP Data Tier Servers

Caution: Your SIP data tier must have three active replicas (three server instances) in each partition in order to upgrade the servers in a production environment. With only two replicas in each partition, a failure of the active replica during the upgrade process will result in the irrecoverable loss of call state data. With only one replica in each partition, the upgrade cannot be initiated without losing call state data.

The procedure for upgrading server instances in the SIP data tier simply involves restarting each server, one at a time, to utilize the updated software. Always ensure that the previous server has fully restarted (is in `ONLINE` state) before restarting the next server.

Caution: Do not shut down or restart a SIP data tier server instance in a partition unless two additional servers in the same partition are available, and both are in the `ONLINE` state. See the discussion on "Monitoring and Troubleshooting SIP Data Tier Servers" in Chapter 4, "Configuring SIP Data Tier Partitions and Replicas," for information about determining the state of SIP data tier servers.

Applying Software Patches and Updates

Oracle may occasionally release software patches and updates to address known bugs and limitations in the Converged Application Server software. To upgrade Converged Application Server, you apply patches using the Ant build tool.

The process for applying patches to Converged Application Server is as follows:

1. The patch, in the form of a JAR file, is provided by Oracle Corporation.
2. The patch is applied to the original EAR file, which results in a new, patched, EAR. For example, if the original EAR version is 5.0, the version of the new EAR is 5.0_1. A sequence number is always suffixed to make each version unique.
3. The new EAR file is deployed and the old version is undeployed.
4. The patched EAR file is deployed.

Applying Patches with Ant

The syntax of `ant` is shown below, and the arguments it takes are described in Table 10–1.

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

Table 10–1 Description of the Arguments for the Ant Build Tool

Argument	Description
`path`	This target applies a patch.
`printpatches`	This target displays the currently applied patches. The information displayed by this option includes: ■ Manifest for the EAR ■ Manifest-Version ■ Bundle-Name ■ Created-By ■ Ant-Version ■ WebLogic-Application-Version ■ Bundle-Vendor ■ Bundle-Version ■ For each JAR contained by the EAR, the class and ID of the patch is displayed.
`src`	The EAR file to apply the patch to.
`dest`	The EAR file that will contain the patch.
`version`	The new version for the EAR. Must be different from the original version.
`patchdir`	The directory containing the patches. The directory paths are relative or absolute.
`patchfiles`	The JAR files that contains the patches. Wild cards are supported.

To apply patches with the Ant build tool:

1. After installing the Converged Application Server software, and configuring the domains for your deployment, run the `setDomainEnv.sh` (UNIX) or `setDomainEnv.cmd` (Windows) command located in `$DOMAIN_HOME/bin` to set the

environment. You must set the environment for your deployment prior to using `ant` to apply patches.

2. Apply the patch using the `ant` command. For example, to apply a single patch:

   ```
   % ant patch -Dsrc=original_filename.ear -Ddest=patched_filename.ear
   -Dversion=5.0.1 -Dpatchdir=directory -Dpatchfiles=filename.jar
   ```

   where:

   `original_filename.ear` is the original EAR file to which you are applying the patch.

   `patched_filename.ear` is the EAR file containing the patch.

   `directory` is the directory containing the patch.

   `filename.jar` is the JAR file containing the patch.

3. The `ant` build tool applies the patch to the Converged Application Server EAR file.

4. Verify that the patch was applied properly using the `printpatches` option to view information about the applied patches.

   ```
   % ant printpatches -Dsrc=filename.ear
   ```

11 Upgrading Deployed SIP Applications

This chapter describes how to upgrade deployed SIP Servlets and converged SIP/HTTP applications in Oracle Communications Converged Applications Server to a newer version of the same application without losing active calls:

Overview of SIP Application Upgrades

With Converged Applications Server, you can upgrade a deployed SIP application to a newer version without losing existing calls being processed by the application. This type of application upgrade is accomplished by deploying the newer application version alongside the older version. Converged Applications Server automatically manages the SIP Servlet mapping so that new requests are directed to the new version. Subsequent messages for older, established dialogs are directed to the older application version until the calls complete. After all of the older dialogs have completed and the earlier version of the application is no longer processing calls, you can safely undeploy it.

Converged Applications Server's upgrade feature ensures that no calls are dropped while during the upgrade of a production application. The upgrade process also enables you to revert or rollback the process of upgrading an application. If, for example, you determine that there is a problem with the newer version of the deployed application, you can undeploy the newer version and activate the older version.

Note: When you undeploy an active version of an application, the previous application version remains in administration mode. You must explicitly activate the older version in order to direct new requests to the application.

You can also use the upgrade functionality with a SIP administration channel to deploy a new application version with restricted access for final testing. After performing final testing using the administration channel, you can open the application to general SIP traffic.

Converged Applications Server application upgrades provide the same functionality as *Oracle WebLogic Server 11g release 1 patch set 2* application upgrades, with the following exceptions:

- Converged Applications Server does not support "graceful" retirement of old application versions. Instead, only timeout-based undeployment is supported using the `-retiretimeout` option to `weblogic.Deployer`.
- If you want to use administration mode with SIP Servlets or converged applications, you must configure a `sips-admin` channel that uses TLS transport.
- Converged Applications Server handles application upgrades differently in replicated and non-replicated environments. In replicated environments, the server behaves as if the `save-sessions-enabled` element was set to "true" in the **weblogic.xml** configuration file. This preserves sessions across a redeployment operation.

  For non-replicated environments, sessions are destroyed immediately upon redeployment.

See the discussion on redeploying applications in a production environment in the *Deploying Applications to Oracle WebLogic Server* in the Oracle WebLogic Server 11*g* documentation for general information and instructions regarding production application redeployment.

Requirements and Restrictions for Upgrading Deployed Applications

To use the application upgrade functionality of Converged Applications Server:

- You must assign version information to your updated application in order to distinguish it from the older application version. Note that only the newer version of a deployed application requires version information; if the currently-deployed application contains no version designation, Converged Applications Server automatically treats this application as the "older" version. See "Assign a Version Identifier".
- A maximum of two different versions of the same application can be deployed at one time.
- If your application hard-codes the use of an application name (for example, in composed applications where multiple SIP Servlets process a given call), you must replace the application name with calls to a helper method that obtains the base application name. WebLogic Server provides `ApplicationRuntimeMBean` methods for obtaining the base application name and version identifier, as well as determining whether the current application version is active or retiring. See "Accessing the Application Name and Version Identifier".
- When applications take part in a composed application (using application composition techniques), Converged Applications Server always uses the latest version of an application when only the base name is supplied.
- If you want to deploy an application in administration mode, you must configure a `sips-admin` channel that uses TLS transport. See "Creating a New SIP or SIPS Channel" in Chapter 5, "Configuring Network Connection Settings" for more information.

Steps for Upgrading a Deployed SIP Application

Follow these steps to upgrade a deployed SIP application to a newer version:

1. Assign a Version Identifier: Package the updated version of the application with a version identifier.
2. Deploy the Updated Application Version: Deploy the updated version of the application alongside the previous version to initiate the upgrade process.
3. Undeploy the Older Application Version: After the older application has finished processing all SIP messages for its established calls, you can safely undeploy that version. This leaves the newly-deployed application version responsible for processing all current and future calls.

Each procedure is described in the sections that follow. You can also roll back the upgrade process if you discover a problem with the newly-deployed application. Applications that are composed of multiple SIP Servlets may also need to use the `ApplicationRuntimeMBean` for accessing the application name and version identifier.

Assign a Version Identifier

Converged Applications Server uses a version identifier—a string value—appended to the application name to distinguish between multiple versions of a given application. The version string can be a maximum of 215 characters long, and must consist of the following characters:

- a-z
- A-Z
- 0-9
- period ("."), underscore ("_"), or hyphen ("-") in combination with other characters

For deployable SIP Servlet WAR files, you must define the version identifier in the MANIFEST.MF file of the application or specify it on the command line at deployment time. See the discussion on specifying an application Version Identifier in the Oracle WebLogic Server 11*g* documentation for more information.

Defining the Version in the Manifest

Both WAR and EAR deployments must specify a version identifier in the MANIFEST.MF file. Example 11–1 shows an application with the version identifier "v2":

Example 11–1 Version Identifier in Manifest

```
Manifest-Version: 1.0
Created-By: 1.4.1_05-b01 (Sun Microsystems Inc.)
Weblogic-Application-Version: v2
```

If you deploy an application without a version identifier, and later deploy with a version identifier, Converged Applications Server recognizes the deployments as separate versions of the same application.

Deploy the Updated Application Version

To begin the upgrade process, simply deploy the updated application archive using either the Administration Console or `weblogic.Deployer` utility. Use the

-retiretimeout option to the weblogic.Deployer utility if you want to automatically undeploy the older application version after a fixed amount of time. For example:

```
java weblogic.Deployer -name MyApp -version v2 -deploy -retiretimeout 7
```

Converged Applications Server examines the version identifier in the manifest file to determine if another version of the application is currently deployed. If two versions are deployed, the server automatically begins routing new requests to the most recently-deployed application. The server allows the other deployed application to complete in-flight calls, directs no new calls to it. This process is referred to as "retiring" the older application, because eventually the older application version will process no SIP messages.

Note that Converged Applications Server does not compare the actual version strings of two deployed applications to determine which is the higher version. New calls are always routed to the most recently-deployed version of an application.

Converged Applications Server also distinguishes between a deployment that has no version identifier (no version string in the manifest) and a subsequent version that does specify a version identifier. This enables you to easily upgrade applications that were packaged before you began including version information as described in "Assign a Version Identifier".

Undeploy the Older Application Version

After deploying a new version of an existing application, the original deployment process messages only for in-flight calls (calls that were initiated with the original deployment). After those in-flight calls complete, the original deployment no longer processes any SIP messages. In most production environments, you will want to ensure that the original deployment is no longer processing messages before you undeploy the application.

To determine whether a deployed application is processing messages, you can obtain the active session count from the application's `SipApplicationRuntimeMBean` instance. Example 11–2 below shows the sample WLST commands for viewing the active session count for the `findme` sample application on the default single-server domain.

Based on the active session count value, you can undeploy the application safely (without losing any in-flight calls) or abruptly (losing the active session counts displayed at the time of undeployment).

Use either the Administration Console or `weblogic.Deployer` utility to undeploy the correct deployment name.

Example 11–2   Sample WLST Session for Examining Session Count

```
connect()
custom()
cd ('examples:Location=myserver,Name=myserver_myserver_findme_
findme,ServerRuntime=myserver,Type=SipApplicationRuntime')
ls()
-rw-   ActiveAppSessionCount                       0
-rw-   ActiveSipSessionCount                       0
-rw-   AppSessionCount                             0
-rw-   CachingDisabled                             true
-rw-   MBeanInfo                                   weblogic.management.tools.In
fo@5ae636
-rw-   Name                                        myserver_myserver_findme_fin
dme
-rw-   ObjectName                                  examples:Location=myserver,N
ame=myserver_myserver_findme_findme,ServerRuntime=myserver,Type=SipApplicationRu
```

```
ntime
-rw-   Parent                                    examples:Location=myserver,N
ame=myserver,Type=ServerRuntime
-rw-   Registered                                false
-rw-   SipSessionCount                           0
-rw-   Type                                      SipApplicationRuntime

-rwx   preDeregister                             void :
```

Roll Back the Upgrade Process

If you deploy a new version of an application and discover a problem with it, you can roll back the upgrade process by:

1. Undeploying the active version of the application.
2. Activating the older version of the application. For example:

   ```
   java weblogic.Deployer -name MyApp -appversion v1 -start
   ```

Note: When you undeploy an active version of an application, the previous application version remains in administration mode. You must explicitly activate the older version in order to direct new requests to the application.

Alternatively, you can use simply use the `-start` option to start the older application version, which causes the older version of the application to process new requests and retire the newer version.

Accessing the Application Name and Version Identifier

If you intend to use Converged Applications Server's production upgrade feature, applications that are composed of multiple SIP Servlets should not hard-code the application name. Instead of hard-coding the application name, your application can dynamically access the deployment name or version identifier by using helper methods in `ApplicationRuntimeMBean`. See the discussion on `ApplicationRuntimeMBean` in the Oracle WebLogic Server 11*g* documentation for more information.

Using Administration Mode

You can optionally use the `-adminmode` option with `weblogic.Deployer` to deploy a new version of an application in administration mode. While in administration mode, SIP traffic is accepted only via a configured network channel named `sips-admin` having the TLS transport. If no `sips-admin` channel is configured, or if a request is received using a different channel, the server rejects the request with a 503 message.

To transition the application from administration mode to a generally-available mode, use the `-start` option with `weblogic.Deployer`.

Note: If using TLS is not feasible with your application, you can alternatively change the Servlet role mapping rules to allow only 1 user on the newer version of the application. This enables you to deploy the newer version alongside the older version, while restricting access to the newer version.

Part II

Configuring Infrastructure Components

Depending on the topology and requirements for the deployment, the Converged Application Server SIP applications may rely on related infrastructure components to operate. These components include, for example, proxy registrar and load balancers. This part describes the components included with the Oracle Communications Converged Application Server.

This part contains the following chapters:

12

Configuring the Proxy Registrar

This chapter describes how to configure a proxy registrar for the Oracle Communications Converged Application Server deployment.

About Proxy Registrar Configuration

The Administration Console exposes the Proxy Registrar MBean attributes that are used to set Proxy and Registrar parameters. Only those parameters and attributes that you typically need to set are exposed in the Administration Console. To modify advanced parameters and attributes, you can modify MBean attributes by using WebLogic Scripting Tool (WLST).

For information about the Proxy and Registrar MBeans, see the Converged Application Server Java API Reference.

Some Proxy Registrar configurations, such as security settings, require that you edit the **sip.xml** deployment descriptor. If you modify **sip.xml**, you must redeploy the Proxy Registrar for the changes to take effect.

Setting Authentication for the Proxy Registrar

Authentication for the Proxy and Registrar is defined in a `security-constraint` element in the **sip.xml** deployment descriptor. Proxy and Registrar authentication is enabled by default. You can disable authentication for the Proxy, Registrar, or both by removing their respective section from the `security-constraint` element:

- To disable Registrar authentication, remove the `registrar servlet` section.
- To disable Proxy authentication, remove the `VoipProxy Servlet` section.

The type of authentication for SIP requests is defined in the `auth-method` subelement of the `login-config` element in **sip.xml**. Converged Application Server supports DIGEST, BASIC and CLIENT-CERT authentications. DIGEST authentication is the default. For more information, see the discussion of authentication for SIP servlets in the *Oracle Communications Converged Application Server Security Guide*.

You can also set the following authentication policy:

- Trusted hosts:

  You can bypass authentication for certain hosts by adding trusted-host definitions in the `sip-security` element. For more information, see "sip-security" in Chapter 21, "Engine Tier Configuration Reference (sipserver.xml)."

Note: You can also configure trusted hosts by using the Administration Console. See "Using the Administration Console to Configure Trusted Hosts" for instructions.

- Identity assertion mode:

  You can set the `identity-assertion` element in **sip.xml** to specify either `P-Asserted-Identity` or `Identity`.

  For more information, see the discussion of SIP servlet identity assertion in the *Oracle Communications Converged Application Server Security Guide*.

- Security provider:

  You configure security providers by using the Administration Console:

  – If you set the identity assertion mode to `P-Asserted-Identity`, then configure a P-Asserted-Identity Assertion Provider. Be sure to set its **Trusted Hosts** parameter.

  – If you set the identity assertion mode to `Identity`, then configure an Identity Header Assertion Provider.

  For more information, see the discussion of SIP servlet identity assertion in the *Oracle Communications Converged Application Server Security Guide*.

Using the Administration Console to Configure Trusted Hosts

You can specify to bypass authentication for certain hosts by adding trusted-host definitions through the Administration Console.

To add trusted hosts:

1. Log in to the Administration Console.
2. In the Domain Structure panel, click **SipServer**. The SIP Server configuration options are displayed in the main window.
3. Click the **Configuration** tab and then the **SIP Security** tab.
4. Enter any trusted hosts and click **Save**.
5. Stop and restart all servers in the domain. See Chapter 2, "Getting Started."

Configuring the Proxy

To configure the Proxy:

1. Log in to the Administration Console.

   For information on using the Administration Console, see the WebLogic Server documentation.

2. In the Domain Structure panel, click **ProxyRegistrar**. The Proxy Registrar configuration options are displayed in the main window.
3. Click the **Proxy** tab.
4. Configure the following Proxy parameters:

Note: Parameter that are preceded by an exclamation mark (!) indicate that you must restart the servers for the new value to take effect. Be sure to restart all servers in the domain to ensure that the new values are propagated to all servers. Chapter 2, "Getting Started."

- **!Enable ENUM Lookup**: Select this option to enable ENUM lookup. If ENUM lookup is disabled and the request URI is a Tel URL, then the request will be rejected with a 404 response.
- **!ENUM DNS Server**: Specify the address of the DNS server.
- **!ENUM Zone**: This is the DNS Zone used by ENUM. The default domain "e164.arpa" provides the infrastructure in the DNS for storage of E.164 numbers.
- **!ENUM SIP Service Field**: Specify which service field of the DNS NAPTR record to retrieve. The default, E2U+sip, is the common value for SIP applications.
- **!ENUM Preset DNS Response**: Specify a comma-separated list of predefined DNS responses. If you set this value, the Proxy does not consult the DNS server for ENUM lookup. Instead, the Proxy uses the **ENUM Preset DNS Response** value as the response from the DNS server.
- **Force Record Route**: Select this option if you want all subsequent requests to go through the Proxy. The default is True.
- **Parallel Forking**: Indicates whether a call that has more than one destination is routed to those destinations in parallel or sequentially. The default is parallel.
- **TimeOut Length**: Specify the period in seconds that the Proxy waits for a final response to a request. The default is 180 seconds.

  If Parallel Forking is set to sequential, the time-out period is the time the container waits for a final response from a branch before it CANCELs the branch and proxies the next destination in the target set.

  If Parallel Forking is set to Parallel, the time-out period is time the container waits for final responses from the entire proxy (that is, all of the parallel branches) before it CANCELs the branches. With parallel forking, the container sends the best final response upstream.
- **!Hosted Domains**: Only set this value if you set **Use Domain Service** to false. Specify a comma-separated list of the domain names for the domains hosted by this server. To specify any domain name, use an asterisk: *
- **!Use Domain Service**: Select this option to use the Domain Service to determine which domains are hosted by the server. If set to false (not selected), then specify the domains in the **Hosted Domains** parameter to specify the domains.
- **Offline Code**: Specify the status code that is returned when a SIP client is offline.

5. If needed, configure advanced parameters. You typically do not need to do this and can accept the default values.

   a. At the bottom of the Proxy window, click **Advanced** to display the advanced parameters.

b. If needed, set the following parameters:

!**Options**: This specifies any option tags supported by the Proxy. When receiving a request that has a Proxy-Require header, all option tags listed in the Proxy-Require header must be specified in the Options field. If not specified, the proxy will reject the request with a 420 response. The value is a comma-separated list of options. For example, “privacy, sec-agree”.

!**Location Service**: Specify the location service class name.

6. Click **Save**.
7. If you changed the value of any parameter that requires restarting the server, stop and restart all servers in the domain. See Chapter 2, "Getting Started."

Configuring the Registrar

To configure the Registrar:

1. Log in to the Administration Console.
2. In the Domain Structure panel, click **ProxyRegistrar**. The Proxy Registrar configuration options are displayed in the main window.
3. Click the **Registrar** tab.
4. Configure the following Registrar parameters:

Note: Parameter that are preceded by an exclamation mark (!) indicate that you must restart the servers for the new value to take effect. Be sure to restart all servers in the domain. See Chapter 2, "Getting Started."

- **Max Contacts**: Specifies the maximum allowed number of Contact headers in a REGISTER request and the maximum number of contacts that will be stored as a result of multiple REGISTER requests. The default is 5.
- **Max Expires**: Specifies in seconds, the maximum allowed expiration time for registrations. If a request is made that exceeds this maximum, a response is sent that the time is too long. The default is 7200 seconds. **Max Expires** must be greater than the value of **Min Expires**.
- **Min Expires**: Specifies in seconds, the minimum allowed expiration time for registrations. If a request is made that is shorter than this minimum, a response is sent that the time is too brief. The default is 60 seconds. **Min Expires** must be less than the value of **Max Expires**.
- **Default Expires**: Specifies in seconds, the default expiration time for registrations. If an expiration time is not requested, the default is used. The default value is 3600 seconds. **Default Expires** must be greater than the value of **Min Expires** and less than the value of **Max Expires**.

Important: The Administration Console does not validate that the expiration times are logically set. For example, if you set **Default Expires** to 3600 seconds and set **Max Expires** to 2700 seconds, the Administration Console will accept the configuration, but registrations may be denied.

5. If needed, configure advanced parameters. You typically do not need to do this and can accept the default values.
   a. At the bottom of the Registrar window, click **Advanced** to display the advanced parameters.
   b. Set the following parameters:

      Supported Methods: Specify the supported methods to insert into the Contact header of REGISTER request responses when the REGISTER request itself does not include supported methods in the Allow header or the Contact header.

      The format of this value is the same as the methods parameter in the contact header.

      Enable Associated ID: Indicates whether to add the P-Associated-URI header when the Registrar sends a response.

      !**Location Service House Keeper Period**: The house keeper removes expired registrations. Specify in seconds, the period between house-keeper runs. A negative value means to run the house keeper only once, when the Proxy Registrar application is started. The default is 300 seconds.

      !**Location Service House Keeper Delay**: Specify in seconds, the time to wait after the Proxy Registrar application is started, before the house keeper is started. A negative value turns off the house keeper. The default is 120 seconds.

      !**Location Service**: Specify the location service class name.
6. Click **Save**.
7. If you changed the value of any parameter that requires restarting the server, stop and restart all servers in the domain. See Chapter 2, "Getting Started."

Provisioning Users

You can use Sash to provision users for the proxy registrar. Sash is a command-line utility for provisioning Converged Application Server users to the database, to the XML Document Management Server (XDMS), and to the RADIUS server. You can provision users from the Sash command line prompt (`sash#`) or by using the CommandService MBean.

See *Oracle Fusion Middleware Administrator's Guide for Oracle Internet Directory* for information on using Oracle Internet Database (OID) as the user provisioning repository for a Converged Application Server deployment.

Launching Sash

The Sash launcher script is located in the same folder that contains the start and stop scripts for Converged Application Server.

Launching Sash from the Command Line

Converged Application Server provides the following scripts for launching Sash from the command line:

`launch_sash.sh` (UNIX)

`launch_sash.cmd` (Windows)

These scripts are located at *domain_home*`/bin`, where *domain_home* is the home directory of the domain.

Connecting Sash to an External Converged Application Server Instance

By default, Sash connects to the local instance of Converged Application Server. If needed, you can override this default behavior and connect Sash to external instances of Converged Application Server.

Connecting to an External Instance of Converged Application Server Sash connects to the Converged Application Server server through RMI. The following example illustrates how to connect Sash to a Converged Application Server instance with the host IP address 10.1.10.23:

```
sash --host 10.1.10.23
```

When you connect to Converged Application Server, Sash prompts you for a username and a password. The user name is the same as that for Converged Application Server administrator. The password is the same as the password associated with the Converged Application Server administrator. Once you log in, the Sash command prompt (`sash`) appears. An error message displays if the login is unsuccessful.

Using Sash

There are two groups of Sash commands:

- Commands that create, delete, and update system objects
- Commands that query the system for information

Note: Whenever a user adds a new application usage, the user must restart the server before the new application usage is available.

Whenever a user deletes an existing application usage, the user must restart the server for the deleted application usage to be completely unloaded (that is, a deleted application usage will remain loaded until the server is restarted, when it is unloaded and is then completely unavailable).

If a space precedes a sash command in a file, and then that file is used as input to the sash command, it does not work. Ensure that you remove any preceding spaces in sash commands in sash input files.

Viewing Available Commands

Entering `help` displays a list of all available commands in the server (described in Table 12–1). The list of commands varies depending on the components deployed to the server.

Table 12–1 Shapphire Shell (Sash) Commands

Command	Description	Aliases	Subcommands
`privateIdentity`	Commands for adding and removing private communication identities used for authentication.	None	Subcommands include: ■ `add` – Adds a new user to the system. For example: `privateIdentity add privateId=alice` ■ `delete` – Removes a user from the system. For example: `privateIdentity delete privateId=`alice

`publicIdentity`	Commands for adding and removing public identities associated with a private identity.	`pubid`	Subcommands include:
			■ `add` – Adds a public identity to the system which is associated with a particular user. For example:
			`publicIdentity add publicId=sip:alice@test.company.com privateId=alice`
			■ `delete` – Deletes a communication identity from the system. For example:
			`publicIdentity delete publicId=sip:alice@test.company.com privateId=alice`

`account`	Contains commands for managing user accounts. This command enables you to set the account as active, locked, or as a temporary account.	None	Subcommands include:
			■ `add` – adds a new account to the system. The syntax is as follows:
			`account add uid=<string> [active=<true|false>] [locked=<true|false>] [accountExpiresAt=<accountExpiresAt>] [tempAccount=<true|false>] [description=<string>] [lockExpiresAt=<lockExpiresAt>] [currentFailedLogins=<integer>]`
			For example: `account add uid=alice active=true`
			■ `delete` – Deletes an account from the system. For example: `account delete uid=<string>`
			■ `update` – Updates an account. For example:
			`account update uid=<string> [active=<true|false>] [locked=<true|false>] [accountExpiresAt=<accountExpiresAt>] [tempAccount=<true|false>] [description=<string>] [lockExpiresAt=<lockExpiresAt>] [currentFailedLogins=<integer>]`
			■ info – Retrieves information for a specific account. For example: `account info uid=<string>`

Table 12–1 (Cont.) Shapphire Shell (Sash) Commands

Command	Description	Aliases	Subcommands
`role`	Manages role types and user roles in the system. `role` is an additional security and authorization mechanism that is defined within the `<auth-constraint>` element of **sip.xml**. This command authorizes a group of users access to applications. The applications in turn check for a specific role. Converged Application Server defines one role for the Proxy Registrar application, "Location Services".	None	Subcommands include `role system` and `role user`.

`role system` (subcommand of `role`)	Manages the roles types.	None	Subcommands include:
			■ `list` – Lists the roles in the system. For example:
			`role system list`
			■ `add` – Adds a new role to the system. For example:
			`role system add name=<string> [description=<string>]`
			■ `update` – Updates a role in the system. For example:
			`role system update name=<string> [description=<string>]`
			■ `delete` – Deletes a role from the system. For example:
			`role system delete name=<string> [description=<string>]`

Table 12–1 (Cont.) Shapphire Shell (Sash) Commands

Command	Description	Aliases	Subcommands

`role user` (subcommand of `role`)	Manages the user roles	None	Subcommands include:
			■ `add` – Adds a role to a user. For example:
			`role user add uid=<string> name=<string>`
			■ `delete` – Deletes a role from a user. For example:
			`role user delete uid=<string> name=<string>`
			■ `list` – Lists roles for a user. For example:
			`role user list uid=<string>`

credentials	Command for managing credentials.	None	Subcommands include:
			■ add – Adds credentials to a user. For example:
			`credentials add password=<string> realm=<string> uid=<string>`
			■ `addAll` – Adds credentials for all of the configured realms in the system to a user. For example:
			`credentials addAll password=<string> uid=<string>`
			■ `delete` – Deletes realm credentials for a user. For example:
			`credentials delete realm=<string> uid=<string>`
			■ `deleteAll` – Deletes all credentials for a user. For example:
			`credentials deleteall uid=<string>`
			■ `update` – Updates the credentials for a user. For example:
			`credentials update password=<string> realm=<string> uid=<string>`
			■ `updateAll` – Updates a user's credentials for all provisioned realms in the system. For example:
			`credentials updateAll password=<string> uid=<string>`
			■ `list` – Lists all of the realms for which credentials exist for a given user. For example:
			`credentials list uid=<string>`

`identity add`	Enables you to create a basic user account.	None	None. See "Creating a User with the Identity Add Command" for more information.

Viewing Subcommands To view the subcommands for a specific command, enter `help <command>`. For example, entering *`help`* for the `account` command (`help account`) retrieves a brief overview of the subcommands available to the `account` command (illustrated in Example 12–1).

Example 12–1 Retrieving Help for a Specific Command

```
*** Description ****
Contains commands for management of user accounts.
In an account you can set if the account is active,
locked or if it perhaps should be a temporarily account.

Aliases: [no aliases]
```

```
Syntax:
account

Sub-commands:
# Adds a new account to the system
  account add uid=<string> [ active=<true|false> ] [ locked=<true|false> ] [
accountExpiresAt=<accountExpiresAt> ] [ tempAccount=<true|false> ] [
description=<string> ] [ lockExpiresAt=<lockExpiresAt> ] [
currentFailedLogins=<integer> ]

# Deletes an account
  account delete uid=<string>

# Updates an account
  account update uid=<string> [ active=<true|false> ] [ locked=<true|false> ] [
accountExpiresAt=<accountExpiresAt> ] [ tempAccount=<true|false> ] [
description=<string> ] [ lockExpiresAt=<lockExpiresAt> ] [
currentFailedLogins=<integer> ]

# Retrieve information about a particular account
  account info uid=<string>
```

In addition to the overview of the command group, the information displayed by entering `help <command>` also includes the aliases (if any) to the command. For example, the overview of the account command illustrated in Example 12–1 notes `[no aliases]` for the command.

Note: The `delete` command used with `account`, `role`, `role system`, `role user`, `privateIdentity`, `publicIdentity`, and `identity` has the following aliases:

- `remove`
- `del`
- `rm`

Some commands require parameters. For example, if you enter `help role system add`, the system informs you that the `add` command requires the name of the role and an optional command for setting the description as well by displaying:

```
role system add name=<string> [description=<string>].
```

Note: Optional commands such as `[description=<string>]` are enclosed within square brackets `[...]`.

The system alerts you if you omit a mandatory parameter or if you pass in a parameter that is not recognized.

Creating a User

This section describes the `publicIdentity` and `privateIdentity` commands and how to use them in conjunction with the `add`, `account`, `role`, and `credentials` subcommands listed in Table 12–1 to provision a user account to the Oracle database.

The Private Identity (`privateIdentity`) uniquely identifies a user within a given authentication realm. The Public Identity (`publicIdentity`) is the SIP address that users enter to register devices. This address is the user's Address of Record (AOR) and the means through which users call one another. A user can have only one Private Identity, but can have several Public Identities associated with that Private Identity.

Note: To enable authentication to third-party databases (such as RADIUS), user accounts that contain authentication data and are stored externally must match the Private Identity to ensure the proper functioning of the Proxy Registrar and other applications that require authentication.

To create a user, first add the user to the system by creating a private identity and then a public identity for the user using the `privateIdentity` and `publicIdentity` commands with the `add privateId` and `add publicId` subcommands, respectively.

After you create the private and public identity for the user, create an account for the user with the `account add uid` command and optionally set the status of the account (such as active or locked). The `role` command sets the role memberships for role-based permissions. Set the level of permissions for the users using the `role` command, and then set user credentials by defining the user's realm and password with the `credentials` command.

Creating a User from the Sash Command-Line Prompt

This section illustrates how to create a user from the Sash command prompt (`sash#`, illustrated in Example 12–2, "Creating a User from the Sash Command-Line Prompt") by creating an Converged Application Server user known as *alice* using the commands described in Table 12–1.

1. Create a user using the `privateIdentity` command as follows:

   ```
   privateIdentity add privateId=alice
   ```

2. Create the public identity for alice by entering the SIP address:

   ```
   publicIdentity add publicId=sip:alice@test.company.com privateId=alice
   ```

3. Add an account for alice and use one of the optional commands described in Table 12–1 to set the status of the account. To create an active account for alice, enter the following:

   ```
   account add uid=alice active=true
   ```

4. Use the `role` command to add alice to the *Location Service* user group. Doing so grants alice permission to the Proxy Registrar's Location Service lookup:

   ```
   role user add uid=alice name="Location Service"
   ```

5. Add user authentication credentials for alice:

   ```
   credentials add uid=alice realm=test.company.com password=welcome1
   ```

   The `credentials` command is not needed for applications configured to use the RADIUS Login Module to authenticate users against RADIUS servers. For more information on these login modules, see *Oracle Communications Converged Application Server Security Configuration Guide*.

Note: You must also configure `realms` using the SIP Servlet Container MBean before you use Sash to add authorization credentials to a user.

Example 12–2 shows the Sash commands for creating a user.

Example 12–2 Creating a User from the Sash Command-Line Prompt

```
sash# privateIdentity add privateId=alice
sash# publicIdentity add publicId=sip:alice@test.company.com privateId=alice
sash# account add uid=alice active=true
sash# role user add uid=alice name="Location Service"
sash# credentials add uid=alice realm=test.company.com password=welcome1
```

Tip: You can create multiple users by creating Sash batch files. For more information, see "Scripting with Sash".

Creating a User with the Command Service MBean

You can execute Sash commands using the CommandService MBean's *execute* operation. The Command Service MBean is defined within the `subscrdataservcommandsear` application.

To create a user:

1. Select the *execute* operation. The *Operation* page for the *execute* operation appears.
2. Enter *privateIdentity add privateId=alice* in the *Value* field.
3. Click **Invoke Operation**. Repeat this process for each of the user creation commands. For example, the subsequent `publicIdentity` and `account` commands would both be followed by **Invoke Operation**.

Creating a User with the Identity Add Command

The `identity add` command enables you to create a user with one command string. This command, which is an alias to the `privateIdentity`, `publicIdentity`, `account`, `role` and `credentials` commands, enables you to quickly create a basic user account that contains the minimum information needed for users to connect to Converged Application Server through a SIP client. For example, to create a basic account for user *alice* using this command, enter the following from either the command line or through the Command Service MBean's *execute* operation:

```
identity add privateId=alice publicId=sip:sip.alice@company.com role="Location
Service" realm=company.com password=welcome1
```

Note: For applications configured to authenticate users against a RADIUS system (the applications with the RADIUS Login Module as the security provider), the command to create a user account is as follows:

```
identity add privateId=alice publicId=sip:sip.alice@company.com
role="Location Service"
```

The `identity add` command only enables you to create a basic user account. Accounts that require more complex construction, such as those that associate multiple `publicIds` with a single `privateId`, must be created using multiple Sash commands as illustrated in Example 12–2, "Creating a User from the Sash Command-Line Prompt".

Deleting a User

The `identity delete` command enables you to delete all of a user's roles, credentials, account information, public and private identities using a single command string. For example, to delete an account for a user **alice** using this command, enter the following from either the command line or through the Command Service MBean's `execute` operation:

```
identity delete privateId=alice
```

Note: The `identity delete` command indicates the delete operation is successful if any of the user's data is deleted, even if certain data, such as the user account, no longer exists due to being previously deleted.

Administering Registrations

Scripting with Sash

You can construct scripts for common tasks that contain several operations. Sash can be evoked to execute a file containing a list of commands. To enable scripting, Sash provides such command-line flags as:

- `-- exec` (short name: `-e`): When this command-line flag is followed by a command enclosed within quotation marks, Sash executes the command and then exits.
- `-- file` (short name: `-f`): When this command-line flag is followed by a filename, Sash reads the file and executes all commands in the file as they were entered and then exits.

  Example 12–3 illustrates a text file named **ocsm_users.txt**, which contains a group of users defined with the `identity add` command. You can provision these users by entering `-f OWLCS_users.txt` from the Sash prompt:

Example 12–3 Creating Users from a Text File (OWLCS_users.txt)

```
identity add privateId=candace publicId=sip:candace@doc.oracle.com role=user
password=1234 realm=doc.oracle.com
identity add privateId=deirdre publicId=sip:deirdre@doc.oracle.com role=user
password=1234 realm=doc.oracle.com
identity add privateId=evelyn publicId=sip:evelyn@doc.oracle.com role=user
password=1234 realm=doc.oracle.com
identity add privateId=frank publicId=sip:frank@doc.oracle.com role=user
password=1234 realm=doc.oracle.com
```

- `-- nonewline`: This command-line flag facilitates parsing output by stripping returns or newlines from the messages returned from the executed commands. Although this command facilitates parsing, it makes reading messages manually more difficult.

Error Logging in Sash

Sash does not log to any files (with the default configuration), it only prints messages on the console. The log level for Sash is configured in `ORCL_HOME/sash/conf/logging.properties`, where *`ORCL_HOME`* is the home directory where you installed the WebLogic Server portion of Converged Application Server (the default *`ORCL_HOME`* is `oracle/middleware/occas_5_0`).

13 Configuring the Converged Load Balancer

This chapter describes the Converged Load Balancer (CLB) provided with the Converged Application Server.

Overview of the Converged Load Balancer

The CLB is a SIP-aware load balancer that you can use to provide load distribution and failover services for your Converged Application Server deployment. The CLB load balances HTTP, HTTPS, SIP, and SIPS traffic.

Note: CLB is intended for SIP-only or converged (SIP and HTTP) application deployments. To load balance HTTP-only application servers, Oracle recommends that you use the WebLogic load balancer plug-in. For more information the plug-in, see the Oracle Fusion Middleware documentation set.

In contrast to traditional, HTTP-focused load balancers, the CLB takes into account the inherent differences between SIP and HTTP applications. For example, in a SIP environment, a single client request can multiple actual requests or responses to be generated or multiple applications to be invoked.

By distributing workload among multiple servers, the converged load balancer increases overall throughput of the system. It also increases the availability of your system by monitoring Converged Application Server health. If a target server becomes unavailable, the CLB redirects the flow of traffic to an available server in the cluster. When the unavailable server becomes available again, CLB resumes targeting the cluster.

Converged Load Balancers are stateless. Thus, Converged Load Balancer instances can easily be added to a deployment for redundancy or to increase traffic capacity.

You configure the Converged Load Balancer in the Admin Console. At startup time, the Converged Load Balancer retrieves its running configuration from the Admin Server. At first start-up, the Converged Load Balancer must be able to retrieve a running configuration before it can operate. At subsequent start-ups, it attempts to retrieve the configuration, but operates with a local copy of the last retrieved configuration if unsuccessful. Additionally, the Converged Load Balancer can poll the Admin Console specified configuration at configured intervals. This enables changes to be applied at runtime.

Configurable parameters for the Converged Load Balancer include its load balancing policy, health monitoring settings, network connections, and other settings. The network settings are divided between internal and external interfaces, isolating the two types of networks. The internal interface receives traffic only from the Converged Application Server, while an external listener receives traffic only from user agents, as described in the following section.

Deployment Topology

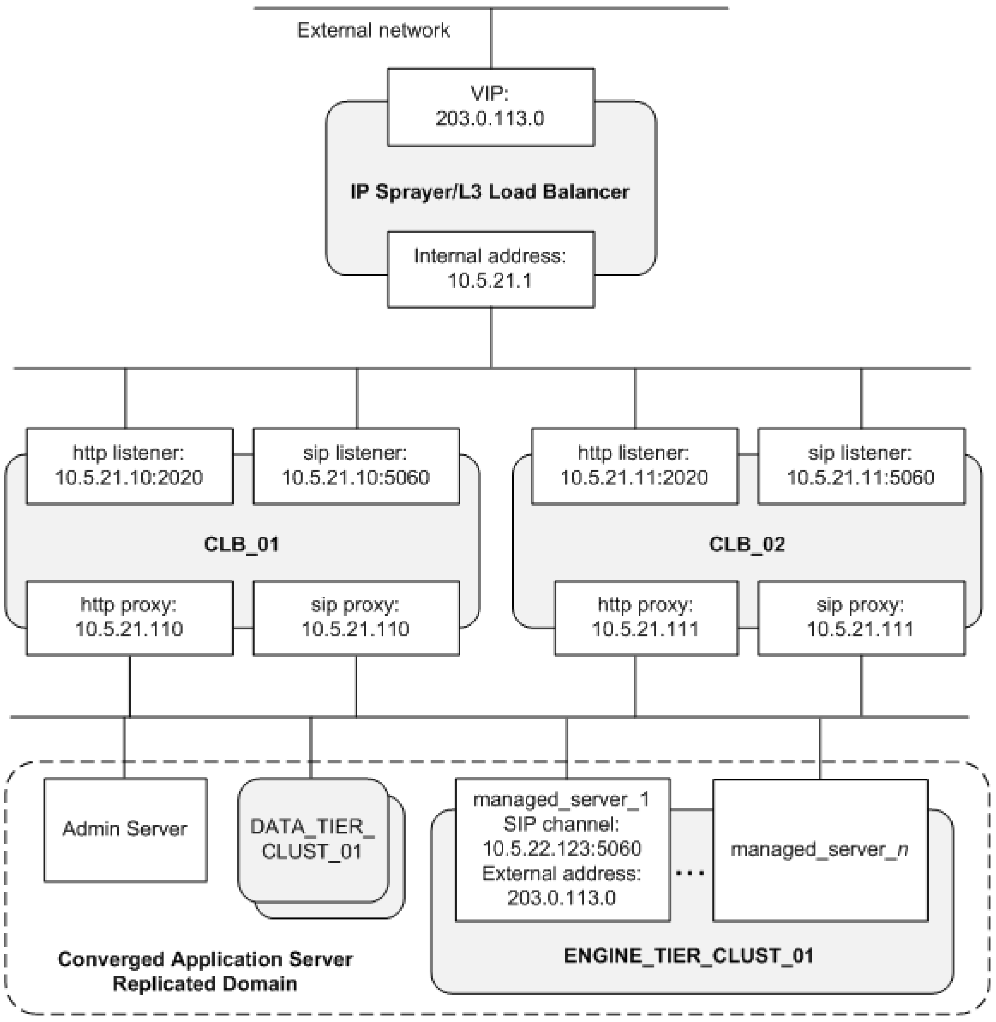
The Converged Load Balancer distributes traffic to clustered Converged Application Servers in a given WebLogic domain.

The Converged Load Balancer is intended to reside at the same location as the target cluster, either on the same machine or on a different machine in the same data center. It is not intended to be used with geographically distributed clusters.

The Converged Load Balancer does not perform IP address virtualization. Therefore, if external clients to address converged application services, an IP sprayer or other network router element must be present to expose a virtual IP (VIP) address for the services. However, if the Converged Load Balancer in invoked from within an internal network only and NATing is not required, you can use round-robin DNS or the IP address virtualization facility offered by the Linux operating system.

Figure 13–1 shows an example deployment scenario in which the CLB is deployed behind an IP sprayer.

Figure 13–1 Example Converged Load Balancer Deployment



As shown in Figure 13–1:

- An IP sprayer exchanges traffic with the external network at the public address 203.0.113.0. No special configuration is required for the IP sprayer. The source IP of the Converged Application Server instance handling a session is conveyed in internal headers managed by the Converged Application Server and Converged Load Balancer.
- Two Converged Load Balancers, labelled CLB_01 and CLB_02, distribute traffic to the managed servers in the engine tier cluster. For redundancy, Oracle recommends using at least two Converged Load Balancers.

  The Converged Load Balancers need to have network access to the Administration Server in order to retrieve its runtime configuration.
- The SIP containers are hosted on a single engine tier cluster, ENGINE_TIER_CLUST_01. Oracle recommends using only a single engine tier cluster for the engine tier managed servers in the deployment, since session failover is not supported across servers in different engine tier clusters.
- Each managed server includes a SIP channel configuration that specifies the external address used to address the hosted services. In the figure, the network channel configuration would specify the VIP presented by the IP sprayer, 203.0.113.0. The HTTP channel require no special or additional configuration.
- Local properties control configurable behavior of the Converged Load Balancer, such as its listen ports. In the figure, the Converged Load Balancers listen on port 2020 for HTTP messages and port 5060 for SIP messages.
- If the proxy address is specified, the source address values of the messages received by the managed servers from the Converged Load Balancers are the proxy address. For example, Figure 13–1 shows the address of 10.5.21.110 for the Converged Load Balancer instance CLB_01 proxy.

There are a few points to consider when planning your application deployments. For SIP applications, the engine tier servers must be mirrored environments. That is, all servers in the engine tier must have the same SIP applications.

For HTTP applications, the servers do not need to be mirrored. The CLB can direct traffic to a particular server based on the request URL. You specify specialized HTTP application routing rules using the context URL setting for the servers.

How the Converged Load Balancer Distributes Traffic

Converged Load Balancers direct messages to the managed servers in your deployment according to the rules you configure. The Converged Load Balancer can target clustered servers only. You would need to add a standalone server to a cluster to enable it to be targeted by the Converged Load Balancer.

In routing requests, the Converged Load Balancer handles HTTP requests differently from SIP requests. For HTTP requests, the CLB first attempts to match the request URL with the context roots of the applications for which it performs load balancing. The context root is the common URL root pattern that identifies the application resource in the request.

The CLB selects one server instance from the list of servers that match the context root based on the configured load balancing algorithm. A SIP request does not have a context root associated with it, so SIP requests bypass the pattern matching step. Consequently, while the servers in a cluster can be heterogeneous in terms of HTTP applications, they must be homogenous in terms of SIP applications.

For new requests (that is, those that are not associated with existing sessions), the Converged Load Balancer can distribute the requests based upon the following algorithms:

- Round-robin, in which the load balancer targets managed servers on a rotating basis.
- Consistent hash, in which the load balancer targets the managed server to service the request based on the value of a hash-key extracted from the request. The hash key is extracted based upon a key derived from the message, such as a header value, as described in the following section.

Using Data Centric Rules to Customize Load Distribution Selection

When using the consistent hash load distribution algorithm, the Converged Load Balancer makes a routing determination based on the value you specify as the policy key. You can configure different keys for each protocol, HTTP and SIP.

By default, the Converged Load Balancer determines the hash key used to make a server targeting decision for new SIP messages based on the value of the call ID header (call-id). For HTTP traffic, the determination is made based on the values of the remote host and remote port headers.

By implementing a DCR plug-in, you can specify complex key extraction logic.

The DCR plug-in extends the `DcrPlugin` interface, which defines two methods, one for HTTP and the other for SIP messages. The methods accept as input the message and return a value used as the consistent hash key.

To install the DCR plug-in, you add the `DcrPlugin` implementation to a Java archive and place it in the following directory:

domain_home/**config**

Next, specify the name of the JAR file that contains the `DcrPlugin` implementation in the **Dcr File Name** field in the Load Balancing Policy tab.

For more information on the `DcrPlugin` interface, see the Converged Application Server Javadoc.

About Session Affinity

The Converged Load Balancer supports session affinity (also called session stickiness). Session affinity ensures that the same server instance that handled an initial request receives subsequent messages associated with that session.

The Converged Load Balancer supports session affinity differently between HTTP/HTTPS and SIP/SIPS.

To support session affinity for HTTP and HTTPS sessions, the Converged Load Balancer uses cookies. If cookie support is disabled for the browser, it uses URL rewriting.

For SIP/SIPS session affinity, the Converged Load Balancer adds server instance information to the BEKey and BERoute parameters. For the BERoute parameter, it adds the information as a header to the outgoing request.

When a message associated with the existing session returns with the BEKey or BERoute header value, the Converged Load Balancer is able to target the server instance that previously services the session.

It includes the key in outgoing responses. The client should retain the key in subsequent messages to enable the Converged Load Balancer and Converged Application Server to identify messages associated with an existing session.

It is possible for a server associated with an existing session to become unavailable while the session is ongoing. If the Converged Load Balancer detects that a server instance associated with an existing session is unavailable, it selects a healthy instance from the same cluster and forwards the request to the selected instance.

In the event of server failover, session maintenance can only occur if the session fails over to another server in the same cluster. That is, a managed server in a cluster cannot take over a session for a server in another cluster.

If all managed servers in a cluster are unavailable, the message is sent to another cluster. However, since session information is lost, the new managed server can only respond with an error response. To avoid this scenario, Oracle recommends using a single engine tier cluster for all engine tier managed servers in the domain.

Configuration Overview

Configuring the Converged Load Balancer involves the following general steps:

1. Deploy and perform the initial configuration of the Converged Load Balancer software. See "Deploying the Converged Load Balancer" for more information.
2. Configure the load balancing policy for the Converged Load Balancer. See "Configuring the Load Balancing Policy" for more information.
3. Configure the network connections. See "Configuring Network Connections" for more information.
4. Identify the target managed server clusters for which the load balancer with distribute traffic. See "Configuring Target Clusters" for more information.
5. Configure any additional optional settings, such as health monitoring settings. See "Configuring Server Health Checks" for more information.
6. Configure the channel settings for the engine tier managed servers to accommodate the Converged Load Balancer instances.

   In the channel configuration, you should set the connection address from the engine tier servers to the internal, server-side interface of the Converged Load Balancer. See "Configuring Network Connection Settings" for more information.

The following sections provide more information about each of the configuration steps.

Deploying the Converged Load Balancer

The Converged Load Balancer software is included with the Converged Application Server installation. It is deployed with the domain files. After you create the domain, you can find the Converged Load Balancer files in the following directory:

domain_home/**clb**

After you create the domain, you need to specify local configuration settings for the Converged Load Balancer, such as the location of the Admin Server. You need to specify local settings for each load balancer instance in your deployment.

In contrast to local settings, common settings are specified in the Administration Server interface, and are shared by the load balancer instances in the deployment.

After creating the domain, configure the Converged Load Balancer local settings as follows:

1. Open the **clb.properties** file in the **clb** directory for editing.
2. In **clb.properties** file, configure the following properties:
   - **clb.server.name**: Set to a unique name for this load balancer instance.
   - **admin.server.address**: Set to the IP address of the Converged Application Server administration server for the domain.
   - **admin.server.port**: Specify the port number on which the Converged Application Server administration server listens for administrative traffic.
3. Save the file.

Additional load balancer configuration options are configured in the Administration Console interface. See "Configuring Network Connections" for more information.

Configuring the Load Balancing Policy

The Converged Load Balancer supports the consistent-hash load distribution algorithm for SIP traffic, and consistent-hash or round-robin policies for HTTP traffic. For consistent hash, you can specify the hash key using a Data Centric Rule (DCR) file. DCR offers flexible consistent hash key options. If you do not specify custom rules, the Converged Load Balancer uses the default rules specified in the configuration.

To use a DCR, before configuring the load balancer policy, create the file that contains the rules used to implement the consistent hash algorithm. See "Using Data Centric Rules to Customize Load Distribution Selection" for information about creating the DCR file.

You can configure different policies between SIP and HTTP traffic. To configure the load balancing policy:

1. In the Administration Console interface, click the **Converged Load Balancer** link in the Domain Structure tree.
2. In the Configuration tab, click the **LoadBalancingPolicy** subtab.
3. Configure the following settings:
   - **Http Policy**: The policy the Converged Load Balancer applies to HTTP message distribution.
   - **Sip Policy**: The policy the Converged Load Balancer applies to SIP message distribution.
   - **Dcr File Name**: The name and location of the file archive that contains your DcrPlugin implementation. Specify the location relative to the domain home, which is where the DCR file should be located.
   - **Default Http Keys**: If you do not specify a DCR file, the headers or request parameters specified in this field are used as the consistent hash key. By default, this is "remote-host,remote-port", which indicates the a concatenation of the hostname and port of the request source will be used to determine the target managed server instance to service this request.
   - **Default Sip Keys**: If you do not specify a DCR file, the headers or request parameters specified in this field are used as the consistent hash key. By default, this is "call-id", which is the identifier for the SIP call.
4. Click **Save** to commit your configuration changes.

Configuring Network Connections

After deploying the Converged Load Balancer, you can configure the connection settings for the load balancer. The settings define:

- Converged Load Balancer host settings, which identify the connection settings used for administration purposes.
- Server-side connections, or connections from the load balancer to the application server
- Client-side connections, or connections from the external network to the load balancer.

Include the listener and proxy settings (as shown in Figure 13–1) for the load balancer instance.

Configuring the Converged Load Balancer Host Settings

Configure the Converged Load Balancer host connection settings as follows:

1. In the Administration Console interface, click the **Converged Load Balancer** link in the Domain Structure tree.
2. Under the Configuration tab, click the **Servers** subtab.
3. Click the **New** button.
4. In the **Create the Load Balancer Server** page, configure these settings:
   - **LoadBalancer Server Identifier**: The Converged Load Balancer instance name. This value should match the value you set for the **LoadBalancer Server Identifier** field for this instance in the **clb.properties** file.
   - **Http Proxy Bind Address**: The address to which web traffic connections to backend managed servers are bound. In the example deployment shown in Figure 13–1, the bind address for the HTTP proxy is 10.5.21.110.
   - **Sip Proxy Bind Address**: The address to which SIP traffic connections to backend managed servers are bound. In the example deployment shown in Figure 13–1, the bind address for the SIP proxy is 10.5.21.110.
   - **Monitoring Address**: Enter the IP address or hostname of the Converged Load Balancer host by which the Admin Server monitors the status of the Converged Load Balancer.
   - **Monitoring Port**: Enter the port on which the Admin Server monitors the status of the Converged Load Balancer.
   - **Server Description**: An optional description of the CLB instance.
5. Click **Save**.

The new instance configuration appears in the list of Converged Load Balancer Servers. From this page, you can remove the Load Balancer Server instance or create additional ones.

Repeat the steps for each Converged Load Balancer instance you want to deploy.

Configuring Client-Side Network Settings

This section describes how to configure the client side network interfaces for the Converged Load Balancer. For converged application traffic, you would typically need at least two client-side network interface settings, one for HTTP and one for SIP. The interface settings encompass protocol settings as well as whether security is enabled for the interface.

To configure client-side network interface settings:

1. In the Administration Console interface, click the **Converged Load Balancer** link in the Domain Structure tree.
2. Under the Configuration tab, click the **Listeners** subtab.
3. Click the **New** button.
4. In the **Create a Listener** page, configure these settings:
   - **Listerner Id**: Enter an identifying name for this listener.
   - **Load Balancer Name**: Choose the load balancer instance associated with tis listener. The load balancer instances you have configured in the Servers tab, as described in "Configuring the Converged Load Balancer Host Settings", appear in the **Load Balancer Name** menu.
   - **Address**: Enter the IP address on which the load balancer listens for traffic from the client network.
   - **Protocol**: Choose the communication protocol for which the interface listens for traffic, from these options:
     - **sip** for SIP traffic.
     - **http** for web traffic.
   - **Secure**: Select to enable security for the interface. Selecting this check box activates the next two options on the page.
   - **CertNickName**: The alias of the server security certificate associated with the Converged Load Balancer host, and used by clients to authenticate the connection. You should import the certificate into the Converged Load Balancer keystore. The default load balancer keystore is named **keystore.jks**, and appears in the **config** directory under the load balancer home.
   - **ClientAuthEnabled**: Whether the load balancer should use two-way authentication to authenticate the client. If enabled, you should import the certificates of the trusted clients into the Java key store, **keystore.jks**, in the **config** directory under the load balancer home.
5. Click **Save**.

The client-side network connection settings appear in the Listener configuration list. From there, you can remove the listener definition or create additional listeners.

Configuring Server-Side Network Settings

The server side network settings control connections from the load balancer to the engine tier servers. You can configure connection parameters specific for HTTP or SIP connections made by the proxy module.

The proxy settings are preconfigured with default settings. In most cases, you will need to tune the settings based on the nature of your deployment.

To configure proxy settings:

1. In the Administration Console interface, click the **Converged Load Balancer** link in the Domain Structure tree.

2. Under the Configuration tab, do one of the following, depending on whether you are configure SIP or HTTP settings:
   - For HTTP proxy settings, click the **Http** and then **Http Proxy** subtabs
   - For SIP proxy settings, click the **Sip** and then **Sip Proxy** subtabs.
3. Click the **Http Proxy** subtab.
4. Configure the default configuration settings applicable to the traffic, include:
   - **Max Connections Per Backend**: The maximum number of connections allowed for each backend server. The default is 20.
   - **Keep Alive Timeout**: The time after which the Converged Load Balancer closes the socket for an idle connection. The default is 30.
   - **Send Retry Count**: The maximum number of times that the Converged Load Balancer will attempt to send a given request to a managed server. The default is 3.
   - **Send Buffer Size**: The size of the Send buffer used for all socket connections established to the managed server instances. The default is 4096.
   - **Receive Buffer Size**: The size of the Receive buffer used for all socket connections established to the managed server instances. The default is 4096.
5. Click the **Save** button.
6. Optionally, use the other settings in the ThreadPool, Network, Http Proxy and OverloadProtection subtabs to configure the interface. See the on-screen descriptions of each field for more information.

The proxy interface settings are now configured. You can now configure load balancer target settings and load balancing policies.

Configuring Target Clusters

The load balancer distributes messages to application servers based on your target configuration. The targets identify the engine tier clusters to which the load balancer should distribute messages.

To identify a target, you only identify the name of the engine tier cluster in the target configuration. You do not have to identify individual server instances that belong to the cluster. They are automatically targeted based on the cluster configuration.

While the load balancer distributes SIP and generic HTTP traffic to the servers as determined by the target cluster configuration, web application traffic can be targeted to a specific cluster based on the target URL in the incoming request. You use web modules to define distribution targets based on the context root of the request URL. This enables you to serve the web application on a separate cluster from other applications.

Before starting, make sure that you have configured the clusters for the Converged Application Server instances. You can check and modify the cluster configuration by clicking the **Environment** link and then **Cluster** in the Domain Structure tree. To identify the cluster to the load balancer, you will need to use the same name for the cluster as assigned in the Cluster Properties page.

After defining the cluster configuration, specify load balancer targets as follows:

1. In the Administration Console interface, click the **Converged Load Balancer** link in the Domain Structure tree.

2. Under the Configuration tab, click the **Targets** subtab.
3. Click the **New** button.
4. From the **Cluster Name** menu, choose the cluster that should be targeted by the load balancer instances. The cluster definitions you have created in the domain's environment settings appear in the menu list.
5. Click the **Enabled** check box to enable load distribution for this cluster.

   This check box provides a convenient mechanism for removing or adding a cluster as a load distribution target manually. For example, you might manually disable the cluster to perform maintenance tasks, such as software upgrades.
6. Click the **Save** button.

If needed, target a specific cluster for handling web application requests as follows:

1. Under the Configuration tab, click the **WebModules** subtab.
2. Click the **New** button.
3. Type a unique name for the web application distribution rule in the **WebModule Id** field.
4. For the **Target Name** menu, choose the cluster that should receive requests addressed to this application. The clusters you have defined in the **Target** tab appear in the menu.
5. In the **WebModulePolicy** menu, choose the distribution algorithm by which the load will be distributed among the servers in this cluster. Options are **round-robin**, **consistent-hash**, or **default**. If you choose **default**, the load balancer uses the distribution method specified as the general policy for HTTP traffic.
6. In the **ContextRoot** field, specify the URL path that the load balancer users to select traffic for this target cluster.

   For example, the context root **/sip-server** would apply to web applications available at: for web applications exposed at **/sip-server/main.jsp** or **/sip-server/TelcoApp/conference.jsp**.
7. Click the **Save** button.

The new web module configuration setting appears it the list. You remove the settings or create additional web module distribution rules from there.

Configuring Server Health Checks

The CLB can monitor the health of the backend server instances to which it distributes traffic.

You can configure one of several methods by which the health monitor checks the availability of the Converged Application Server instances, including:

- SIP OPTIONS ping, in which the CLB sends an OPTIONS message to the Converged Application Server instance. Any response (in this case, a SIP Response 404) indicates the availability of the server.
- JMX notification, in which the CLB registers as a notification listener to the `ServerNames` attribute of the `ClusterRuntimeMBean`. Change to the cluster view map generate a notification. Since this mechanism requires at least one instance in the cluster to be available, as a supplementary measure, the CLB periodically checks the server instance connections.

- OCCASClusterService notification is the default and recommended mechanism. The OCCASClusterService is a service that runs separately from the managed server instances. The service generates notifications in the form of HTTP requests to the CLB when the state of a cluster changes, for instance, when an instance in the cluster becomes unavailable.

The OCCASClusterService notification mechanism sends state notification messages to the address specified by the HTTP listener in the CLB configuration. Therefore, to use OCCASClusterService notification mechanism, you must configure an HTTP listener that the CLB will use to listen for the managed server instance's health state. See "Configuring the Converged Load Balancer Host Settings" for information about configuring the HTTP proxy settings.

To configure the health monitor mechanism, follow these steps:

1. In the Administration Console interface, click the **Converged Load Balancer** link in the Domain Structure tree.
2. If you are using OCCASClusterService notification mechanism, verify the listener configuration as follows:
   a. Under the Configuration tab, click the **Listeners** subtab.
   b. Verify the default listener configuration, or create a new listener configuration. The listener should listen for HTTP protocol traffic. See "Configuring Client-Side Network Settings" for more information on how to configure a listener.
3. Under the Configuration tab, click the **HealthMonitor** subtab.
4. From the Policy menu, choose the monitoring mechanism used by the health monitor.

   By default, this is set to ClusterNotify, the recommended option. Other options are PING, for SIP OPTIONS ping, or JMX, which uses the `CLusterRuntimeMBean`.
5. If using PING or JMX, verify the other settings for the health monitor. The default values for the other values should be appropriate for most implementations.
6. Click the **Save** button to commit your changes to the configuration.

The health monitor now performs server monitoring of all managed instances in the target clusters based on your settings.

Starting and Stopping the Converged Load Balancer

You start the Converged Load Balancer using the start script located in the CLB installation directory.

To start the load balancer:

1. Change directories to the following location under the Converged Application Server domain:

   domain_home/**clb/**
2. Run the start script appropriate for you operating system:
   - **startClb.cmd** for Microsoft Windows systems.
   - **startClb.sh** for Linux-based systems.

The Converged Load Balancer process starts up.

To stop the process, you can use the Control-C key sequence.

Tuning and Monitoring the Converged Load Balancer

The Admin Console presents extensive information on the operation of the CLB. It presents information by protocol, with information organized by SIP traffic activity and HTTP traffic activity.

To view monitoring information:

1. In the Administration Console interface, click the **Converged Load Balancer** link in the Domain Structure tree.
2. Click the **Monitoring** tab.
3. Click either:
    - **Http** subtab to view HTTP traffic statistics
    - **Sip** subtab to view SIP traffic statistics

You can tune the performance of the CLB by protocol. The settings include thread size, queue size, buffer size, and so on.

To configure performance settings:

1. In the Administration Console interface, click the **Converged Load Balancer** link in the Domain Structure tree.
2. Click the **Configuration** tab.
3. Click either:
    - **Http** subtab to tune HTTP traffic statistics.
    - **Sip** subtab to tune SIP traffic statistics.

14

Configuring Diameter Client Nodes and Relay Agents

This chapter describes how to configure individual servers to act as Diameter client nodes or relays in a Oracle Communications Converged Application Server domain:

Overview of Diameter Protocol Configuration

A typical Converged Application Server domain includes support for the Diameter base protocol and one or more IMS Diameter interface applications (Sh, Ro, Rf) deployed to engine tier servers that act as Diameter client nodes. SIP Servlets deployed on the engines can use the available Diameter applications to initiate requests for user profile data, accounting, and credit control, or to subscribe to and receive notification of profile data changes.

One or more server instances may be also be configured as Diameter relay agents, which route Diameter messages from the client nodes to a configured Home Subscriber Server (HSS) or other nodes in the network, but do not modify the messages. Oracle recommends configuring one or more servers to act as relay agents in a domain. The relays simplify the configuration of Diameter client nodes, and reduce the number of network connections to the HSS. Using at least two relays ensures that a route can be established to an HSS even if one relay agent fails.

Note: In order to support multiple HSSs, 3GPP defines the Dh interface to look up the correct HSS. Converged Application Server does not provide a Dh interface application, and can be configured only with a single HSS.

Note that relay agent servers do not function as either engine or SIP data tier instances—they should not host applications, store call state data, maintain SIP timers, or even use SIP protocol network resources (sip or sips network channels).

Converged Application Server also provides simulator applications for the Sh and Ro protocols. You can use the simulator applications for testing while developing Sh and Ro clients. The simulator applications are not intended for deployment to a production system.

About the Diameter Domain Template

Converged Application Server includes a Diameter domain template that creates a domain having four Converged Application Server instances:

- An Administration Server (AdminServer)
- A Diameter Sh client node (hssclient)
- A Diameter relay node (relay)
- An HSS simulator (hss)

You can use the Diameter domain template as the basis for creating your own Diameter domain. Or, you can use the customized Diameter Web Applications as templates for configuring existing Converged Application Server instances to function as HSS client or relay agent nodes. The configuration instructions in the sections that follow assume that you have access to the Diameter domain configuration.

Table 14–1 describes the server configuration installed with the Diameter domain.

Table 14–1 Key Configuration Elements of the Diameter Domain

Server Name	Network Channel Configuration	Diameter Applications	Notes
`AdminServer`	n/a	n/a	The Administration Server provides no SIP or Diameter protocol functionality.
`hssclient`	diameter (TCP over port 3868) sip (UDP/TCP over port 5060)	`WlssShApplication`	The `hssclient` engine functions as a Diameter Sh client node. The server contains network channels supporting both SIP and Diameter traffic. The Diameter node configuration deploys `WlssShApplication` (`com.bea.wcp.diameter.sh.WlssShApplication`) to provide IMS Sh interface functionality for deployed SIP Servlets.
`relay`	diameter (TCP over port 3869)	`RelayApplication`	The `relay` engine functions as a Diameter Sh relay node. The server contains a network channel to support both Diameter traffic. The server does not contain a channel to support SIP traffic, as a relay performs no SIP message processing. The Diameter node configuration deploys `RelayApplication` (`com.bea.wcp.diameter.relay.RelayApplication`) to provide relay services. The node configuration also defines a realm-based route for relaying messages from the `hssclient` engine.
`hss`	diameter (TCP over port 3870)	`HssSimulator`	The `hss` engine's Diameter node configuration deploys only the `HssSimulator` application (`com.bea.wcp.diameter.sh.HssSimulator`). The server is configured with a Diameter network channel.

Steps for Configuring Diameter Client Nodes and Relay Agents

To configure Diameter support in a Converged Application Server domain, follow these steps:

1. Install the Converged Application Server Diameter Domain. The Diameter domain contains a sample configuration and template applications configured for different Diameter node types. You may use the Diameter domain as a template for your own domain, or to better understand how to configure different Diameter node types.
2. Enable the Diameter console extension. If you are working with the sample Diameter domain, the Diameter console extension is already enabled. If you are starting with a basic Converged Application Server domain, edit the **config.xml** file to enable the extension.
3. Create Diameter network channels. Create the network channels necessary to support Diameter over TCP, TLS, or SCTP transports on engine tier servers and relays.
4. Create and configure the Diameter nodes. Configure the Diameter protocol client applications on engine tier servers with the host name, peers, and routes to relay agents or other network elements, such as an HSS. You can also configure Diameter nodes that operate in standalone mode, without a Converged Application Server instance.

The sections that follow describe each step in detail. See also the "Example Domain Configuration".

Installing the Diameter Domain Template

You install and configure the Diameter domain using the JAR file (`diameterdomain.jar`) located at: *WL_home*/**common/templates/domains**

See the *Converged Application Server Installation Guide* for information on installing the Diameter domain template using the Converged Application Server Configuration Wizard.

Creating TCP, TLS, and SCTP Network Channels for the Diameter Protocol

The Converged Application Server Diameter implementation supports the Diameter protocol over the TCP, TLS, and SCTP transport protocols. (SCTP transport is provided with certain restrictions as described in "Configuring and Using SCTP for Diameter Messaging".)

To enable incoming Diameter connections on a server, you must configure a dedicated network channel of the appropriate protocol type:

- "diameter" channels use TCP transport
- "diameters" channels use TCP/TLS transport
- "diameter-sctp" channels use TCP/SCTP transport.

Servers that use a TCP/TLS channel for Diameter (diameters channels) must also enable two-way SSL. Converged Application Server may automatically upgrade Diameter TCP connections to use TLS as described in the Diameter specification (RFC 3558).

To configure a TCP or TCP/TLS channel for use with the Diameter provider, follow these steps:

1. Access the Administration Console for the Converged Application Server domain.
2. Click Lock & Edit to obtain a configuration lock.

   (If you are using a development domain, Lock & Edit is only present if you enable configuration locking. See "Enable and disable the domain configuration lock" in the Administration Console Online Help for more information.)
3. In the left pane, select the name of the server to configure.
4. In the right pane, select Protocols, and then select Channels to display the configured channels.
5. Click New to configure a new channel.
6. Fill in the fields of the Identity Properties page as follows:
   - **Name:** Enter an administrative name for this channel, such as "Diameter TCP/TLS Channel."
   - **Protocol:** Select "diameter" to support the TCP transport, "diameters" to support both TCP and TLS transports, or "diameter-sctp" to support TCP transport.

     Note: If a server configures at least one TLS channel, the server operates in TLS mode and will reject peer connections from nodes that do not support TLS (as indicated in their capabilities exchange).
7. Click Next to continue.
8. Fill in the fields of the Network Channel Addressing page as follows:
   - **Listen Address:** Enter the IP address or DNS name for this channel. On a multi-homed machine, enter the exact IP address of the interface you want to configure, or a DNS name that maps to the exact IP address.
   - **Listen Port:** Enter the port number used to communication via this channel. Diameter nodes conventionally use port 3868 for incoming connections.
   - **External Listen Port**: Re-enter the Listen Port value.
9. Click Next to continue.
10. Chose attributes in the Network Channel Properties page as follows:
    - **Enabled**: Select this attribute to ensure that the new channel accepts network traffic.
    - **Tunneling Enabled**: Un-check this attribute for Diameter channels.
    - **HTTP Enabled for this Protocol**: Un-check this attribute for Diameter channels.
    - **Outbound Enabled**: Select this attribute to ensure that the node can initiate Diameter messages using the channel.
11. Click Next to continue.
12. For "diameters" channels, select the following two attributes:
    - **Two Way SSL Enabled**: Two-way SSL is required for TLS transport.

- **Client Certificate Enforced**: Select this attribute to honor available client certificates for secure communication.

13. Click Finish to create the new channel.
14. Select the name of the newly-created channel in the Network Channel table.
15. Display the advanced configuration items for the newly-created channel by clicking the Advanced link.
16. Change the **Idle Connection Timeout** value from the default (65 seconds) to a larger value that will ensure the Diameter connection remains consistently available.

> **Note:** If you do not change the default value, the Diameter connection will be dropped and recreated every 65 seconds with idle traffic.

17. Click Save.
18. Click Activate Changes.

The servers installed with the Diameter domain template include network channel configurations for Diameter over TCP transport. Note that the relays server includes only a diameter channel and *not* a sip or sips channel. Relay agents should not host SIP Servlets or other applications, therefore no SIP transports should be configured on relay server nodes.

Configuring Two-Way SSL for Diameter TLS Channels

Diameter channels that use TLS (diameters channels) require that you also enable two-way SSL, which is disabled by default.

Follow these steps to enable two-way SSL for a server. If you have not already configured SSL, see the information on Configuring SSL in *Administrator's Guide* in the Oracle WebLogic Server 11*g* documentation for instructions.

Configuring and Using SCTP for Diameter Messaging

SCTP is a reliable, message-based transport protocol that is designed for use in telephony networks. SCTP provides several benefits over TCP:

- SCTP preserves the internal structure of messages when transmitting data to an endpoint, whereas TCP transmits raw bytes that must be received in order.
- SCTP supports multihoming, where each endpoint may have multiple IP addresses. The SCTP protocol can transparently failover to another IP address should a connection fail.
- SCTP provides multistreaming capabilities, where multiple streams in a connection transmit data independently of one another.

Converged Application Server supports SCTP for Diameter network traffic, with several limitations:

- Only 1 stream per connection is currently supported.
- SCTP can be used only for Diameter network traffic; SIP traffic cannot use a configured SCTP channel.
- TLS is not supported over SCTP.

In addition, Converged Application Server only supports SCTP channels on the following software platforms:

- Red Hat Enterprise Linux 4.0 AS, ES, WS (Kernel 2.6.9, GCC 3.4 or higher) on 32- or 64-bit hardware
- Sun Solaris 10 on SPARC

SCTP channels can operate on either IPv4 or IPv6 networks. "Creating TCP, TLS, and SCTP Network Channels for the Diameter Protocol" describes how to create a new SCTP channel. To enable multihoming capabilities for an existing SCTP channel, specify the IPv4 address `0.0.0.0` as the listen address for the channel (or use the `::` address for IPv6 networks).

Configuring Diameter Nodes

The Diameter node configuration for Converged Application Server engines is stored in the **diameter.xml** configuration file, which is located in the directory: *MW_home***/user_projects/domains/***domain_name***/bin/**

Where *MW_home* is the directory in which the Converged Application Server software is installed (the installation program used to install Converged Application Server refers to this as Middleware Home), and *domain_name* is the name of the Diameter domain. For example:

```
/Oracle/Middleware/user_projects/domains/Diameter_domain/config/custom
```

If you want to provide diameter services (client, server, or relay functionality) on an engine tier server, you must create a new node configuration and target the configuration to an existing engine server instance.

Diameter node configurations are divided into several categories:

- General configuration defines the host identity and realm for the node, as well as basic connection information and default routing behavior.
- Application configuration defines the Diameter application(s) that run on the node, as well as any optional configuration parameters passed to those applications.
- Peer configuration defines the other Diameter nodes with which this node operates.
- Routes configuration defines realm-based routes that the node can use when resolving messages.

The sections that follow describe how to configure each aspect of a Diameter node.

Creating a New Node Configuration (General Node Configuration)

Follow these steps to create a new Diameter node configuration and target it to an existing Converged Application Server engine tier instance:

1. Log in to the Administration Console for the Converged Application Server domain you want to configure.
2. Click Lock & Edit to obtain a configuration lock.

   (If you are using a development domain, Lock & Edit is only present if you enable configuration locking. See "Enable and disable the domain configuration lock" in the Administration Console Online Help for more information.)
3. Select the Diameter node in the left pane of the Console.

4. Click New in the right pane to create a new Diameter configuration.
5. Fill in the fields of the Create a New Configuration page as described in Table 14–2, then click Finish.

Table 14–2 Diameter Node General Configuration Properties

Property Name	Description
Name	Enter the administrative name for this Diameter node configuration.
Host	Enter the host identity of this Diameter node, or leave the field blank to automatically assign the host name of the target engine tier server as the Diameter node's host identity. Note that the host identity may or may not match the DNS name. When configuring Diameter support for multiple Sh client nodes, it is best to omit the `host` element from the **diameter.xml** file. This enables you to deploy the same Diameter Web Application to all servers in the engine tier cluster, and the host name is dynamically obtained for each server instance.
Realm	Enter the realm name for which this node has responsibility, or leave the field blank to use the domain name portion of the target engine tier server's fully-qualified host name (for example, host@oracle.com). You can run multiple Diameter nodes on a single host using different realms and listen port numbers. **Note:** An HSS, Application Server, and relay agents must all agree on a realm name or names. The realm name for the HSS and Application Server need not match.
Address	Enter the listen address for this Diameter node, using either the DNS name or IP address, or leave the field blank to use the host identity as the listen address. **Note:** The host identity may or may not match the DNS name of the Diameter node. Oracle recommends configuring the Address property with an explicit DNS name or IP address to avoid configuration errors.
TLS	Select this option if the Diameter node us configured with support for TLS (diameters network channels). This field is used to advertise TLS capabilities when the node is interrogated by another Diameter node.
Debug	Select this option if you want to enable debug message output. Debug messages are disabled by default.
Message Debug	Select this option if you want to enable tracing for Diameter messages processed by this node. Message tracing is disabled by default.
Dynamic Peers Allowed	Select this option to allow dynamic discovery of Diameter peer nodes. Dynamic peer support is disabled by default. Oracle recommends enabling dynamic peers only when using the TLS transport, because no access control mechanism is available to restrict hosts from becoming peers.
Peer Retry Delay	Enter the amount of time, in seconds, this node waits before retrying a request to a Diameter peer. The default value is 30 seconds.
Request Timeout	Enter the amount of time, in milliseconds, this node waits for an answer message before timing out.
Watchdog Timeout	Enter the number of seconds this node uses for the value of the Diameter Tw watchdog timer interval.
Targets	Enter one or more target engine tier server names. The Diameter node configuration only applies to servers listed in this field.

Table 14–2 (Cont.) Diameter Node General Configuration Properties

Property Name	Description
Default Route Action	Specify an action type that describes the role of this Diameter node when using a default route. The value of this element can be one of the following: ■ none ■ local ■ relay ■ proxy ■ redirect
Default Route Servers	Specifies one or more target servers for the default route. Any server you include in this element must also be defined as a peer to this Diameter node, or dynamic peer support must be enabled.

6. Click Activate Changes to apply the configuration to target servers.

After creating a general node configuration, the configuration name appears in the list of Diameter nodes. You can select the node to configure Diameter applications, peers, and routes, as described in the sections that follow.

Configuring Diameter Applications

Each Diameter node can deploy one or more applications. You configure Diameter applications in the Administration Console using the Configuration > Applications page for a selected Diameter node. Follow these steps:

1. Log in to the Administration Console for the Converged Application Server domain you want to configure.
2. Click Lock & Edit to obtain a configuration lock.

   (If you are using a development domain, Lock & Edit is only present if you enable configuration locking. See "Enable and disable the domain configuration lock" in the Administration Console Online Help for more information.)
3. Select the Diameter node in the left pane of the Console.
4. Select the name of a Diameter node configuration in the right pane of the Console.
5. Select the Configuration > Applications tab.
6. Click New to configure a new Diameter application, or select an existing application configuration from the table.
7. Fill in the application properties as follows:
   - **Application Name**: Enter a name for the application configuration.
   - **Class Name**: Enter the classname of the application to deploy on this node.
   - **Parameters**: Enter optional parameters to pass to the application upon startup.
8. Click Finish to create the new application configuration.
9. Click Activate Changes to apply the configuration to the Diameter node.

Converged Application Server includes several Diameter applications to support clients using the Sh, Rf, and Ro interfaces, Diameter relays, and simulators for the Sh and Ro interfaces. The sections that follow provide more information about configuring these Converged Application Server Diameter applications.

You can also use the base Diameter API included in Converged Application Server to create and deploy your own Diameter applications. See "Using the Diameter Base Protocol API" in *Converged Application Server Diameter Application Development Guide* for more information.

Configuring the Sh Client Application

The Sh client application is implemented as a provider to the Profile Service API. The application transparently generates and responds to the Diameter command codes defined in the Sh application specification. The Profile Service API enables SIP Servlets to manage user profile data as an XML document using XML Document Object Model (DOM). Subscriptions and notifications for changed profile data are managed by implementing a profile listener interface in a SIP Servlet.

See "Using the Diameter Sh Interface Application" in *Converged Application Server Diameter Application Development Guide* for more information about the API.

The Diameter nodes on which you deploy the Sh client application should be configured with:

- The host names of any relay agents configured in the domain, defined as Diameter peer nodes. If no relay agents are used, all engine tier servers must be added to the list of peers, or dynamic peers must be enabled.
- One or more routes to access relay agent nodes (or the HSS) in the domain.

To configure the Sh client application, you specify the `com.bea.wcp.diameter.sh.WlssShApplication` class. `WlssShApplication` accepts the following parameters:

- `destination.host` configures a static route to the specified host. Include a `destination.host` param definition only if servers communicate directly to an HSS (static routing), without using a relay agent. Omit the `destination.host` param completely when routing through relay agents.
- `destination.realm` configures a static route to the specified realm. Specify the realm name of relay agent servers or the HSS, depending on whether or not the domain uses relay agents.

Example 14–1 shows a sample node configuration for an Sh client node that uses a relay.

Example 14–1 Sample Diameter Node Configuration with Sh Client Application

```
<?xml version='1.0' encoding='utf-8'?>
<diameter xmlns="http://www.bea.com/ns/wlcp/diameter/300"
xmlns:sec="http://www.bea.com/ns/weblogic/90/security"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:wls="http://www.bea.com/ns/weblogic/90/security/wls">
  <configuration>
    <name>hssclient</name>
    <target>hssclient</target>
    <host>hssclient</host>
    <realm>oracle.com</realm>
    <!-- Omit the host and realm elements to dynamically assign the host name
     and domain name of individual engine tier servers. - >
    <message-debug-enabled>true</message-debug-enabled>
    <application>
      <name>WlssShApplication</name>
      <class-name>com.bea.wcp.diameter.sh.WlssShApplication</class-name>
      <param>
      <!-- Include a destination.host param definition only if servers will
```

```
          communicate directly to an HSS (static routing), without using
          a relay agent. Omit the destination.host param completely when
          routing through relay agents. - >
      <!-- Specify the realm name of relay agent servers or the HSS,
          depending on whether or not the domain uses relay agents. - >
        <name>destination.realm</name>
        <value>hss.com</value>
      </param>
    </application>
    <peer>
    <!-- Include peer entries for each relay agent server used in the domain.
         If no relay agents are used, include a peer entry for the HSS
         itself, as well as for all other Sh client nodes (all other engine
         tier servers in the domain).
         Alternately, use the allow-dynamic-peers functionality in
         combination with TLS transport to allow peers to be recognized
         automatically. - >
      <host>relay</host>
      <address>localhost</address>
      <!-- The address element can specify either a DNS name or IP address,
          whereas the host element must specify a diameter host identity.
          The diameter host identity may or may not match the DNS name. - >
      <port>3869</port>
    </peer>
    <!-- Enter a default route to a selected relay agent. If the domain does
         not use a relay agent, specify a default route to relay messages
         directly to the HSS. - >
    <default-route>
      <action>relay</action>
      <server>relay</server>
    </default-route>
  </configuration>
</diameter>
```

Configuring the Rf Client Application

The Converged Application Server Rf client application enables SIP Servlets to issue offline charging messages using the IMS Rf interface. To configure the Rf application, specify the class `com.bea.wcp.diameter.charging.RfApplication`. The Rf application accepts the following parameters:

- `cdf.host` specifies the host name of the Charging Data Function (CDF).
- `cdf.realm` specifies the realm of the CDF.

See "Using the Diameter Rf Interface Application for Offline Charging" in *Converged Application Server Diameter Application Development Guide* for more information about using the Rf application API in deployed applications.

Configuring the Ro Client Application

The Converged Application Server Ro client application enables SIP Servlets to issue online charging messages using the IMS Ro interface. To configure the Rf application, specify the class `com.bea.wcp.diameter.charging.RoApplication.` The Ro application accepts the following parameters:

- `ocs.host` specifies the host identity of the Online Charging Function (OCF). The OCF you specify host must also be configured as the peer for the Diameter node on which the Ro application is deployed.

- `ocs.realm` can be used instead of `ocs.host` for realm-based routing when using more than one OCF host. The corresponding realm definition must also exist in the Diameter node's configuration.

See "Using the Diameter Ro Interface Application for Online Charging" in *Converged Application Server Diameter Application Development Guide* for more information about using the Ro application API in deployed applications.

Configuring a Diameter Relay Agent

Relay agents are not required in a Diameter configuration, but Oracle recommends using at least two relay agent servers to limit the number of direct connections to the HSS, and to provide multiple routes to the HSS in the event of a failure.

Note: You must ensure that relay servers *do not* also act as Converged Application Server engine tier servers or SIP data tier servers. This means that the servers should not be configured with "sip" or "sips" network channels.

Relay agent nodes route Sh messages between client nodes and the HSS, but they do not modify the messages except as defined in the Diameter Sh specification. Relays always route responses from the HSS back the client node that initiated the message, or the message the response is dropped if that node is unavailable.

To configure a Diameter relay agent, simply configure the node to deploy an application with the class `com.bea.wcp.diameter.relay.RelayApplication`.

The node on which you deploy the relay application should also configure:

- All other nodes as peers to the relay node.
- A default route that specifies the relay action.

Example 14–2 shows the sample **diameter.xml** configuration for a relay agent node.

Example 14–2 Diameter Relay Node Configuration

```
<?xml version='1.0' encoding='utf-8'?>
<diameter xmlns="http://www.bea.com/ns/wlcp/diameter/300"
xmlns:sec="http://www.bea.com/ns/weblogic/90/security"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:wls="http://www.bea.com/ns/weblogic/90/security/wls">
  <configuration>
<name>relay</name>
    <target>relay</target>
    <host>relay</host>
    <realm>oracle.com</realm>
    <message-debug-enabled>true</message-debug-enabled>
    <application>
      <name>RelayApplication</name>
      <class-name>com.bea.wcp.diameter.relay.RelayApplication</class-name>
    </application>
    <!-- Define peer connection information for each Diameter node, or use
         the allow-dynamic-peers functionality in combination with TLS
         transport to allow peers to be recognized automatically. - >
    <peer>
      <host>hssclient</host>
      <address>localhost</address>
      <port>3868</port>
    </peer>
    <peer>
      <host>hss</host>
      <address>localhost</address>
      <port>3870</port>
    </peer>
    <route>
      <realm>oracle.com</realm>
      <application-id>16777217</application-id>
      <action>relay</action>
      <server>hssclient</server>
    </route>
    <!-- Enter a default route for this agent to relay messages
         to the HSS. - >
    <default-route>
      <action>relay</action>
      <server>hss</server>
    </default-route>
  </configuration>
</diameter>
```

Configuring the Sh and Rf Simulator Applications

Converged Application Server contains two simulator applications that you can use in development or testing environments to evaluate Diameter client applications. To configure a simulator application, you simply deploy the corresponding class to a configured Diameter node:

- `com.bea.wcp.diameter.sh.HssSimulator` simulates an HSS in your domain for testing Sh client applications.
- `com.bea.wcp.diameter.rf.RfSimulator` simulates an CDF host for testing Rf client applications

Note: These simulators are provided for testing or development purposes only, and is not meant as a substitute for a production HSS or CDF.

Diameter nodes that deploy simulator applications can be targeted to running engine tier servers, or they may be started as standalone Diameter nodes. When started in standalone mode, simulator applications accept the command-line options described in Table 14–3. See "Working with Diameter Nodes" in *Converged Application Server Diameter Application Development Guide* for more information.

Table 14–3 Command-Line Options for Simulator Applications

Option	Description
`-r, -realm` *`realm_name`*	Specifies the realm name of the Diameter node.
`-h, -host host_name`	Specifies the host identity of the node.
`-a, -address address`	Specifies the listen address for this node.
`-p, -port port_number`	Specifies the listen port number for this node.
`-d, -debug`	Enables debug output.
`-m, -mdebug`	Enables Diameter message tracing.

Enabling Profile Service (Using an Sh Backend)

As noted earlier, Sh, Ro, and Rf applications can be configured and used separately, but Sh can take advantage of the Profile Service API. To do so:

1. Configure ShApplication in **diameter.xml** (see Example 14–4, "diameter.xml Configuration for Sample Engine Tier Cluster (Sh Clients)" for more information).
2. Add a **profile.xml** file to **DOMAIN_HOME/config/custom/profile.xml**. You can either install the Diameter domain as a template and modify the file, or you can manually create **profile.xml** as shown in Example 14–3.

Example 14–3 profile.xml sample

```
<profile-service xmlns="http://www.bea.com/ns/wlcp/wlss/profile/300">
  <mapping>
    <map-by>prefix</map-by>
    <map-by-prefix>
      <provider-prefix-set>
        <name>sh</name>
        <prefix>sh</prefix>
      </provider-prefix-set>
    </map-by-prefix>
  </mapping>
  <provider>
    <name>sh</name>
    <provider-class>com.bea.wcp.profile.ShProviderCached</provider-class>
  </provider>
</profile-service>
```

Configuring Peer Nodes

A Diameter node should define peer connection information for each other Diameter node in the realm, or enable dynamic peers in combination with TLS transport to allow peers to be recognized automatically. You configure Diameter peer nodes in the Administration Console using the Configuration > Peers page for a selected Diameter node. Follow these steps:

1. Log in to the Administration Console for the Converged Application Server domain you want to configure.
2. Click Lock & Edit to obtain a configuration lock.

   (If you are using a development domain, Lock & Edit is only present if you enable configuration locking. See "Enable and disable the domain configuration lock" in the Administration Console Online Help for more information.)
3. Select the Diameter node in the left pane of the Console.
4. Select the name of a Diameter node configuration in the right pane of the Console.
5. Select the Configuration > Peers tab.
6. Click New to define a new peer entry.
7. Fill in the fields of the Create a New Peer page as follows:
   - **Host**: Enter the peer node's host identity.
   - **Address**: Enter the peer node's address (DNS name or IP address).
   - **Port Number**: Enter the listen port number of the peer node.
   - **Protocol**: Select the protocol used to communicate with the peer (TCP or SCTP).

Note: Converged Application Server attempts to connect to the peer using *only* the protocol you specify (TCP or SCTP). The other protocol is not used, even if a connection fails using the selected protocol. TCP is used as by default if you do not specify a protocol.

- **Watchdog**: Indicate whether the peer supports the Diameter Tw watchdog timer interval.

8. Click Finish to create the new peer entry.
9. Click Activate Changes to apply the configuration.

Configuring Routes

Certain Diameter nodes, such as relays, should configure realm-based routes for use when resolving Diameter messages. You configure Diameter routes in the Administration Console using the Configuration > Routes page for a selected Diameter node. Follow these steps:

1. Log in to the Administration Console for the Converged Application Server domain you want to configure.
2. Click Lock & Edit to obtain a configuration lock.

   (If you are using a development domain, Lock & Edit is only present if you enable configuration locking. See "Enable and disable the domain configuration lock" in the Administration Console Online Help for more information.)
3. Select the Diameter node in the left pane of the Console.
4. Select the name of a Diameter node configuration in the right pane of the Console.
5. Select **Configuration**, then select the **Routes** tab.
6. Click New to configure a new Route.
7. Fill in the fields of the Create a New Route page as follows:
   - **Name**: Enter an administrative name for the route.
   - **Realm**: Enter the target realm for this route.
   - **Application ID**: Enter the target Diameter application ID for this route.
   - **Action**: Select an action that this node performs when using the configured route. The action type may be one of: none, local, relay, proxy, or redirect.
   - **Server Names**: Enter the names of target servers that will use the route.
8. Click Finish to create the new route entry.
9. Click Activate Changes to apply the configuration.

See Example 14–2 for an example **diameter.xml** node configuration containing a route entry.

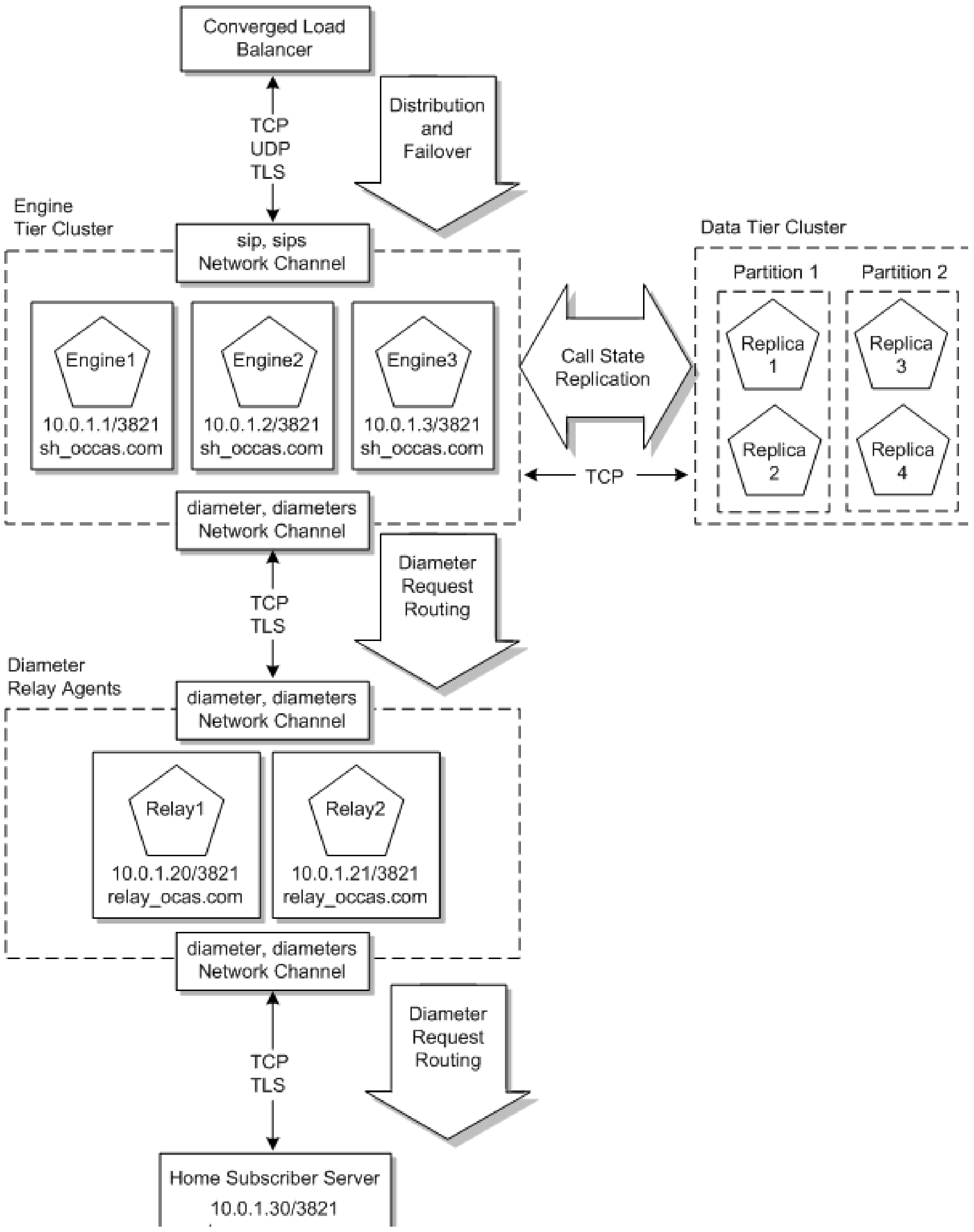
Example Domain Configuration

This section describes a sample Converged Application Server configuration that provides basic Diameter Sh protocol capabilities. The layout of the sample domain includes the following:

- Three engine tier servers which host SIP applications and also deploy the Diameter Sh application for accessing user profiles.
- Four SIP data tier servers arranged into two partitions with two replicas each.
- Two servers that act as Diameter relay agents and forward diameter requests to an HSS.

Figure 14–1 shows the individual servers in the sample configuration.

Figure 14–1 Sample Diameter Domain



Example 14–4 shows the contents of the **diameter.xml** file used to configure engine tier servers (Sh Clients) in the sample domain. Example 14–5 shows the **diameter.xml** file used to configure the relay agents.

Example 14–4 diameter.xml Configuration for Sample Engine Tier Cluster (Sh Clients)

```
<?xml version='1.0' encoding='utf-8'?>
<diameter xmlns="http://www.bea.com/ns/wlcp/diameter/300"
xmlns:sec="http://www.bea.com/ns/weblogic/90/security"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:wls="http://www.bea.com/ns/weblogic/90/security/wls">
<configuration>
    <name>clientnodes</name>
    <target>Engine1</target>
    <target>Engine2</target>
    <target>Engine3</target>
    <realm>sh_occas.com</realm>
    <application>
      <name>WlssShApplication</name>
      <class-name>com.bea.wcp.diameter.sh.WlssShApplication</class-name>
      <param>
        <name>destination.realm</name>
        <value>relay_occas.com</value>
      </param>
    </application>
    <peer>
      <host>Relay1</host>
      <address>10.0.1.20</address>
      <port>3821</port>
    </peer>
    <peer>
      <host>Relay2</host>
      <address>10.0.1.21</address>
      <port>3821</port>
    </peer>
    <default-route>
      <action>relay</action>
      <server>Relay1</server>
    </default-route>
    <route>
      <action>relay</action>
      <server>Relay2</server>
    </route>
  </configuration>
</diameter>
```

Example 14–5 diameter.xml Configuration for Sample Relay Agents

```
<?xml version='1.0' encoding='utf-8'?>
<diameter xmlns="http://www.bea.com/ns/wlcp/diameter/300"
xmlns:sec="http://www.bea.com/ns/weblogic/90/security"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:wls="http://www.bea.com/ns/weblogic/90/security/wls">
  <configuration>
    <name>relaynodes</name>
    <target>Relay1</target>
    <target>Relay2</target>
    <realm>relay_occas.com</realm>
    <application>
      <name>RelayApplication</name>
      <class-name>com.bea.wcp.diameter.relay.RelayApplication</class-name>
    </application>
    <peer>
      <host>Engine1</host>
      <address>10.0.1.1</address>
      <port>3821</port>
    </peer>
    <peer>
      <host>Engine2</host>
      <address>10.0.1.2</address>
      <port>3821</port>
    </peer>
    <peer>
      <host>Engine3</host>
      <address>10.0.1.3</address>
      <port>3821</port>
    </peer>
    <peer>
      <host>Relay1</host>
      <address>10.0.1.20</address>
      <port>3821</port>
    </peer>
    <peer>
      <host>Relay2</host>
      <address>10.0.1.21</address>
      <port>3821</port>
    </peer>
    <peer>
      <host>hss</host>
      <address>hssserver</address>
      <port>3870</port>
    </peer>
    <default-route>
      <action>relay</action>
      <server>hss</server>
    </default-route>
  </configuration>
</diameter>
```

Troubleshooting Diameter Configurations

SIP Servlets deployed on Converged Application Server use the available Diameter applications to initiate requests for user profile data, accounting, and credit control, or to subscribe to and receive notification of profile data changes. If a SIP Servlet performing these requests generates an error similar to:

```
Failed to dispatch Sip message to servlet ServletName
java.lang.IllegalArgumentException: No registered provider for protocol: Protocol
```

The message may indicate that you have not properly configured the associated Diameter application for the protocol. See "Configuring Diameter Applications" for more information.

If you experience problems connecting to a Diameter peer node, verify that you have configured the correct protocol for communicating with the peer in "Configuring Peer Nodes". Be aware that Converged Application Server tries only the protocol you specify for the peer configuration (or TCP if you do not specify a protocol).

Part III

Monitoring and Troubleshooting

This part provides information on operating and maintaining Oracle Communications Converged Application Server. It includes information on starting and stopping servers, logging, diagnostics, SNMP traps, upgrading Converged Application Server software and deployed SIP applications, and avoiding and recovering from server failure.

This part contains the following chapters:

15 Configuring SNMP

This chapter describes how to configure and manage SNMP services with Oracle Communications Converged Application Server:

- Overview of Converged Application Server SNMP
- Browsing the MIB
- Configuring SNMP
- Understanding and Responding to SNMP Traps

Overview of Converged Application Server SNMP

Converged Application Server includes a dedicated SNMP MIB to monitor activity on engine tier and SIP data tier server instances. The Converged Application Server MIB is available on both Managed Servers and the Administration Server of a domain. However, Converged Application Server engine and SIP data tier traps are generated only by the Managed Server instances that make up each tier. If your Administration Server is not a target for the `sipserver` custom resource, it will generate only WebLogic Server SNMP traps (for example, when a server in a cluster fails). Administrators should monitor both WebLogic Server and Converged Application Server traps to evaluate the behavior of the entire domain.

Note: Converged Application Server MIB objects are read-only. You cannot modify a Converged Application Server configuration using SNMP.

Browsing the MIB

The Converged Application Server MIB file is installed in *`WLSS_HOME`*`/server/lib/WLSS-MIB.asn1`. Use an available SNMP management tool or MIB browser to view the contents of this file. See also "Trap Descriptions" for a description of common SNMP traps.

Configuring SNMP

To enable SNMP monitoring for the entire Converged Application Server domain, follow these steps:

1. Login to the Administration Console for the Converged Application Server domain.
2. In the left pane, select the **Diagnostics > SNMP** node.

3. In the Server SNMP Agents table, click the **New** button to create a new agent.

   Note: Ensure that you create a new Server SNMP agent, rather than a Domain-Scoped agent.

   ---

4. Enter a unique name for the new SNMP agent (for example, "engine1snmp") and click **OK**.
5. Select the newly-created SNMP agent from the Server SNMP Agents table.
6. Select**Configuration**, then select the **General** tab:

   Note: You can also set this parameter to true by selecting the Symmetric Response Routing option. To do this, select **Configuration**, then select the **General** tab of the SipServer Administration console extension.

   ---

   a. Select the Enabled check box to enable the agent.
   b. Enter an unused port number in the SNMP UDP Port field.

   Note: If you run multiple Managed Server instances on the same machine, each server instance must use a dedicated SNMP agent with a unique SNMP port number.

   ---

   c. Click **Save**.
7. Repeat the above steps to generate a unique SNMP agent for each server in your deployment (SIP data tier server, engine tier server, and Administration Server).

Understanding and Responding to SNMP Traps

The following sections describe the Converged Application Server SNMP traps in more detail. Recovery procedures for responding to individual traps are also included where applicable.

Files for Troubleshooting

The following Converged Application Server log and configuration files are frequently helpful for troubleshooting problems, and may be required by your technical support contact:

- `$`*`DOMAIN_HOME`*`/config/config.xml`
- `$`*`DOMAIN_HOME`*`/config/custom/sipserver.xml`
- `$`*`DOMAIN_HOME`*`/`*`servername`*`/*.log (server and message logs)`
- `sip.xml (in the /WEB-INF subdirectory of the application)`
- `web.xml (in the /WEB-INF subdirectory of the application)`

General information that can help the technical support team includes:

- The specific versions of:
  - Converged Application Server

  – Java SDK
  – Operating System
- Thread dumps for hung Converged Application Server processes
- Network analyzer logs

Trap Descriptions

Table 15–1 lists the Converged Application Server SNMP traps and indicates whether the trap is generated by servers in the engine tier or SIP data tier. Each trap is described in the sections that follow.

Table 15–1 Converged Application Server SNMP Traps

Server Node in which Trap is Generated	Trap Name
Engine Tier Servers	connectionLostToPeer
Engine Tier Servers	connectionReestablishedToPeer
Engine Tier Servers	overloadControlActivated, overloadControlDeactivated
Engine Tier Servers	sipAppDeployed
Engine Tier Servers	sipAppUndeployed
Engine Tier Servers	sipAppFailedToDeploy
Engine and SIP Data Tier Servers, if servers are members of a cluster	serverStopped
SIP Data Tier Servers	dataTierServerStopped
SIP Data Tier Servers	replicaAddedToPartition
SIP Data Tier Servers	replicaRemovedEnginesRegistration
SIP Data Tier Servers	replicaRemovedFromPartition

connectionLostToPeer

This trap is generated by an engine tier server instance when it loses its connection to a replica in the SIP data tier. It may indicate a network connection problem between the engine and SIP data tiers, or may be generated with additional traps if a SIP data tier server fails.

Recovery Procedure: If this trap occurs in isolation from other traps indicating a server failure, it generally indicates a network failure. Verify or repair the network connection between the affected engine tier server and the SIP data tier server.

If the trap is accompanied by additional traps indicating a SIP data tier server failure (for example, `dataTierServerStopped`), follow the recovery procedures for the associated traps.

connectionReestablishedToPeer

This trap is generated by an engine tier server instance when it successfully reconnects to a SIP data tier server after a prior failure (after a `connectionLostToPeer` trap was generated). Repeated instances of this trap may indicate an intermittent network failure between the engine and SIP data tiers.

Recovery Procedure: See "connectionLostToPeer".

dataTierServerStopped

Converged Application Server SIP data tier nodes generate this alarm when an unrecoverable error occurs in a WebLogic Server instance that is part of the SIP data tier. Note that this trap may be generated by the server that is shutting down, by another replica in the same partition, or in some cases by both servers (network outages can sometimes trigger both servers to generate the same trap).

Recovery Procedure: See the Recovery Procedure for "serverStopped".

overloadControlActivated, overloadControlDeactivated

Converged Application Server engine tier nodes use a configurable throttling mechanism that helps you control the number of new SIP requests that are processed. After a configured overload condition is observed, Converged Application Server destroys new SIP requests by responding with "503 Service Unavailable" to the caller. The servers continues to destroy new requests until the overload condition is resolved according to a configured threshold control value. This alarm is generated when the throttling mechanism is activated. The throttling behavior should eventually return the server to a non-overloaded state, and further action may be unnecessary. See "overload" in Chapter 21, "Engine Tier Configuration Reference (sipserver.xml)" for more information.

Recovery Procedure: Follow this recovery procedure:

1. Check other servers to see if they are nearly overloaded.
2. Check to see if the load balancer is correctly balancing load across the application servers, or if it is overloading one or more servers. If additional servers are nearly overloaded, Notify Tier 4 support immediately.
3. If the issue is limited to one server, notify Tier 4 support within one hour.

Additional Overload Information: If you set the queue length as an incoming call overload control, you can monitor the length of the queue using the Administration Console. If you specify a session rate control, you cannot monitor the session rate using the Administration Console. (The Administration Console only displays the current number of SIP sessions, not the rate of new sessions generated.)

replicaAddedToPartition

Converged Application Server SIP data tier nodes generate this alarm when a server instance is added to a partition in the SIP data tier.

Recovery Procedure: This trap is generated during normal startup procedures when SIP data tier servers are booted.

replicaRemovedEnginesRegistration

SIP data tier nodes generate this alarm if an engine server client that was not registered (or was removed from the list of registered engines) attempts to communicate with the SIP data tier. This trap is generally followed by a `serverStopped` trap indicating that the engine tier server was shut down to preserve SIP data tier consistency.

Recovery Procedure: Restart the engine tier server. Repeated occurrences of this trap may indicate a network problem between the engine tier server and one or more replicas.

replicaRemovedFromPartition

Converged Application Server SIP data tier nodes generate this alarm when a server is removed from the SIP data tier, either as a result of a normal shutdown operation or because of a failure. There must be at least one replica remaining in a partition to generate this trap; if a partition has only a single replica and that replica fails, the trap cannot be generated. In addition, because engine tier nodes determine when a replica has failed, an engine tier node must be running in order for this trap to be generated.

Recovery Procedure: If this trap is generated as a result of a server instance failure, additional traps will be generated to indicate the exception. See the recovery procedures for traps generated in addition to `replicaRemovedFromPartition`.

serverStopped

This trap indicates that the WebLogic Server instance is now down. This trap applies to both engine tier and SIP data tier server instances, but only when the servers are members of a named WebLogic Server cluster. If this trap is received spontaneously and not as a result of a controlled shutdown, follow the steps below.

Recovery Procedure: Follow this recovery procedure:

1. Use the following command to identify the hung process:

   ```
   ps -ef | grep java
   ```

   There should be only one PID for each WebLogic Server instance running on the machine.
2. After identifying the affected PID, use the following command to kill the process:

   ```
   kill -3 [pid]
   ```
3. This command generates the actual thread dump. If the process is not immediately killed, repeat the command several times, spaced 5-10 seconds apart, to help diagnose potential deadlock problems, until the process is killed.
4. Attempt to restart Converged Application Server immediately.
5. Make a backup copy of all SIP logs on the affected server to aid in troubleshooting. The location of the logs varies based on the server configuration.
6. Copy each log to assist Tier 4 support with troubleshooting the problem.

   Note: Converged Application Server logs are truncated according to your system configuration. Make backup logs immediately to avoid losing critical troubleshooting information.
7. Notify Tier 4 support and include the log files with the trouble ticket.
8. Monitor the server closely over next 24 hours. If the source of the problem cannot be identified in the log files, there may be a hardware or network issue that will reappear over time.

Additional Shutdown Information: The Administration Console generates SNMP messages for managed WebLogic Server instances only until the ServerShutDown message is received. Afterwards, no additional messages are generated.

sipAppDeployed

Converged Application Server engine tier nodes generate this alarm when a SIP Servlet is deployed to the container.

Recovery Procedure: This trap is generated during normal deployment operations and does not indicate an exception.

sipAppUndeployed

Converged Application Server engine tier nodes generate this alarm when a SIP application shuts down, or if a SIP application is undeployed. This generally occurs when Converged Application Server is shutdown while active requests still exist.

Recovery Procedure: During normal shutdown procedures this alarm should be filtered out and should not reach operations. If the alarm occurs during the course of normal operations, it indicates that someone has shutdown the application or server unexpectedly, or there is a problem with the application. Notify Tier 4 support immediately.

sipAppFailedToDeploy

Converged Application Server engine tier nodes generate this trap when an application deploys successfully as a Web Application but fails to deploy as a SIP application.

Recovery Procedure: The typical failure is caused by an invalid **sip.xml** configuration file and should occur only during software installation or upgrade procedures. When it occurs, undeploy the application, validate the **sip.xml** file, and retry the deployment.

Note: This alarm should never occur during normal operations. If it does, contact Tier 4 support immediately.

16 Using the WebLogic Server Diagnostic Framework (WLDF)

This chapter describes the integration of Oracle Communications Converged Application Server with the WebLogic Diagnostic Framework (WLDF):

Overview of Converged Application Server and the WLDF

The WebLogic Diagnostic Framework (WLDF) consists of a number of components that work together to collect, archive, and access diagnostic information about a WebLogic Server instance and its applications. Converged Application Server version integrates with several components of the WLDF in order to monitor and diagnose the operation of engine and SIP data tier nodes, as well as deployed SIP Servlets:

- Data Collectors: Converged Application Server integrates with the Harvester service to collect information from runtime MBeans, and with the Logger service to archive SIP requests and responses.
- Watches and Notifications: Administrators can use the Watches and Notifications component to create complex rules, based on Converged Application Server runtime MBean attributes, that trigger automatic notifications using JMS, JMX, SNMP, SMTP, and so forth.
- Image Capture: Converged Application Server instances can collect certain diagnostic data and write the data to an image file when requested by an Administrator. This data can then be used to diagnose problems in a running server.
- Instrumentation: Converged Application Server instruments the server and application code with monitors to help you configure diagnostic actions that are performed on SIP messages (requests and responses) that match certain criteria.

The sections that follow provide more details about how Converged Application Server integrates with each of the above WLDF components. See *Configuring and Using* *the Diagnostics Framework for Oracle WebLogic Server* in the Oracle Fusion Middleware documentation for more information about WLDF.

Data Collection and Logging

Converged Application Server uses the WLDF Harvester service to collect data from the attributes of these runtime MBeans:

- `ReplicaRuntimeMBean`
- `SipApplicationRuntimeMBean`
- `SipServerRuntimeMBean`

You can add charts and graphs of this data to your own custom views using the WLDF console extension. To do so, first enable the WLDF console extension by copying the JAR file into the `console-ext` subdirectory of your domain directory:

```
cp ~/ORACLE_HOME/Middleware/wlserver_
10.3/server/lib/console-ext/diagnostics-console-extension.jar ~/ORACLE_
HOME/Middleware/user_projects/domains/mydomain/console-ext
```

When accessing the WLDF console extension, the Converged Application Server runtime MBean attributes are available in the Metrics tab of the extension.

Converged Application Server also uses the WLDF Logger service to archive SIP and Diameter messages to independent, dedicated log files (by default, *`domain_home`*`/logs/`*`server_name`*`/sipMessages.log`). You can configure the name and location of the log file, as well as log rotation policies, using the Configuration > Message Debug tab in the SIP Server Administration Console extension. See "Enabling Message Logging" in *Converged Application Server Developer's Guide*. Note that a server restart is necessary in order to initiate independent logging and log rotation.

Watches and Notifications

The data collected from Converged Application Server runtime MBeans can be used to create automated monitors, or "watches," that observe a server's diagnostic state. One or more notifications can then be configured for use by a watch, in order to generate a message using SMTP, SNMP, JMX, or JMS when your configured watch conditions and rules occur.

To use watches and notifications, you select the Diagnostics > Diagnostic Modules node in the left pane of the Administration Console and create a new module with the watch rules and notifications required for monitoring your servers. The watch rules can use the metrics collected from Converged Application Server runtime MBeans, messages written to the log file, or events generated by the diagnostic framework.

Image Capture

Converged Application Server adds its own image capture information to the diagnostic image generated by the WLDF. You can generate diagnostic images either on demand, or automatically by configuring watch rules.

The information contained in diagnostic images is intended for use by Oracle technical support personnel when troubleshooting a potential server problem and includes:

- SIP data tier partition and replica configuration.
- Call state and timer statistics.

- Work manager statistics.

Instrumentation

The WLDF instrumentation system creates diagnostic monitors and inserts them into Converged Application Server or application code at specific points in the flow of execution. Converged Application Server integrates with the instrumentation service to provide a built-in DyeInjection monitor. When enabled, this monitor injects dye flags into the diagnostic context when certain SIP messages enter or exist the system. Dye flags are injected based on the monitor's configuration properties, and on certain request attributes.

Converged Application Server adds the dye flags described in Table 16–1 below, as well as the WebLogic Server dye flags USER and ADDR. See *Oracle Fusion Middleware Configuring and Using the Diagnostics Framework for Oracle WebLogic Server* for more information.

Table 16–1 Converged Application Server DyeInjection Flags

Dye Flag	Description
PROTOCOL_SIP	Set in the diagnostic context of all SIP protocol messages.
SIP_REQ	Set in the diagnostic context for all SIP requests that match the value of the property SIP_REQ.
SIP_RES	Set in the diagnostic context for all SIP responses that match the value of the property SIP_RES.
SIP_REQURI	Set if a SIP request's request URI matches the value of property SIP_REQURI.
SIP_ANY_HEADER	Set if a SIP request contains a header matching the value of the property SIP_ANY_HEADER. The value of SIP_ANY_HEADER is specified using the format `*messageType.headerName*=headerValue` where `*headerValue*` is either a value or regular expression. For example, you can specify the property as SIP_ANY_HEADER=request.Contact=sip:sipp@localhost:5061 or SIP_ANY_ HEADER=response.Contact=sip:findme@172.17.30.50:5060.

Dye flags can be applied to both incoming and outbound SIP messages. The flags are useful for dye filtering, and can be used by delegating monitors to trigger further diagnostic actions.

Converged Application Server provides several delegating monitors that can be applied at the application and server scope, and which may examine dye flags set by the `DyeInjection` monitor. The delegating monitors are described in Table 16–2.

Table 16–2 Converged Application Server Diagnostic Monitors

Monitor Name	**Monitor Type**	**Scope**	**Pointcuts**
occas/Sip_Servlet_Before_Service	Before	Application	At entry of `SipServlet.do*` or `SipServlet.service` methods of all implementing subclasses.
occas/Sip_Servlet_After_Service	After	Application	At exit of `SipServlet.do*` or `SipServlet.service` methods of all implementing subclasses.
occas/Sip_Servlet_Around_Service	Around	Application	At entry and exit of `SipServlet.do*` or `SipServlet.service` methods of all implementing subclasses.
occas/Sip_Servlet_Before_Session	Before	Application	At entry of `getAttribute`, `set`, `remove`, and `invalidate` methods for both `SipSession` and `SipApplicationSession`.
occas/Sip_Servlet_After_Session	After	Application	At exit of `getAttribute`, `set`, `remove`, and `invalidate` methods for both `SipSession` and `SipApplicationSession`.
occas/Sip_Servlet_Around_Session	Around	Application	At entry and exit of `getAttribute`, `set`, `remove`, and `invalidate` methods for both `SipSession` and `SipApplicationSession`.
occas/SipSessionDebug	Around	Application	This is a built-in, application-scoped monitor having fixed pointcuts and a fixed debug action. Before and after a pointcut, the monitor performs the `SipSessionDebug` diagnostic action, which calculates the size of the SIP session after serializing the underlying object. The pointcuts for this monitor are as follows: **1.** Before and after calls to `getSession` and `getApplicationSession` of the `SipServletMessage` class hierarchy. **2.** Before and after calls to `getAttribute`, `setAttribute`, and `removeAttribute` methods in the `SipSession` and `SipApplicationSession` classes. **Note:** The `occas/SessionDebugAction-Before` event is not triggered for the `req.getSession()` or `req.getApplicationSession()` joinpoints. Only the `occas/SessionDebugAction-After` is triggered, because the Session is made available for inspection only after the joinpoints have executed. **Note:** If you compile your application using Apache Ant, you must enable the `debug` attribute to embed necessary debug information into the generated class files.

Table 16–2 (Cont.) Converged Application Server Diagnostic Monitors

Monitor Name	Monitor Type	Scope	Pointcuts
occas/Sip_Servlet_Before_Message_Send_Internal	Before	Server	At entry of Converged Application Server code that writes messages to the wire.
occas/Sip_Servlet_After_Message_Send_Internal	After	Server	At exit of Converged Application Server code that writes messages to the wire.
occas/Sip_Servlet_Around_Message_Send_Internal	Around	Server	At entry and exit of Converged Application Server code that writes messages to the wire.

Configuring Server-Scoped Monitors

To use the server-scoped monitors, you must create a new diagnostic module and create and configure one or more monitors in the module. For the built-in DyeInjection monitor, you then add monitor properties to define the specific dye flags. For delegating monitors such as `occas/Sip_Servlet_Before_Message_Send_Internal`, you add monitor properties to define diagnostic actions.

Follow these steps to configure the Converged Application Server DyeInjection monitor, a delegate monitor, and enable dye filtering:

1. Access the Administration Console for you domain.
2. Select **Diagnostics**, then select the **Diagnostic Modules** node in the left pane of the console.
3. Click New to create a new Diagnostic Module. Give the module a descriptive name, such as "instrumentationModule," and click OK.
4. Select the new "instrumentationModule" from the list of modules in the table.
5. Select the **Targets** tab.
6. Select a server on which to target the module and click Save.
7. Return to the **Diagnostic Modules** node and select instrumentationModule from the list of modules.
8. Select **Configuration**, then select the **Instrumentation** tab.
9. Select Enabled to enable instrumentation at the server level, then click Save.
10. Add the DyeInjection monitor to the module:
    a. Click **Add/Remove**.
    b. Select the name of a monitor from the Available list (for example, DyeInjection), and use the arrows to move it to the Chosen list.
    c. Click OK.
    d. Select the newly-created monitor from the list of available monitors.
    e. Ensure that the monitor is enabled, and edit the Properties field to add any required properties. For the DyeInjection monitor, sample properties include:

    ```
    SIP_RES=180
    SIP_REQ=INVITE
    SIP_ANY_HEADER=request.Contact=sip:sipp@localhost:5061
    ```

    f. Click Save
11. Add one or more delegate monitors to the module:

a. Return to the Configuration > Instrumentation tab for the new module.

b. Click Add/Remove.

c. Select the name of a delegate monitor from the Available list (for example, `occas/Sip_Servlet_Before_Message_Send_Internal`), and use the arrows to move it to the Chosen list.

d. Click OK.

e. Select the newly-created monitor from the list of available monitors.

f. Ensure that the monitor is enabled, then select one or more Actions from the available list, and use the arrows to move the actions to the Chosen list. For the `occas/Sip_Servlet_Before_Message_Send_Internal` monitor, sample actions include `DisplayArgumentsAction`, `StackDumpAction`, `ThreadDumpAction`, and `TraceAction`.

g. Select the check box to EnableDyeFiltering.

h. Select one or more Dye Masks, such as SIP_REQ, from the Available list and use the arrows to move them to the Chosen list.

i. Click Save

Note: You can repeat the above steps to create additional delegate monitors.

Configuring Application-Scoped Monitors

You configure application-scoped monitors in an XML configuration file named **weblogic-diagnostics.xml**. You must store the **weblogic-diagnostics.xml** file in the SIP module's or enterprise application's **META-INF** directory.

The XML file enables instrumentation at the application level, defines point cuts, and also defines delegate monitor dye masks and actions. Example 16–1 shows a sample configuration file that uses the `occas/Sip_Servlet_Before_Service` monitor.

Example 16–1 Sample weblogic-diagnostics.xml File

```
<wldf-resource xmlns="http://www.bea.com/ns/weblogic/90/diagnostics">
  <instrumentation>
    <enabled>true</enabled>
    <include>demo.ProxyServlet</include>
    <wldf-instrumentation-monitor>
      <name>occas/Sip_Servlet_Before_Service</name>
      <enabled>true</enabled>
      <dye-mask>SIP_ANY_HEADER</dye-mask>
      <dye-filtering-enabled>true</dye-filtering-enabled>
      <action>DisplayArgumentsAction</action>
    </wldf-instrumentation-monitor>
   </instrumentation>
</wldf-resource>
```

In this example, if an incoming request's diagnostic context contains the `SIP_ANY_HEADER` dye flag, then the `occas/Sip_Servlet_Before_Service` monitor is triggered and the `DisplayArgumentsAction` is executed.

See "Configuring Instrumentation" in *Configuring and Using the Diagnostics Framework for Oracle WebLogic Server* in the Oracle Fusion Middleware documentation for more information about creating the **weblogic-diagnostics.xml** configuration file.

17 Logging SIP Requests and Responses

This chapter describes how to configure and manage logging for SIP requests and responses that Oracle Communications Converged Application Server processes:

Overview of SIP Logging

Converged Application Server enables you to perform Protocol Data Unit (PDU) logging for the SIP requests and responses it processes. Logged SIP messages are placed either in the domain-wide log file for Converged Application Server, or in the log files for individual Managed Server instances. Because SIP messages share the same log files as Converged Application Server instances, you can use advanced server logging features such as log rotation, domain log filtering, and maximum log size configuration when managing logged SIP messages.

Administrators configure SIP PDU logging by defining one or more SIP Servlets using the `com.bea.wcp.sip.engine.tracing.listener.TraceMessageListenerImpl` class. Logging criteria are then configured either as parameters to the defined servlet, or in separate XML files packaged with the application.

As SIP requests are processed or SIP responses generated, the logging Servlet compares the message with the filtering patterns defined in a standalone XML configuration file or Servlet parameter. SIP requests and responses that match the specified pattern are written to the log file along with the name of the logging servlet, the configured logging level, and other details. To avoid unnecessary pattern matching, the Servlet marks new SIP Sessions when an initial pattern is matched and then logs subsequent requests and responses for that session automatically.

Logging criteria are defined either directly in **sip.xml** as parameters to a logging Servlet, or in external XML configuration files. See "Specifying the Criteria for Logging Messages".

Note: Engineers can implement PDU logging functionality in their Servlets either by creating a delegate with the `TraceMessageListenerFactory` in the Servlet's `init()` method, or by using the tracing class in deployed Java applications. Using the delegate enables you to perform custom logging or manipulate incoming SIP messages using the default trace message listener implementation. See "Adding Tracing Functionality to SIP Servlet Code" for an example of using the factory in a Servlet's `init()` method.

Defining Logging Servlets in sip.xml

Logging Servlets for SIP messages are created by defining Servlets having the implementation class `com.bea.wcp.sip.engine.tracing.listener.TraceMessageListenerImpl`. The definition for a sample `msgTraceLogger` is shown in Example 17–1.

Example 17–1 Sample Logging Servlet

```
<servlet>
    <servlet-name>msgTraceLogger</servlet-name>

<servlet-class>com.bea.wcp.sip.engine.tracing.listener.TraceMessageListenerImpl</s
ervlet-class>
    <init-param>
      <param-name>domain</param-name>
      <param-value>true</param-value>
    </init-param>
    <init-param>
      <param-name>level</param-name>
      <param-value>full</param-value>
    </init-param>
    <load-on-startup/>
  </servlet>
```

Configuring the Logging Level and Destination

Logging attributes such as the level of logging detail and the destination log file for SIP messages are passed as initialization parameters to the logging Servlet. Table 17–1 lists the parameters and parameter values that you can specify as `init-param` entries. Example 17–1 shows the sample `init-param` entries for a Servlet that logs full SIP message information to the domain log file.

Specifying the Criteria for Logging Messages

The criteria for selecting SIP messages to log can be defined either in XML files that are packaged with the logging Servlet's application, or as initialization parameters in the Servlet's **sip.xml** deployment descriptor. The sections that follow describe each method.

Using XML Documents to Specify Logging Criteria

If you do not specify logging criteria as an initialization parameter to the logging Servlet, the Servlet looks for logging criteria in a pair of XML descriptor files in the top level of the logging application. These descriptor files, named **request-pattern.xml** and

response-pattern.xml, define patterns that Converged Application Server uses for selecting SIP requests and responses to place in the log file.

Note: By default Converged Application Server logs both requests and responses. If you do not want to log responses, you must define a **response-pattern.xml** file with empty matching criteria.

A typical pattern definition defines a condition for matching a particular value in a SIP message header. For example, the sample **response-pattern.xml** used by the `msgTraceLogger` Servlet matches all MESSAGE requests. The contents of this descriptor are shown in

Example 17–2 Sample response-pattern.xml for msgTraceLogger Servlet

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE pattern
   PUBLIC "Registration//Organization//Type Label//Definition Language"
   "trace-pattern.dtd">
<pattern>
  <equal>
    <var>response.method</var>
    <value>MESSAGE</value>
  </equal>
</pattern>
```

Additional operators and conditions for matching SIP messages are described in "trace-pattern.dtd Reference". Most conditions, such as the `equal` condition shown in Example 17–2, require a variable (`var` element) that identifies the portion of the SIP message to evaluate. Table 17–1 lists some common variables and sample values. For additional variable names and examples, see *Section 16: Mapping Requests to Servlets* in the SIP Servlet API 1.1 specification (`http://jcp.org/en/jsr/detail?id=289`); Converged Application Server enables mapping of both request and response variables to logging Servlets.

Table 17–1 Pattern-matching Variables and Sample Values

Variable	Sample Values
request.method, response.method	MESSAGE, INVITE, ACK, BYE, CANCEL
request.uri.user, response.uri.user	guest, admin, joe
request.to.host, response.to.host	server.mydomain.com

Both **request-pattern.xml** and **response-pattern.xml** use the same Document Type Definition (DTD). See "trace-pattern.dtd Reference" for more information.

Using Servlet Parameters to Specify Logging Criteria

Pattern-matching criteria can also be specified as initialization parameters to the logging Servlet, rather than as separate XML documents. The parameter names used to specify matching criteria are `request-pattern-string` and `response-pattern-string`. They are defined along with the logging level and destination as described in "Configuring the Logging Level and Destination".

The value of each pattern-matching parameter must consist of a valid XML document that adheres to the DTD for standalone pattern definition documents (see "Using XML Documents to Specify Logging Criteria"). Because the XML documents that define the patterns and values must not be parsed as part of the **sip.xml** descriptor, you must enclose the contents within the `CDATA` tag. Example 17–3 shows the full **sip.xml** entry for the sample logging Servlet, `invTraceLogger`. The final two `init-param` elements specify that the Servlet log only `INVITE` request methods and `OPTIONS` response methods.

Example 17–3 Logging Criteria Specified as init-param Elements

```
<servlet>
      <servlet-name>invTraceLogger</servlet-name>

<servlet-class>com.bea.wcp.sip.engine.tracing.listener.TraceMessageListenerImpl</s
ervlet-class>
      <init-param>
        <param-name>domain</param-name>
        <param-value>true</param-value>
      </init-param>
      <init-param>
        <param-name>level</param-name>
        <param-value>full</param-value>
      </init-param>
      <init-param>
        <param-name>request-pattern-string</param-name>
        <param-value>
            <![CDATA[
                <?xml version="1.0" encoding="UTF-8"?>
                <!DOCTYPE pattern
                   PUBLIC "Registration//Organization//Type Label//Definition
Language"
                   "trace-pattern.dtd">
                <pattern>
                  <equal>
                    <var>request.method</var>
                    <value>INVITE</value>
                  </equal>
                </pattern>
            ]]>
        </param-value>
      </init-param>
      <init-param>
        <param-name>response-pattern-string</param-name>
        <param-value>
            <![CDATA[
                <?xml version="1.0" encoding="UTF-8"?>
                <!DOCTYPE pattern
                   PUBLIC "Registration//Organization//Type Label//Definition
Language"
                   "trace-pattern.dtd">
                <pattern>
                  <equal>
                    <var>response.method</var>
                    <value>OPTIONS</value>
                  </equal>
                </pattern>
            ]]>
        </param-value>
      </init-param>
      <load-on-startup/>
  </servlet>
```

Specifying Content Types for Unencrypted Logging

By default Converged Application Server uses String format (UTF-8 encoding) to log the content of SIP messages having a text or application/sdp Content-Type value. For all other Content-Type values, Converged Application Server attempts to log the message content using the character set specified in the `charset` parameter of the message, if one is specified. If no `charset` parameter is specified, or if the `charset` value is invalid or unsupported, Converged Application Server uses Base-64 encoding to encrypt the message content before logging the message.

If you want to avoid encrypting the content of messages under these circumstances, specify a list of String-representable Content-Type values using the `string-rep` element in **sipserver.xml**. The `string-rep` element can contain one or more `content-type` elements to match. If a logged message matches one of the configured `content-type` elements, Converged Application Server logs the content in String format using UTF-8 encoding, regardless of whether or not a `charset` parameter is included.

Note: You do not need to specify text/* or application/sdp content types as these are logged in String format by default.

Example 17–4 shows a sample `message-debug` configuration that logs String content for three additional Content-Type values, in addition to text/* and application/sdp content.

Example 17–4 Logging String Content for Additional Content Types

```
<message-debug>
  <level>full</level>
  <string-rep>
    <content-type>application/msml+xml</content-type>
    <content-type>application/media_control+xml</content-type>
    <content-type>application/media_control</content-type>
  </string-rep>
</message-debug>
```

Enabling Log Rotation and Viewing Log Files

The Converged Application Server logging infrastructure enables you to automatically write to a new log file when the existing log file reaches a specified size. You can also view log contents using the Administration Console or configure additional server-level events that are written to the log.

trace-pattern.dtd Reference

`trace-pattern.dtd` defines the required contents of the **request-pattern.xml** and **response-pattern.xml**, documents, as well as the values for the `request-pattern-string` and `response-pattern-string` Servlet `init-param` variables.

Example 17–5 trace-pattern.dtd

```
<!--
The different types of conditions supported.
- >
```

```
<!ENTITY % condition "and | or | not |
                      equal | contains | exists | subdomain-of">

<!--
A pattern is a condition: a predicate over the set of SIP requests.
- >

<!ELEMENT pattern (%condition;)>

<!--
An "and" condition is true if and only if all its constituent conditions
are true.
- >

<!ELEMENT and (%condition;)+>

<!--
An "or" condition is true if at least one of its constituent conditions
is true.
- >

<!ELEMENT or (%condition;)+>

<!--
Negates the value of the contained condition.
- >

<!ELEMENT not (%condition;)>

<!--
True if the value of the variable equals the specified literal value.
- >

<!ELEMENT equal (var, value)>

<!--
True if the value of the variable contains the specified literal value.
- >

<!ELEMENT contains (var, value)>

<!--
True if the specified variable exists.
- >

<!ELEMENT exists (var)>

<!--
- >

<!ELEMENT subdomain-of (var, value)>

<!--
Specifies a variable. Example:
  <var>request.uri.user</var>
- >

<!ELEMENT var (#PCDATA)>

<!--
```

```
Specifies a literal string value that is used to specify rules.
- >

<!ELEMENT value (#PCDATA)>

<!--
Specifies whether the "equal" test is case sensitive or not.
- >

<!ATTLIST equal ignore-case (true|false) "false">

<!--
Specifies whether the "contains" test is case sensitive or not.
- >

<!ATTLIST contains ignore-case (true|false) "false">

<!--
The ID mechanism is to allow tools to easily make tool-specific
references to the elements of the deployment descriptor. This allows
tools that produce additional deployment information (i.e information
beyond the standard deployment descriptor information) to store the
non-standard information in a separate file, and easily refer from
these tools-specific files to the information in the standard sip-app
deployment descriptor.
- >

<!ATTLIST pattern id ID #IMPLIED>
<!ATTLIST and id ID #IMPLIED>
<!ATTLIST or id ID #IMPLIED>
<!ATTLIST not id ID #IMPLIED>
<!ATTLIST equal id ID #IMPLIED>
<!ATTLIST contains id ID #IMPLIED>
<!ATTLIST exists id ID #IMPLIED>
<!ATTLIST subdomain-of id ID #IMPLIED>
<!ATTLIST var id ID #IMPLIED>
<!ATTLIST value id ID #IMPLIED>
```

Adding Tracing Functionality to SIP Servlet Code

Tracing functionality can be added to your own Servlets or to Java code by using the `TraceMessageListenerFactory`. `TraceMessageListenerFactory` enables clients to reuse the default trace message listener implementation behaviors by creating an instance and then delegating to it. The factory implementation instance can be found in the servlet context for SIP Servlets by looking up the value of the `TraceMessageListenerFactory.TRACE_MESSAGE_LISTENER_FACTORY` attribute.

Note: Instances created by the factory are not registered with Converged Application Server to receive callbacks upon SIP message arrival and departure.

To implement tracing in a Servlet, you use the factory class to create a delegate in the Servlet's `init()` method as shown in Example 17–6.

Example 17–6 Using the TraceMessageListenerFactory

```
public final class TraceMessageListenerImpl extends SipServlet implements
```

```
MessageListener {
  private MessageListener delegate;

  public void init() throws ServletException {
    ServletContext sc = (ServletContext) getServletContext();
    TraceMessageListenerFactory factory = (TraceMessageListenerFactory)
sc.getAttribute(TraceMessageListenerFactory.TRACE_MESSAGE_LISTENER_FACTORY);
    delegate = factory.createTraceMessageListener(getServletConfig());
  }
  public final void onRequest(SipServletRequest req, boolean incoming) {
    delegate.onRequest(req,incoming);
  }
  public final void onResponse(SipServletResponse resp, boolean incoming) {
    delegate.onResponse(resp,incoming);
  }
}
```

Order of Startup for Listeners and Logging Servlets

If you deploy both listeners and logging servlets, the listener classes are loaded first, followed by the Servlets. Logging Servlets are deployed in order according to the load order specified in their Web Application deployment descriptor.

18 Avoiding and Recovering From Server Failures

This chapter describes the Oracle Communications Converged Application Server failure prevention and recovery features, and describes the configuration artifacts that are required in order to restore different portions of a Converged Application Server domain:

- Failure Prevention and Automatic Recovery Features
- Directory and File Backups for Failure Recovery
- Restarting a Failed Administration Server
- Restarting Failed Managed Servers

A variety of events can lead to the failure of a server instance. Often one failure condition leads to another. Loss of power, hardware malfunction, operating system crashes, network partitions, or unexpected application behavior may each contribute to the failure of a server instance.

Converged Application Server uses a highly clustered architecture as the basis for minimizing the impact of failure events. However, even in a clustered environment it is important to prepare for a sound recovery process in the event that an individual server or server machine fails.

Failure Prevention and Automatic Recovery Features

Converged Application Server, and the underlying WebLogic Server platform, provide many features that protect against server failures. In a production system, all available features should be used in order to ensure uninterrupted service.

Overload Protection

Converged Application Server detects increases in system load that could affect the performance and stability of deployed SIP Servlets, and automatically throttles message processing at predefined load thresholds.

Using overload protection helps you avoid failures that could result from unanticipated levels of application traffic or resource utilization.

Converged Application Server attempts to avoid failure when certain conditions occur:

- The rate at which SIP sessions are created reaches a configured value, or
- The size of the SIP timer and SIP request-processing execute queues reaches a configured length.

See "overload" in the chapter Chapter 21, "Engine Tier Configuration Reference (sipserver.xml)" for more information.

The underlying WebLogic Server platform also detects increases in system load that can affect deployed application performance and stability. WebLogic Server allows administrators to configure failure prevention actions that occur automatically at predefined load thresholds. Automatic overload protection helps you avoid failures that result from unanticipated levels of application traffic or resource utilization as indicated by:

- A workload manager's capacity being exceeded
- The HTTP session count increasing to a predefined threshold value
- Impending out of memory conditions

See the discussion on avoiding and managing overload in *Configuring Server Environments for Oracle WebLogic Server* in the Oracle WebLogic Server 11*g* documentation for more information.

Redundancy and Failover for Clustered Services

You can increase the reliability and availability of your applications by using multiple engine tier servers in a dedicated cluster, as well as multiple SIP data tier servers (replicas) in a dedicated SIP data tier cluster.

Because engine tier clusters maintain no stateful information about applications, the failure of an engine tier server does not result in any data loss or dropped calls. Multiple replicas in a SIP data tier partition store redundant copies of call state information, and automatically failover to one another should a replica fail.

See *Oracle Communications Converged Application Server Concepts* for more information.

Automatic Restart for Failed Server Instances

WebLogic Server self-health monitoring features improve the reliability and availability of server instances in a domain. Selected subsystems within each server instance monitor their health status based on criteria specific to the subsystem. (For example, the JMS subsystem monitors the condition of the JMS thread pool while the core server subsystem monitors default and user-defined execute queue statistics.) If an individual subsystem determines that it can no longer operate in a consistent and reliable manner, it registers its health state as "failed" with the host server.

Each WebLogic Server instance, in turn, checks the health state of its registered subsystems to determine its overall viability. If one or more of its critical subsystems have reached the FAILED state, the server instance marks its own health state FAILED to indicate that it cannot reliably host an application.

When used in combination with Node Manager, server self-health monitoring enables you to automatically reboot servers that have failed. This improves the overall reliability of a domain, and requires no direct intervention from an administrator. For more information, see the discussion on using Node Manager to start Managed Servers in a domain or cluster in the Oracle WebLogic Server 11*g* documentation.

Managed Server Independence Mode

Managed Servers maintain a local copy of the domain configuration. When a Managed Server starts, it contacts its Administration Server to retrieve any changes to the domain configuration that were made since the Managed Server was last shut down. If a Managed Server cannot connect to the Administration Server during startup, it can use its locally-cached configuration information—this is the configuration that was current at the time of the Managed Server's most recent shutdown. A Managed Server that starts up without contacting its Administration Server to check for configuration updates is running in *Managed Server Independence (MSI)* mode. By default, MSI mode is enabled. See "Replicate domain config files for Managed Server Independence" in the Administration Console online Help.

Automatic Migration of Failed Managed Servers

When using Linux or UNIX operating systems, you can use WebLogic Server's server migration feature to automatically start a candidate (backup) server if a Network tier server's machine fails or becomes partitioned from the network. The server migration feature uses node manager, in conjunction with the `wlsifconfig.sh` script, to automatically boot candidate servers using a floating IP address. Candidate servers are booted only if the primary server hosting a Network tier instance becomes unreachable. See the discussion on whole server migration in *Using Clusters for Oracle WebLogic Server* in the Oracle WebLogic Server 11*g* documentation for more information about using the server migration feature.

Geographic Redundancy for Regional Site Failures

In addition to server-level redundancy and failover capabilities, Converged Application Server enables you to configure peer sites to protect against catastrophic failures, such as power outages, that can affect an entire domain. This enables you to failover from one geographical site to another, avoiding complete service outages.

Directory and File Backups for Failure Recovery

Recovery from the failure of a server instance requires access to the domain's configuration data. By default, the Administration Server stores a domain's primary configuration data in a file called ***DOMAIN_HOME*/config/config.xml**, where ***DOMAIN_HOME*** is the root directory of the domain. The primary configuration file may reference additional configuration files for specific WebLogic Server services, such as JDBC and JMS, and for Converged Application Server services, such as SIP container properties and SIP data tier configuration. The configuration for specific services are stored in additional XML files in subdirectories of the ***DOMAIN_HOME*/config** directory, such as ***DOMAIN_HOME*/config/jms**, ***DOMAIN_HOME*/config/jdbc**, and ***DOMAIN_HOME*/config/custom** for Converged Application Server configuration files.

The Administration Server can automatically archive multiple versions of the domain configuration (the entire ***DOMAIN_HOME*/config** directory). The configuration archives can be used for system restoration in cases where accidental configuration changes need to be reversed. For example, if an administrator accidentally removes a configured resource, the prior configuration can be restored by using the last automated backup.

The Administration Server stores only a finite number of automated backups locally in ***DOMAIN_HOME*/config**. For this reason, automated domain backups are limited in their ability to guard against data corruption, such as a failed hard disk. Automated backups also do not preserve certain configuration data that are required for full domain restoration, such as LDAP repository data and server start-up scripts. Oracle recommends that you also maintain multiple backup copies of the configuration and security offline, in a source control system.

This section describes file backups that Converged Application Server performs automatically, as well as manual backup procedures that an administrator should perform periodically.

Enabling Automatic Configuration Backups

Follow these steps to enable automatic domain configuration backups on the Administration Server for your domain:

1. Access the Administration Console for your domain.
2. In the left pane of the Administration Console, select the name of the domain.
3. In the right pane, click **Configuration**, and then select the **General** tab.
4. Select **Advanced** to display advanced options.
5. Select **Configuration Archive Enabled**.
6. In the Archive Configuration Count box, enter the maximum number of configuration file revisions to save.
7. Click **Save**.

When you enable configuration archiving, the Administration Server automatically creates a configuration JAR file archive. The JAR file contains a complete copy of the previous configuration (the complete contents of the ***DOMAIN_HOME*\config** directory). JAR file archive files are stored in the ***DOMAIN_HOME*configArchive** directory. The files use the naming convention **config-number.jar**, where `number` is the sequential number of the archive.

When you save a change to a domain's configuration, the Administration Server saves the previous configuration in ***DOMAIN_HOME*\configArchive\config.xml#n**. Each time the Administration Server saves a file in the `configArchive` directory, it increments the value of the `#n` suffix, up to a configurable number of copies—5 by default. Thereafter, each time you change the domain configuration:

- The archived files are rotated so that the newest file has a suffix with the highest number,
- The previous archived files are renamed with a lower number, and
- The oldest file is deleted.

Be aware that configuration archives are stored locally within the domain directory, and they may be overwritten according to the maximum number of revisions you selected. For these reasons, you must also create your own off-line archives of the domain configuration, as described in "Storing the Domain Configuration Offline".

Storing the Domain Configuration Offline

Although automatic backups protect against accidental configuration changes, they do not protect against data loss caused by a failure of the hard disk that stores the domain configuration, or accidental deletion of the domain directory. To protect against these failures, you must also store a complete copy of the domain configuration offline, preferably in a source control system.

Oracle recommends storing a copy of the domain configuration at regular intervals. For example, backup a new revision of the configuration when:

- You first deploy the production system
- You add or remove deployed applications

- The configuration is tuned for performance
- Any other permanent change is made.

The domain configuration backup should contain the complete contents of the *DOMAIN_HOME/config* directory. For example:

```
cd ~/ORACLE_HOME/Middleware/user_projects/domains/mydomain
tar cvf domain-backup-06-17-2007.jar config
```

Store the new archive in a source control system, preserving earlier versions should you need to restore the domain configuration to an earlier point in time.

Backing Up Server Start Scripts

In a Converged Application Server deployment, the start scripts used to boot engine and SIP data tier servers are generally customized to include domain-specific configuration information such as:

- JVM Garbage Collection parameters required to achieve throughput targets for SIP message processing (see "Modifying JVM Parameters in Server Start Scripts" in Chapter 19, "Tuning JVM Garbage Collection for Production Deployments." Different parameters (and therefore, different start scripts) are generally used to boot engine and SIP data tier servers.
- Configuration parameters and startup information for the Converged Application Server heartbeat mechanism (see "Enabling and Configuring the Heartbeat Mechanism on Servers" in Chapter 6, "Configuring Server Failure Detection." If you use the heartbeat mechanism, engine tier server start scripts should include startup options to enable and configure the heartbeat mechanism. SIP data tier server start scripts should include startup options to enable heartbeats and start the `WlssEchoServer` process.

Backup each distinct start script used to boot engine tier, SIP data tier, or diameter relay servers in your domain.

Backing Up Logging Servlet Applications

If you use Converged Application Server logging Servlets (see Chapter 17, "Logging SIP Requests and Responses") to perform regular logging or auditing of SIP messages, backup the complete application source files so that you can easily redeploy the applications should the staging server fail or the original deployment directory becomes corrupted.

Backing Up Security Data

The WebLogic Security service stores its configuration data **config.xml** file, and also in an LDAP repository and other files.

Backing Up the WebLogic LDAP Repository

The default Authentication, Authorization, Role Mapper, and Credential Mapper providers that are installed with Converged Application Server store their data in an LDAP server. Each Converged Application Server contains an embedded LDAP server. The Administration Server contains the master LDAP server, which is replicated on all Managed Servers. If any of your security realms use these installed providers, you should maintain an up-to-date backup of the following directory tree:

DOMAIN_NAME`\servers\AdminServer\data\ldap` where ***DOMAIN_HOME*** is the domain's root directory and **AdminServer\data\ldap** is the directory in which the Administration Server stores runtime and security data.

Each Converged Application Server has an LDAP directory, but you only need to back up the LDAP data on the Administration Server—the master LDAP server replicates the LDAP data from each Managed Server when updates to security data are made. WebLogic security providers cannot modify security data while the domain's Administration Server is unavailable. The LDAP repositories on Managed Servers are replicas and cannot be modified.

The **ldap\ldapfiles** subdirectory contains the data files for the LDAP server. The files in this directory contain user, group, group membership, policies, and role information. Other subdirectories under the `ldap` directory contain LDAP server message logs and data about replicated LDAP servers.

Do not update the configuration of a security provider while a backup of LDAP data is in progress. If a change is made—for instance, if an administrator adds a user—while you are backing up the **ldap** directory tree, the backups in the **ldapfiles** subdirectory could become inconsistent. If this does occur, consistent, but potentially out-of-date, LDAP backups are available.

Once a day, a server suspends write operations and creates its own backup of the LDAP data. It archives this backup in a ZIP file below the **ldap\backup** directory and then resumes write operations. This backup is guaranteed to be consistent, but it might not contain the latest security data.

For information about configuring the LDAP backup, see the discussion on backing up the LDAP repository in *Avoiding and Recovering From Server Failure* in the Oracle WebLogic Server 11*g* Documentation.

Backing Up SerializedSystemIni.dat and Security Certificates

All servers create a file named **SerializedSystemIni.dat** and place it in the server's root directory. This file contains encrypted security data that must be present to boot the server. You must back up this file.

If you configured a server to use SSL, also back up the security certificates and keys. The location of these files is user-configurable.

Backing Up Additional Operating System Configuration Files

Certain files maintained at the operating system level are also critical in helping you recover from system failures. Consider backing up the following information as necessary for your system:

- Load Balancer configuration scripts. For example, any automated scripts used to configure load balancer pools and virtual IP addresses for the engine tier cluster, as well as NAT configuration settings.
- NTP client configuration scripts used to synchronize the system clocks of engine and SIP data tier servers.
- Host configuration files for each Converged Application Server machine (host names, virtual and real IP addresses for multi-homed machines, IP routing table information).

Restarting a Failed Administration Server

If an Administration Server fails, only configuration, deployment, and monitoring features are affected, but Managed Servers continue to operate and process client requests. Potential losses incurred due to an Administration Server failure include:

- Loss of in-progress management and deployment operations.
- Loss of ongoing logging functionality.
- Loss of SNMP trap generation for WebLogic Server instances (as opposed to Converged Application Server instances). On Managed Servers, Converged Application Server traps are generated even in the absence of the Administration Server.

To resume normal management activities, restart the failed Administration Server instance as soon as possible.

When you restart a failed Administration Server, no special steps are required. Start the Administration Server as you normally would.

If the Administration Server shuts down while Managed Servers continue to run, you do not need to restart the Managed Servers that are already running in order to recover management of the domain. The procedure for recovering management of an active domain depends upon whether you can restart the Administration Server on the same machine it was running on when the domain was started.

Restarting an Administration Server on the Same Machine

If you restart the WebLogic Administration Server while Managed Servers continue to run, by default the Administration Server can discover the presence of the running Managed Servers.

Note: Make sure that the startup command or startup script does not include `-Dweblogic.management.discover=false`, which disables an Administration Server from discovering its running Managed Servers.

The root directory for the domain contains a file, **running-managed-servers.xml**, which contains a list of the Managed Servers in the domain and describes whether they are running or not. When the Administration Server restarts, it checks this file to determine which Managed Servers were under its control before it stopped running.

When a Managed Server is gracefully or forcefully shut down, its status in **running-managed-servers.xml** is updated to "not-running." When an Administration Server restarts, it does not try to discover Managed Servers with the "not-running" status. A Managed Server that stops running because of a system crash, or that was stopped by killing the JVM or the command prompt (shell) in which it was running, will still have the status "running' in **running-managed-servers.xml**. The Administration Server will attempt to discover them, and will throw an exception when it determines that the Managed Server is no longer running.

Restarting the Administration Server does not cause Managed Servers to update the configuration of static attributes. *Static attributes* are those that a server refers to only during its startup process. Servers instances must be restarted to take account of changes to static configuration attributes. Discovery of the Managed Servers only enables the Administration Server to monitor the Managed Servers or make runtime changes in attributes that can be configured while a server is running (dynamic attributes).

Restarting an Administration Server on Another Machine

If a machine crash prevents you from restarting the Administration Server on the same machine, you can recover management of the running Managed Servers as follows:

1. Install the Converged Application Server software on the new administration machine (if this has not already been done).
2. Make your application files available to the new Administration Server by copying them from backups or by using a shared disk. Your application files should be available in the same relative location on the new file system as on the file system of the original Administration Server.
3. Make your configuration and security data available to the new administration machine by copying them from backups or by using a shared disk. For more information, refer to "Storing the Domain Configuration Offline" and "Backing Up Security Data".
4. Restart the Administration Server on the new machine.

   Make sure that the startup command or startup script does not include `-Dweblogic.management.discover=false`, which disables an Administration Server from discovering its running Managed Servers.

When the Administration Server starts, it communicates with the Managed Servers and informs them that the Administration Server is now running on a different IP address.

Restarting Failed Managed Servers

If the machine on which the failed Managed Server runs can contact the Administration Server for the domain, simply restart the Managed Server manually or automatically using Node Manager. Note that you must configure Node Manager and the Managed Server to support automated restarts, as described in the discussion on using Node Manager to start Managed Servers in a Domain or Cluster in the *Oracle WebLogic Server 11g release 1 patch set 2* documentation.

If the Managed Server cannot connect to the Administration Server during startup, it can retrieve its configuration by reading locally-cached configuration data. A Managed Server that starts in this way is running in Managed Server Independence (MSI) mode. For a description of MSI mode, and the files that a Managed Server must access to start up in MSI mode, see "Replicate domain config files for Managed Server independence" in the Administration Console online Help.

To start up a Managed Server in MSI mode:

1. Ensure that the following files are available in the Managed Server's root directory:

   – **msi-config.xml**

   – **SerializedSystemIni.dat**

   – **boot.properties**

   If these files are not in the Managed Server's root directory:

   a. Copy the **config.xml** and **SerializedSystemIni.dat** file from the Administration Server's root directory (or from a backup) to the Managed Server's root directory.

   b. Rename the configuration file to **msi-config.xml**. When you start the server, it will use the copied configuration files.

Note: Alternatively, use the `-Dweblogic.RootDirectory=path` startup option to specify a root directory that already contains these files.

2. Start the Managed Server at the command line or using a script.

   The Managed Server will run in MSI mode until it is contacted by its Administration Server. For information about restarting the Administration Server in this scenario, see "Restarting a Failed Administration Server".

19

Tuning JVM Garbage Collection for Production Deployments

This chapter describes how to tune Java Virtual Machine (JVM) garbage collection performance for engine tier servers:

- Goals for Tuning Garbage Collection Performance
- Modifying JVM Parameters in Server Start Scripts
- Tuning Garbage Collection with JRockit
- Tuning Garbage Collection with Sun JDK

Goals for Tuning Garbage Collection Performance

Production installations of Oracle Communications Converged Application Server generally require extremely small response times (under 50 milliseconds) for clients at all times, even under peak server loads. A key factor in maintaining brief response times is the proper selection and tuning of the JVM's Garbage Collection (GC) algorithm for Converged Application Server instances in the engine tier.

Whereas certain tuning strategies are designed to yield the lowest average garbage collection times or to minimize the frequency of full GCs, those strategies can sometimes result in one or more very long periods of garbage collection (often several seconds long) that are offset by shorter GC intervals. With a production SIP Server installation, all long GC intervals must be avoided in order to maintain response time goals.

The sections that follow describe GC tuning strategies for JRockit and Sun's JVM that generally result in best response time performance.

Note: For more information on JRockit, see *Introduction to Oracle WebLogic Server* in the WebLogic Server documentation.

Modifying JVM Parameters in Server Start Scripts

If you use custom startup scripts to start Converged Application Server engines and replicas, simply edit those scripts to include the recommended JVM options described in the sections that follow.

The Configuration Wizard also installs default startup scripts when you configure a new domain. by default, these scripts are installed in the `*CAS50_home*/user_projects/domains/*domain_name*/bin` directory, where *`CAS50_home`* is where you installed the Converged Application Server software and`domain_name` is the name of the domain's directory. The `/bin` directory includes:

- `startWebLogic.cmd`, `startWebLogic.sh`: These scripts start the Administration Server for the domain.
- `startManagedWebLogic.cmd`, `startManagedWebLogic.sh`: These scripts start managed engines and replicas in the domain.

If you use the Oracle-installed scripts to start engines and replicas, you can override JVM memory arguments by first setting the `USER_MEM_ARGS` environment variable in your command shell.

Note: Setting the `USER_MEM_ARGS` environment variable overrides all default JVM memory arguments specified in the Oracle-installed scripts. Always set `USER_MEM_ARGS` to the full list of JVM memory arguments you intend to use. For example, when using the Sun JVM, always add `-XX:MaxPermSize=128m` to the `USER_MEM_ARGS` value, even if you only intend to change the default heap space (`-Xms, -Xmx`) parameters.

Tuning Garbage Collection with JRockit

JRockit provides several monitoring tools that you can use to analyze the JVM heap at any given moment, including:

- JRockit Runtime Analyzer: provides a view into the runtime behavior of garbage collection and pause times.
- JRockit Stack Dumps: reveals applications' thread activity to help you troubleshoot and/or improve performance.

Use these and other tools in a controlled environment to determine the effects of JVM settings before you use the settings in a production deployment.

The following sections describe suggested starting JVM options for use with the JRockit. If you use JRockit with the deterministic garbage collector (recommended), use the options described in "Using Oracle JRockit Real Time (Deterministic Garbage Collection)".

Using Oracle JRockit Real Time (Deterministic Garbage Collection)

Very short response times are most easily achieved by using JRockit Real Time, which implements a deterministic garbage collector.

Oracle recommends using the following JVM arguments for engine tier servers in replicated cluster configurations:

```
-Xms1024m -Xmx1024m -XgcPrio:deterministic -XpauseTarget=30ms -XXtlasize:min=8k
-XXnosystemgc
```

Note: The above settings are configured by default in the `$WL_HOME/common/bin/wlssCommenv.sh` file when you use the Configuration Wizard to create a new domain with the JRockit JVM.

You may need to increase the `-XpauseTarget` value for allocation-intensive applications. The value can be decreased for smaller applications under light loads.

Adjust the heap size according to the amount of live data used by deployed applications. As a starting point, set the heap size from 2 to 3 times the amount required by your applications. A value closer to 3 times the required amount generally yields the best performance.

For replica servers, increase the available memory:

```
-Xms3072m -Xmx3072m -XgcPrio:deterministic -XpauseTarget=30ms -XXtlasize:min=8k
-XXnosystemgc
```

These settings fix the heap size and enable the dynamic garbage collector with deterministic garbage collection. `-XpauseTarget` sets the maximum pause time and `-XXtlasize=3k` sets the thread-local area size. `-XXnosystemgc` prevents `System.gc()` application calls from forcing garbage collection.

Using Oracle JRockit without Deterministic Garbage Collection

When using Oracle's JRockit JVM without deterministic garbage collection (not recommended for production deployments), the best response time performance is obtained by using the generational concurrent garbage collector.

The full list of example startup options for an engine tier server are:

```
-Xms1024m -Xmx1024m -Xgc:gencon -XXnosystemgc -XXtlasize:min=3k -XXkeeparearatio=0
-Xns:48m
```

Note: Fine tune the heap size according to the amount of live data used by deployed applications.

The full list of example startup options for a replica server are:

```
-Xms3072m -Xmx3072m -Xgc:gencon -XXnosystemgc -XXtlasize:min=3k -XXkeeparearatio=0
-Xns:48m
```

Tuning Garbage Collection with Sun JDK

When using Sun's JDK, the goal in tuning garbage collection performance is to reduce the time required to perform a full garbage collection cycle. You should not attempt to tune the JVM to minimize the frequency of full garbage collections, because this generally results in an eventual forced garbage collection cycle that may take up to several full seconds to complete.

The simplest and most reliable way to achieve short garbage collection times over the lifetime of a production server is to use a fixed heap size with the default collector and the parallel young generation collector, restricting the new generation size to at most one third of the overall heap.

The following example JVM settings are recommended for most engine tier servers:

```
-server -Xmx1024m -XX:MaxPermSize=128m -XX:+UseParNewGC -XX:+UseConcMarkSweepGC
        -XX:+UseTLAB -XX:+CMSIncrementalMode -XX:+CMSIncrementalPacing
        -XX:CMSIncrementalDutyCycleMin=0 -XX:CMSIncrementalDutyCycle=10
        -XX:MaxTenuringThreshold=0 -XX:SurvivorRatio=256
        -XX:CMSInitiatingOccupancyFraction=60 -XX:+DisableExplicitGC
```

For replica servers, use the example settings:

```
-server -Xmx3072m -XX:MaxPermSize=128m -XX:+UseParNewGC -XX:+UseConcMarkSweepGC
        -XX:+UseTLAB -XX:+CMSIncrementalMode -XX:+CMSIncrementalPacing
        -XX:CMSIncrementalDutyCycleMin=0 -XX:CMSIncrementalDutyCycle=10
        -XX:MaxTenuringThreshold=0 -XX:SurvivorRatio=256
        -XX:CMSInitiatingOccupancyFraction=60 -XX:+DisableExplicitGC
```

The above options have the following effect:

- `-XX:+UseTLAB`: Uses thread-local object allocation blocks. This improves concurrency by reducing contention on the shared heap lock.
- `-XX:+UseParNewGC`: Uses a parallel version of the young generation copying collector alongside the concurrent mark-and-sweep collector. This minimizes pauses by using all available CPUs in parallel. The collector is compatible with both the default collector and the Concurrent Mark and Sweep (CMS) collector.
- `-Xms, -Xmx`: Places boundaries on the heap size to increase the predictability of garbage collection. The heap size is limited in replica servers so that even Full GCs do not trigger SIP retransmissions. `-Xms` sets the starting size to prevent pauses caused by heap expansion.
- `-XX:MaxTenuringThreshold=0`: Makes the full NewSize available to every NewGC cycle, and reduces the pause time by not evaluating tenured objects. Technically, this setting promotes all live objects to the older generation, rather than copying them.
- `-XX:SurvivorRatio=128`: Specifies a high survivor ratio, which goes along with the zero tenuring threshold to ensure that little space is reserved for absent survivors.

20 Avoiding JVM Delays Caused By Random Number Generation

This chapter describes how to avoid Java Virtual Machine (JVM) delays in Oracle Communications Converged Application Server processes for delays caused by random number generation.

Avoiding JVM Delays Caused by Random Number Generation

The library used for random number generation in Sun's JVM relies on `/dev/random` by default for UNIX platforms. This can potentially block the Converged Application Server process because on some operating systems `/dev/random` waits for a certain amount of "noise" to be generated on the host machine before returning a result. Although `/dev/random` is more secure, Oracle recommends using `/dev/urandom` if the default JVM configuration delays Converged Application Server startup.

To determine if your operating system exhibits this behavior, try displaying a portion of the file from a shell prompt:

```
head -n 1 /dev/random
```

If the command returns immediately, you can use `/dev/random` as the default generator for SUN's JVM. If the command does not return immediately, use these steps to configure the JVM to use `/dev/urandom`:

1. Open the `$JAVA_HOME/jre/lib/security/java.security` file in a text editor.
2. Change the line:

   ```
   securerandom.source=file:/dev/random
   ```

   to read:

   ```
   securerandom.source=file:/dev/urandom
   ```

3. Save your change and exit the text editor.

Part IV

Reference

This part provides reference information on Oracle Communications Converged Application Server XML configuration files and their entries. It also provides a list of startup configuration options.

This part contains the following chapters:

21 Engine Tier Configuration Reference (sipserver.xml)

This chapter describes the engine tier configuration file, **sipserver.xml**:

- Overview of sipserver.xml
- Editing sipserver.xml
- XML Schema
- Example sipserver.xml File
- XML Element Description

Overview of sipserver.xml

The **sipserver.xml** file is an XML document that configures the SIP container features provided by an Oracle Communications Converged Application Server instance in the engine tier of a server installation. **sipserver.xml** is stored in the ***CAS_Home* /config/custom** subdirectory where ***CAS_Home*** is the root directory of the Converged Application Server domain.

Editing sipserver.xml

You should never move, modify, or delete the **sipserver.xml** file during normal operations.

Oracle recommends using the Administration Console to modify **sipserver.xml** indirectly, rather than editing the file manually with a text editor. Using the Administration Console ensures that the **sipserver.xml** document always contains valid XML.

You may need to manually view or edit **sipserver.xml** to troubleshoot problem configurations, repair corrupted files, or to roll out custom configurations to a large number of machines when installing or upgrading Converged Application Server. When you manually edit **sipserver.xml**, you must reboot Converged Application Server instances to apply your changes.

Caution: Always use the SipServer node in the Administration Console or the WLST utility to make changes to a running Converged Application Server deployment. See Chapter 3, "Configuring Converged Application Container Properties."

Steps for Editing sipserver.xml

If you need to modify **sipserver.xml** on a production system, follow these steps:

1. Use a text editor to open the ***CAS_home*/config/custom/sipserver.xml** file, where ***CAS_home*** is the root directory of the Converged Application Server domain.
2. Modify the **sipserver.xml** file as necessary. See "XML Schema" for a full description of the XML elements.
3. Save your changes and exit the text editor.
4. Reboot or start servers to have your changes take effect:

   Caution: Always use the SipServer node in the Administration Console or the WLST utility to make changes to a running Converged Application Server deployment. See Chapter 3, "Configuring Converged Application Container Properties" for more information.

5. Test the updated system to validate the configuration.

XML Schema

The schema file for **sipserver.xml** (**wcp-sipserver.xsd**) is installed inside the **wlss-descriptor-binding.jar** library, located in ***WL_home*/sip/server/lib**, where *WL_home* is the path to the directory where WebLogic Server is installed.

Example sipserver.xml File

The following shows a simple example of a **sipserver.xml** file:

```
<?xml version="1.0" encoding="UTF-8"?>
<sip-server xmlns="http://www.bea.com/ns/wlcp/wlss/300">
  <overload>
    <threshold-policy>queue-length</threshold-policy>
    <threshold-value>200</threshold-value>
    <release-value>150</release-value>
  </overload>
</sip-server>
```

XML Element Description

The following sections describe each element used in the **sipserver.xml** configuration file. Each section describes an XML element that is contained within the main `sip-server` element.

enable-timer-affinity

The `enable-timer-affinity` element determines the way in which engine tier servers process expired timers. By default (when `enable-timer-affinity` is omitted from **sipserver.xml**, or is set to "false"), an engine tier server that polls the SIP data tier for expired timers processes all available expired timers. When `enable-timer-affinity` is set to "true," engine tier servers polling the SIP data tier process only those expired timers that are associated with call states that the engine last modified (or expired timers for call states that have no owner).

See "Configuring Timer Processing" in Chapter 3, "Configuring Converged Application Container Properties," for more information.

overload

The `overload` element enables you to throttle incoming SIP requests according to a configured overload condition. When an overload condition occurs, Converged Application Server destroys new SIP requests by responding with "503 Service Unavailable" until the configured release value is observed, or until the size of the server's capacity constraints is reduced (see "Overload Control Based on Capacity Constraints").

User-configured overload controls are applied only to initial SIP requests; SIP dialogues that are already active when an overload condition occurs may generate additional SIP requests that are not throttled.

To configure an overload control, you define the three elements described in Table 21–1.

Table 21–1 Nested overload Elements

Element	Description
`threshold-policy`	A String value that identifies the type of measurement used to monitor overload conditions: ▪ `session-rate` measures the rate at which new SIP requests are generated. Converged Application Server determines the session rate by calculating the number of new SIP application connections that were created in the last 5 seconds of operation. See "Overload Control Based on Session Generation Rate". ▪ `queue-length` measures the sum of the sizes of the capacity constraint work manager components that processes SIP requests and SIP timers. See "Overload Control Based on Capacity Constraints". ▪ **Note:** Execute queues are deprecated and no longer used in `Converged Application Server. Capacity constraints are used in place of execute queues. The policy name "queue-length"` was kept for backward compatibility. You must use only one of the above policies to define an overload control. See "Selecting an Appropriate Overload Policy" for more information.

Table 21–1 (Cont.) Nested overload Elements

Element	Description
`threshold-value`	Specifies the measured value that causes Converged Application Server to recognize an overload condition and *start* throttling new SIP requests: ■ When using the `session-rate` threshold policy, `threshold-value` specifies the number of new SIP requests per second that trigger an overload condition. See "Overload Control Based on Session Generation Rate". ■ When using the `queue-length` threshold policy, `threshold-value` specifies the size of the combined number of requests in the SIP transport and SIP timer capacity constraint components that triggers an overload condition. See "Overload Control Based on Capacity Constraints". ■ After the `threshold-value` is observed, Converged Application Server recognizes an overload condition for a minimum of 512 milliseconds during which time new SIP requests are throttled. If multiple overloads occur over a short period of time, the minimum overload of 512 ms is dynamically increased to avoid repeated overloads. ■ After the minimum overload recognition period expires, the overload condition is terminated only after the configured `release-value` is observed.
`release-value`	Specifies the measured value that causes Converged Application Server to end an overload condition and *stop* throttling new SIP requests: ■ When using the `session-rate` threshold policy, `release-value` specifies the number of new SIP requests per second that terminates session throttling. See "Overload Control Based on Session Generation Rate". ■ When using the `queue-length` threshold policy, `release-value` specifies the combined number of requests in the capacity constraints that terminates session throttling. See "Overload Control Based on Capacity Constraints".

Selecting an Appropriate Overload Policy

Converged Application Server provides two different policies for throttling SIP requests:

- The `session-rate` policy throttles sessions when the volume new SIP sessions reaches a configured rate (a specified number of sessions per second).
- The `queue-length` policy throttles requests after the sum of the requests in the `wlss.trasnport` work manager and `wlss.timer.capacity` capacity constraint components reaches a configured size.

Note that you must select only one of the available overload policies. You cannot use both policies simultaneously.

The `session-rate` policy is generally used when a back-end resource having a known maximum throughput (for example, an RDBMS) is used to set up SIP calls. In this case, the `session-rate` policy enables you to tie the Converged Application Server overload policy to the known throughput capabilities of the back-end resource.

With the `queue-length` policy, Converged Application Server monitors both CPU and I/O bottlenecks to diagnose an overload condition. The `queue-length` policy is generally used with CPU-intensive SIP applications in systems that have no predictable upper bound associated with the call rate.

The following sections describe each policy in detail.

Overload Control Based on Session Generation Rate

Converged Application Server calculates the session generation rate (sessions per second) by monitoring the number of application sessions created in the last 5 seconds. When the session generation rate exceeds the rate specified in the `threshold-value` element, Converged Application Server throttles initial SIP requests until the session generation rate becomes smaller than the configured `release-value`.

The following example configures Converged Application Server to begin throttling SIP requests when the new sessions are created at a rate higher than 50 sessions per second. Throttling is discontinued when the session rate drops to 40 sessions per second:

```
<overload>
  <threshold-policy>session-rate</threshold-policy>
  <threshold-value>50</threshold-value>
  <release-value>40</release-value>
</overload>
```

Overload Control Based on Capacity Constraints

By default, SIP messages are handled by a work manager named `wlss.transport` and SIP timers are processed by a work manager named `wlss.timer`. Each work manager has an associated capacity constraint component that sets the number of requests allotted for SIP message handling and timer processing. Work managers are configured in the **config.xml** file for your Converged Application Server. Work managers allocate threads automatically, as described in the Oracle WebLogic Server documentation. You can also allocate additional threads to the server at boot time using the startup option `-Dweblogic.threadpool.MinPoolSize=number_of_threads`.

Converged Application Server performs `queue-length` overload control by monitoring the combined lengths of the configured capacity constraints. When the sum of the requests in the two constraints exceeds the length specified in the `threshold-value` element, Converged Application Server throttles initial SIP requests until the total requests are reduced to the configured `release-value`.

Example 21–1 shows a sample `overload` configuration from **sipserver.xml**. Here, Converged Application Server begins throttling SIP requests when the combined size of the constraints exceeds 200 requests. Throttling is discontinued when the combined length returns to 200 or fewer simultaneous requests.

Example 21–1 Sample overload Definition

```
<overload>
  <threshold-policy>queue-length</threshold-policy>
  <threshold-value>200</threshold-value>
  <release-value>150</release-value>
</overload>
```

Two Levels of Overload Protection

User-configured overload controls (defined in **sipserver.xml**) represent the first level of overload protection provided by Converged Application Server. They mark the onset of an overload condition and initiate simple measures to avoid dropped calls (generating 503 responses for new requests).

If the condition that caused the overload persists or worsens, then the work manager component used to perform work in the SIP Servlet container may itself become overloaded. At this point, the server no longer utilizes threads to generate 503 responses, but instead begins to drop messages. In this way, the configured size of the SIP container's work manager components represent the second and final level of overload protection employed by the server.

Always configure overload controls in **sipserver.xml** conservatively, and resolve the circumstances that caused the overload in a timely fashion.

message-debug

The `message-debug` element is used to enable and configure access logging with log rotation for Converged Application Server. This element should be used only in a development environment, because access logging logs *all* SIP requests and responses. See "Enabling Access Logging" in the *Converged Application Server Application Developer's Guide* for information about configuring and using access logging.

If you want to perform more selective logging in a production environment, see Chapter 17, "Logging SIP Requests and Responses."

proxy—Setting Up an Outbound Proxy Server

RFC 3261 defines an outbound proxy as "A proxy that receives requests from a client, even though it may not be the server resolved by the Request-URI. Typically, a UA is manually configured with an outbound proxy, or can learn about one through auto-configuration protocols."

In Converged Application Server an outbound proxy server is specified using the `proxy` element in **sipserver.xml**. The proxy element defines one or more proxy server URIs. You can change the behavior of the proxy process by setting a proxy policy with the `proxy-policy` tag. Table 21–2, " Nested proxy Elements" describes the possible values for the `proxy` elements.

The default behavior is as if **proxy** policy is in effect. The **proxy** policy means that the request is sent out to the configured outbound Proxy and the Route headers in the request preserve any routing decision taken by Converged Application Server. This enables the outbound proxy to send the request over to the intended recipient after it has performed its actions on the request. The **proxy** policy comes into effect only for the initial requests. As for the subsequent request the Route Set takes precedence over any policy in a dialog. (If the outbound proxy wants to be in the Route Set it can turn record routing on).

Also if a proxy application written on Converged Application Server wishes to override the configured behavior of outbound proxy traversal, then it can add a special header with name X-BEA-Proxy-Policy with the value `domain`. This header is stripped from the request while sending, but the effect is to ignore the configured outbound proxy. The X-BEA-Proxy-Policy custom header can be used by applications to override the configured policy on a request-by-request basis. The value of the header can be `domain` or `proxy`. Note, however, that if the policy is overridden to `proxy`, the configuration must still have the outbound proxy URIs in order to route to the outbound proxy.

Table 21–2 Nested proxy Elements

Element	Description
`routing-policy`	An optional element that configures the behavior of the proxy. Valid values are: ▪ **domain** - Proxies messages using the routing rule defined by RFC 3261, ignoring any outbound proxy that is specified. ▪ **proxy** - Sends the message to the downstream proxy specified in the default proxy URI. If there are multiple proxy specifications they are tried in the order in which they are specified. However, if the transport tries a UDP proxy, the settings for subsequent proxies are ignored.
`uri`	The TCP or UDP URI of the proxy server. You must specify at least one URI for a `proxy` element. Place multiple URIs in multiple `uri` elements within the `proxy` element.

Example 21–2 shows the default proxy configuration for Converged Application Server domains. The request in this case is created in accordance with the SIP routing rules, and finally the request is sent to the outbound proxy "sipoutbound.oracle.com".

Example 21–2 Sample proxy Definition

```
<proxy>
     <routing-policy>proxy</routing-policy>
     <uri>sip:sipoutbound.oracle.com:5060</uri>
     <!-- Other proxy uri tags can be added. - >
</proxy>
```

t1-timeout-interval

This element sets the value of the SIP protocol T1 timer, in milliseconds. Timer T1 also specifies the initial values of Timers A, E, and G, which control the retransmit interval for INVITE requests and responses over UDP.

Timer T1 also affects the values of timers F, H, and J, which control retransmit intervals for INVITE responses and requests; these timers are set to a value of 64*T1 milliseconds. See the Session Initiation Protocol for more information about SIP timers. See also "Configuring NTP for Accurate SIP Timers" in Chapter 3, "Configuring Converged Application Container Properties."

If `t1-timeout-interval` is not configured, Converged Application Server uses the SIP protocol default value of 500 milliseconds.

t2-timeout-interval

This elements sets the value of the SIP protocol T2 timer, in milliseconds. Timer T2 defines the retransmit interval for INVITE responses and non-INVITE requests. See the Session Initiation Protocol for more information about SIP timers. See also "Configuring NTP for Accurate SIP Timers" in Chapter 3, "Configuring Converged Application Container Properties."

If `t2-timeout-interval` is not configured, Converged Application Server uses the SIP protocol default value of 4 seconds.

t4-timeout-interval

This elements sets the value of the SIP protocol T4 timer, in milliseconds. Timer T4 specifies the maximum length of time that a message remains in the network. Timer T4 also specifies the initial values of Timers I and K, which control the wait times for retransmitting ACKs and responses over UDP. See the Session Initiation Protocol for more information about SIP timers. See also "Configuring NTP for Accurate SIP Timers" in Chapter 3, "Configuring Converged Application Container Properties."

If `t4-timeout-interval` is not configured, Converged Application Server uses the SIP protocol default value of 5 seconds.

timer-b-timeout-interval

This elements sets the value of the SIP protocol Timer B, in milliseconds. Timer B specifies the length of time a client transaction attempts to retry sending a request. See the Session Initiation Protocol for more information about SIP timers. See also "Configuring NTP for Accurate SIP Timers" in Chapter 3, "Configuring Converged Application Container Properties."

If `timer-b-timeout-interval` is not configured, the Timer B value is derived from timer T1 (64*T1, or 32000 milliseconds by default).

timer-f-timeout-interval

This elements sets the value of the SIP protocol Timer F, in milliseconds. Timer F specifies the timeout interval for retransmitting non-INVITE requests. See the Session Initiation Protocol for more information about SIP timers. See also "Configuring NTP for Accurate SIP Timers" in Chapter 3, "Configuring Converged Application Container Properties."

If `timer-f-timeout-interval` is not configured, the Timer F value is derived from timer T1 (64*T1, or 32000 milliseconds by default).

max-application-session-lifetime

This element sets the maximum amount of time, in minutes, that a SIP application session can exist before Converged Application Server invalidates the session. `max-application-session-lifetime` acts as an upper bound for any timeout value specified using the `session-timeout` element in a **sip.xml** file, or using the `setExpires` API.

A value of -1 (the default) specifies that there is no upper bound to application-configured timeout values.

enable-local-dispatch

`enable-local-dispatch` is a server optimization that helps avoid unnecessary network traffic when sending and forwarding messages. You enable the optimization by setting this element "true." When `enable-local-dispatch` enabled, if a server instance needs to send or forward a message and the message destination is the engine tier's cluster address or the local server address, then the message is routed internally to the local server instead of being sent via the network. Using this optimization can dramatically improve performance when chained applications process the same request as described in "Composing SIP Applications" in the *Converged Application Server Application Developer's Guide*.

You may want to disable this optimization if you feel that routing internal messages could skew the load on servers in the engine tier, and you prefer to route all requests via a configured load balancer.

By default `enable-local-dispatch` is set to "false."

cluster-loadbalancer-map

The `cluster-loadbalancer-map` element is used only when upgrading Converged Application Server software, or when upgrading a production SIP Servlet to a new version. It is not required or used during normal server operations.

During a software upgrade, multiple engine tier clusters are defined to host the older and newer software versions. `A cluster-loadbalancer-map` defines the virtual IP address (defined on your load balancer) that correspond to an engine tier cluster configured for an upgrade. Converged Application Server uses this mapping to ensure that engine tier requests for timers and call state data are received from the correct "version" of the cluster. If a request comes from an incorrect version of the software, the `cluster-loadbalancer-map` entries are used to forward the request to the correct cluster.

Each `cluster-loadbalancer-map` entry contains the two elements described in Table 21–3.

Table 21–3 Nested cluster-loadbalancer-map Elements

Element	Description
`cluster-name`	The configured name of an engine tier cluster.
`sip-uri`	The internal SIP URI that maps to the engine tier cluster. This corresponds to a virtual IP address that you have configured in your load balancer. The internal URI is used to forward requests to the correct cluster version during an upgrade.

Example 21–3 shows a sample `cluster-loadbalancer-map` entry used during an upgrade.

Example 21–3 Sample cluster-loadbalancer-map Entry

```
<cluster-loadbalancer-map>
   <cluster-name>EngineCluster</cluster-name>
   <sip-uri>sip:172.17.0.1:5060</sip-uri>
</cluster-loadbalancer-map>
<cluster-loadbalancer-map>
   <cluster-name>EngineCluster2</cluster-name>
   <sip-uri>sip:172.17.0.2:5060</sip-uri>
</cluster-loadbalancer-map>
```

See Chapter 10, "Upgrading Production Converged Application Server Software," for more information.

default-behavior

This element defines the default behavior of the Converged Application Server instance if the server cannot match an incoming SIP request to a deployed SIP Servlet (or if the matching application has been invalidated or timed out). Valid values are:

- `proxy`: Act as a proxy server.
- `ua`: Act as a User Agent.

`proxy` is used as the default if you do not specify a value.

When acting as a User Agent (UA), Converged Application Server acts in the following way in response to SIP requests:

- ACK requests are discarded without notice.

- CANCEL or BYE requests receive response code 481 - Transaction does not exist.
- All other requests receive response code 500 - Internal server error.

When acting as a proxy requests are automatically forwarded to an outbound proxy (see "proxy—Setting Up an Outbound Proxy Server") if one is configured. If no proxy is defined, Converged Application Server proxies to a specified Request URI only if the Request URI does not match the IP and port number of a known local address for a SIP Servlet container, or a load balancer address configured for the server. This ensures that the request does not constantly loop to the same servers. When the Request URI matches a local container address or load balancer address, Converged Application Server instead acts as a UA.

default-servlet-name

This element specifies the name of a default SIP Servlet to call if an incoming initial request cannot be matched to a deployed Servlet (using standard `servlet-mapping` definitions in **sip.xml**). The name specified in the `default-servlet-name` element must match the `servlet-name` value of a deployed SIP Servlet. For example:

```
<default-servlet-name>myServlet</default-servlet-name>
```

If the name defined in `default-servlet-name` does not match a deployed Servlet, or no value is supplied (the default configuration), Converged Application Server registers the name `com.bea.wcp.sip.engine.BlankServlet` as the default Servlet. The `BlankServlet` name is also used if a deployed Servlet registered as the `default-servlet-name` is undeployed from the container.

`BlankServlet`'s behavior is configured with the `default-behavior` element. By default the Servlet proxies all unmatched requests. However, if the `default-behavior` element is set to "ua" mode, `BlankServlet` is responsible for returning 481 responses for CANCEL and BYE requests, and 500/416 responses in all other cases. `BlankServlet` does not respond to ACK, and it always invalidates the application session.

retry-after-value

Specifies the number of seconds used in the `Retry-After` header for 5xx response codes. This value can also include a parameter or a reason code, such as "Retry-After: 18000;duration=3600" or "Retry-After: 120 (I'm in a meeting)."

If the this value is not configured, Converged Application Server uses the default value of 180 seconds.

sip-security

Converged Application Server enables you to configure one or more trusted hosts for which authentication is not performed. When Converged Application Server receives a SIP message, it calls `getRemoteAddress()` on the SIP Servlet message. If this address matches an address defined in the server's trusted host list, no further authentication is performed for the message.

The `sip-security` element defines one or more trusted hosts, for which authentication is not performed. The `sip-security` element contains one or more `trusted-authentication-host` or `trusted-charging-host` elements, each of which contains a trusted host definition. A trusted host definition can consist of an IP address (with or without wildcard placeholders) or a DNS name. Example 21–4 shows a sample `sip-security` configuration.

Example 21–4 Sample Trusted Host Configuration

```
<sip-security>

<trusted-authentication-host>myhost1.mycompany.com</trusted-authentication-host>
   <trusted-authentication-host>172.*</trusted-authentication-host>
</sip-security>
```

route-header

3GPP TS 24.229 Version 7.0.0 :

`http://www.3gpp.org/ftp/Specs/archive/24_series/24.229/24229-700.zip` requires that IMS Application Servers generating new requests (for example, as a B2BUA) include the S-CSCF route header. In Converged Application Server, the S-CSCF route header must be statically defined as the value of the `route-header` element in **sipserver.xml**. For example:

```
<route-header>
   <uri>Route: sip:wlss1.bea.com</uri>
</route-header>
```

engine-call-state-cache-enabled

Converged Application Server provides the option for engine tier servers to cache a portion of the call state data locally, as well as in the SIP data tier, to improve performance with SIP-aware load balancers. When a local cache is used, an engine tier server first checks its local cache for existing call state data. If the cache contains the required data, and the local copy of the data is up-to-date (compared to the SIP data tier copy), the engine locks the call state in the SIP data tier but reads directly from its cache.

By default the engine tier cache is enabled. To disable caching, set `engine-call-state-cache-enabled` to false:

```
<engine-call-state-cache-enabled>false</engine-call-state-cache-enabled>
```

See Chapter 7, "Using the Engine Tier Cache," for more information.

server-header

Converged Application Server enables you to control when a Server header is inserted into SIP messages. You can use this functionality to limit or eliminate Server headers to reduce the message size for wireless networks, or to increase security.

By default, Converged Application Server inserts no Server header into SIP messages. Set the `server-header` to one of the following string values to configure this behavior:

- `none` (the default) inserts no Server header.
- `request` inserts the Server header only for SIP requests generated by the server.
- `response` inserts the Server header only for SIP responses generated by the server.
- `all` inserts the Server header for all SIP requests and responses.

For example, the following element configures Converged Application Server to insert a Server header for all generated SIP messages:

```
<server-header>all</server-header>
```

See also "server-header-value".

server-header-value

Converged Application Server enables you to control the text that is inserted into the Server header of generated messages. This provides additional control over the size of SIP messages and also enables you to mask the server entity for security purposes. By default, Converged Application Server does not insert a Server header into generated SIP messages (see "server-header"). If Server header insertion is enabled but no `server-header-value` is specified, Converged Application Server inserts the value "WebLogic SIP Server." To configure the header contents, enter a string value. For example:

```
<server-header-value>MyCompany Application Server</server-header-value>
```

persistence

The `persistence` element defines enables or disables writing call state data to an RDBMS and/or to a remote, geographically-redundant Converged Application Server installation. For sites that utilize geographically-redundant replication features, the `persistence` element also defines the site ID and the URL at which to persist call state data.

The `persistence` element contains the sub-elements described in Table 21–4.

Table 21–4 Nested persistence Elements

Element	Description
`default-handling`	Determines whether or not Converged Application Server observes persistence hints for RDBMS persistence and/or geographical-redundancy. This element can have one of the following values: ▪ **all**: Specifies that call state data may be persisted to both an RDBMS store and to a geographically-redundant Converged Application Server installation. This is the default behavior. Note that actual replication to either destination also requires that the available resources (JDBC datasource and remote JMS queue) are available. ▪ **db**: Specifies that long-lived call state data is replicated to an RDBMS if the required JDBC datasource and schema are available. ▪ **geo**: Specifies that call state data is persisted to a remote, geographically-redundant site if the configured site URL contains the necessary JMS resources. ▪ **none**: Specifies that only in-memory replication is performed to other replicas in the SIP data tier cluster. Call state data is not persisted in an RDBMS or to an external site.
`geo-site-id`	Specifies the site ID of this installation. All installations that participate in geographically-redundant replication require a unique site ID.
`geo-remote-t3-url`	Specifies the remote Converged Application Server installation to which this site replicates call state data. You can specify a single URL corresponding to the engine tier cluster of the remote installation. You can also specify a comma-separated list of addresses corresponding to each engine tier server. The URLs must specify the t3 protocol.

Example 21–5 shows a sample configuration that uses RDBMS storage for long-lived call state as well as geographically-redundant replication. Call states are replicated to two engine tier servers in a remote location.

Example 21–5 Sample persistence Configuration

```
<persistence>
```

```
  <default-handling>all</default-handling>
  <geo-site-id>1</geo-site-id>

<geo-remote-t3-url>t3://remoteEngine1:7050,t3://remoteEngine2:7051</geo-remote-t3-
url>
</persistence>
```

See Chapter 8, "Storing Long-Lived Call State Data in an RDBMS" and "Chapter 9, "Configuring Geographically-Redundant Installations" for more information.

use-header-form

This element configures the server-wide, default behavior for using or preserving compact headers in SIP messages. You can set this element to one of the following values:

- `compact`: Converged Application Server uses the compact form for all system-generated headers. However, any headers that are copied from an originating message (rather than generated) use their original form.
- `force compact`: Converged Application Server uses the compact form for all headers, converting long headers in existing messages into compact headers as necessary.
- `long`: Converged Application Server uses the long form for all system-generated headers. However, any headers that are copied from an originating message (rather than generated) use their original form.
- `force long`: Converged Application Server uses the long form for all headers, converting compact headers in existing messages into long headers as necessary.

enable-dns-srv-lookup

This element enables or disables Converged Application Server DNS lookup capabilities. If you set the element to "true," then the server can use DNS to:

- Discover a proxy server's transport, IP address, and port number when a request is sent to a SIP URI.
- Resolve an IP address and/or port number during response routing, depending on the contents of the Sent-by field.

For proxy discovery, Converged Application Server uses DNS resolution only once per SIP transaction to determine transport, IP, and port number information. All retransmissions, ACKs, or CANCEL requests are delivered to the same address and port using the same transport. For details about how DNS resolution takes place, see *RFC 3263: Session Initiation Protocol (SIP): Locating SIP Servers* (`http://www.ietf.org/rfc/rfc3263.txt`).

When a proxy needs to send a response message, Converged Application Server uses DNS lookup to determine the IP address and/or port number of the destination, depending on the information provided in the `sent-by` field and Via header.

By default, DNS resolution is not used ("false").

Note: Because DNS resolution is performed within the context of SIP message processing, any DNS performance problems result in increased latency performance. Oracle recommends using a caching DNS server in a production environment to minimize potential performance problems.

connection-reuse-pool

Converged Application Server includes a connection pooling mechanism that can be used to minimize communication overhead with a Session Border Control (SBC) function or Serving Call Session Control Function (S-CSCF). You can configure multiple, fixed pools of connections to different addresses.

Converged Application Server opens new connections from the connection pool on demand as the server makes requests to a configured address. The server then multiplexes new SIP requests to the address using the already-opened connections, rather than repeatedly terminating and recreating new connections. Opened connections are re-used in a round-robin fashion. Opened connections remain open until they are explicitly closed by the remote address.

Note that connection re-use pools are not used for incoming requests from a configured address.

To configure a connection re-use pool, you define the four nested elements described in Table 21–5.

Table 21–5 Nested connection-reuse-pool Elements

Element	Description
`pool-name`	A String value that identifies the name of this pool. All configured `pool-name` elements must be unique to the domain.
`destination`	Specifies the IP address or host name of the destination SBC or S-CSCF. Converged Application Server opens or re-uses connection in this pool only when making requests to the configured address.
`destination-port`	Specifies the port number of the destination SBC or S-CSCF.
`maximum-connections`	Specifies the maximum number of opened connections to maintain in this pool.

Example 21–6 shows a sample `connection-reuse-pool` configuration having two pools.

Example 21–6 Sample connection-reuse-pool Configuration

```
<connection-reuse-pool>
   <pool-name>SBCPool</pool-name>
   <destination>MySBC</destination>
   <destination-port>7070</destination-port>
   <maximum-connections>10</maximum-connections>
</connection-reuse-pool>
<connection-reuse-pool>
   <pool-name>SCSFPool</pool-name>
   <destination>192.168.1.6</destination>
   <destination-port>7071</destination-port>
   <maximum-connections>10</maximum-connections>
</connection-reuse-pool>
```

globally-routable-uri

This element enables you to specify a Globally-Routable User Agent URI (GRUU) that Converged Application Server automatically inserts into Contact and Route-Set headers when communicating with network elements. The URI specified in this element should be the GRUU for the entire Converged Application Server cluster. (In a single-server domain, use a GRUU for the server itself.)

Note that User Agents (UAs) deployed on Converged Application Server typically obtain GRUUs via a registration request. In this case, the application code is responsible both for requesting and subsequently handling the GRUU. To request a GRUU, the UA includes the `+sip.instance` field parameter in the Contact header in each Contact for which GRUU is required. Upon receiving a GRUU, the UA uses the GRUU as the URI for the Contact header field when generating new requests.

domain-alias-name

This element defines one or more domains for which Converged Application Server is responsible. If a message has a destination domain that matches a domain specified with a `domain-alias-name` element, Converged Application Server processes the message locally, rather than forwarding it.

The **sipserver.xml** configuration file can have multiple `main-alias-name` elements. Each element can specify either:

- an individual, fully-qualified domain name, such as `myserver.mycompany.com`, or
- a domain name starting with an initial wildcard character, such as `*.mycompany.com`, used to represent all matching domains. Note that only a single wildcard character is supported, and it must be used as the first element of the domain name.

Note: You can also identify these domain names using the Domain Aliases field in the Configuration > General tab of the SipServer Administration Console extension.

enable-rport

This element determines whether or not Converged Application Server automatically adds an `rport` parameter to Via headers when acting as a UAC. By default, the server does not add the `rport` parameter; set the element to `true` to automatically add `rport` to requests generated by the server.

Note: You can also set this parameter to `true` by selecting the Symmetric Response Routing option in the Administration Console. In the Administration Console, select **Configuration**, then select the **General** tab of the SipServer Administration console extension.

The `rport` parameter is used for symmetric response routing as described in RFC 3581 (`http://www.ietf.org/rfc/rfc3581.txt`). When a message is received by an RFC 3581-compliant server, such as Converged Application Server, the server responds using the remote UDP port number from which the message was received, rather than the port number specified in the Via header. This behavior is frequently used when servers reside behind gateway devices that perform Network Address Translation (NAT). The NAT devices maintain a binding between the internal and external port numbers, and all communication must be initiated via the gateway port.

Note that Converged Application Server is compliant with RFC 3581, and will honor the `rport` parameter even if you set the `enable-rport` element to "false." The `enable-rport` element only specifies whether the server automatically adds `rport` to the requests it generates when acting as a UAC. To disable `rport` handling completely (disable RFC 3581 support), you must start the server with the command-line option, `-Dwlss.udp.uas.rport=false`.

Note: `rport` support as described in RFC 3581 requires that SIP responses include the source port of the original SIP request. Because source port information is frequently treated as sensitive data, Oracle recommends using the TLS transport.

See information on rport-based configurations in *Oracle Communications Converged Application Concepts* for an example SIP and NAT interaction using the `rport` parameter.

image-dump-level

This element specifies the level of detail to record in Converged Application Server diagnostic image files. You can set this element to one of two values:

- `basic`: Records all diagnostic data except for call state data.
- `full`: Records all diagnostic data including call state data.

Note: Recording call state data in the image file can be time consuming. By default, image dump files are recorded using the `basic` option.

You can also set this parameter using the Configuration > General tab of the SipServer Administration console extension.

See Chapter 16, "Using the WebLogic Server Diagnostic Framework (WLDF)," for more information about generating diagnostic image files.

stale-session-handling

Converged Application Server uses encoded URIs to identify the call states and application sessions associated with a message. When an application is undeployed or upgraded to a new version, incoming requests may have encoded URIs that specify "stale" or nonexistent call or session IDs. The `stale-session-handling` element enables you to configure the action that Converged Application Server takes when it encounters stale session data in a request. The following actions are possible:

- `drop`: Drops the message without logging an error. This setting is desirable for systems that frequently upgrade applications using Converged Application Server's in-place upgrade feature. Using the `drop` action ensures that messages intended for older, incompatible versions of a deployed application are dropped.
- `error`: Responds with an error, so that a UAC might correct the problem. This is the default action. Messages having a `To:` tag cause a `481 Call/Transaction Does Not Exist` error, while those without the tag cause a `404 Not Found` error.
- `continue`: Ignores the stale session data and continues processing the request.

Note: When it encounters stale session data, Converged Application Server applies the action specified by `stale-session-handling` before considering the value of the `default-behavior` element. This means that the `default-behavior` is performed only when you have configured `stale-session-handling` to perform the `continue` action.

---

enable-contact-provisional-response

By default Converged Application Server does not place a Contact header in non-reliable provisional (1xx) responses that have a To header. If you deploy applications that expect the Contact header to be present in such 1xx responses, set this element to true:

```
<enable-contact-provisional-response>true</enable-contact-provisional-response>
```

Note that setting this element to true does not affect 100 Trying responses.

app-router

The `app-router` stanza contains several elements that configure SIP Servlet v1.1 application router behavior. See "use-custom-app-router", "app-router-config-data", "custom-app-router-jar-file-name", and "default-application-name".

use-custom-app-router

The `use-custom-app-router` element determines whether Converged Application Server uses the default, built-in Application AR (AR), or a custom AR that you specify with the `custom-app-router-jar-file-name` element. The default value, "false," configures the server to use the default AR. See "Configuring a Custom Application Router" in *Converged Application Server Application Development Guide* for more information.

app-router-config-data

The `app-router-config-data` element defines properties to pass to the default or custom Application Router (AR) in the `init` method. All configuration properties must conform to the Java Properties format, and each individual property must be entered on a separate, single line without line breaks or spaces. DAR properties must conform to the detailed property format described in Appendix C of `http://jcp.org/en/jsr/detail?id=289`.

Example 21–7 shows an example configuration.

Example 21–7 Sample app-router-config-data element

```
<?xml version='1.0' encoding='UTF-8'?>
<sip-server xmlns="http://www.bea.com/ns/wlcp/wlss/300"
xmlns:sec="http://www.bea.com/ns/weblogic/90/security"
xmlns:wls="http://www.bea.com/ns/weblogic/90/security/wls"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <app-router>
    <use-custom-app-router>false</use-custom-app-router>

<app-router-config-data>INVITE:("OriginatingCallWaiting","DAR:From","ORIGINATING",
"","NO_ROUTE","0"),("CallForwarding","DAR:To","TERMINATING","","NO_ROUTE","1")
SUBSCRIBE:("CallForwarding","DAR:To","TERMINATING","","NO_
ROUTE","1")</app-router-config-data>
```

```
    <custom-app-router-jar-file-name></custom-app-router-jar-file-name>
    <default-application-name></default-application-name>
  </app-router>
</sip-server>
```

You can optionally specify AR initialization properties when starting the Converged Application Server instance by including the `-Djavax.servlet.sip.ar.dar.configuration` Java option. (To specify a property file, rather than a URI, include the prefix `file:///`) If you specify the Java startup option, the container ignores any configuration properties defined in `app-router-config-data`. You can modify the properties in at any time, but the properties are not passed to the AR until the server is restarted with the `-Djavax.servlet.sip.ar.dar.configuration` option omitted.

See "Configuring a Custom Application Router" in the *Converged Application Server Application Developer's Guide* for more information.

custom-app-router-jar-file-name

The `custom-app-router-jar-file-name` element specifies the filename of the custom Application Router (AR), packaged as a JAR file, to use. The custom AR implementation must reside in the *$DOMAIN_HOME*`/approuter` subdirectory.

See "Configuring a Custom Application Router" in the *Converged Application Server SIP Application Developer's Guide* for more information.

default-application-name

The `default-application-name` element specifies the name of a default application that the container should call when the custom Application Router (AR) cannot find an application to process an initial request. If no default application is specified, the container returns a 500 error if the AR cannot select an application.

Note: You must first deploy an application before specifying its name as the value of **Default application name**.

See "Configuring a Custom Application Router" in the *Converged Application Server Application Developer's Guide* for more information.

22 SIP Data Tier Configuration Reference (datatier.xml)

The chapter describes the SIP data tier configuration file, **datatier.xml**:

Overview of datatier.xml

The **datatier.xml** configuration file identifies servers that manage the concurrent call state for SIP applications, and defines how those servers are arranged into SIP data tier *partitions*. A *partition* refers to one or more SIP data tier server instances that manage the same portion of the call state. Multiple servers in the same partition are referred to as *replicas* because they all manage a copy of the same portion of the call state.

datatier.xml is stored in the ***CAS_Home*/config/custom** subdirectory where ***CAS_Home*** is the root directory of Oracle Communications Converged Application Server.

Editing datatier.xml

You can edit **datatier.xml** using either the Administration Console or a text editor. Note that changes to the SIP data tier configuration cannot be applied to servers dynamically; you must restart servers in order to change SIP data tier membership or reconfigure partitions.

XML Schema

This schema file is bundled within the **wlss-descriptor-binding.jar** library, installed in the ***WLS_Home*/server/lib** directory.

Example datatier.xml File

Example 22–1 shows the template **datatier.xml** file created using the Configuration Wizard. See also "Example SIP Data Tier Configurations and Configuration Files" in Chapter 4, "Configuring SIP Data Tier Partitions and Replicas."

Example 22–1 Default datatier.xml File

```
<st:data-tier xmlns:st="http://bea.com/wcp/sip/management/internal/webapp">
 <st:partition>
   <st:name>partition-0</st:name>
   <st:server-name>replica1</st:server-name>
   <st:server-name>replica2</st:server-name>
 </st:partition>
</st:data-tier>
```

XML Element Description

datatier.xml contains one or more `partition` elements that define servers' membership in a SIP data tier partition. All SIP data tier clusters must have at least one `partition`. Each partition contains the XML elements described in Table 22–1.

Table 22–1 Nested partition Elements

Element	Description
`name`	A String value that identifies the name of the partition. Oracle recommends including the number of the partition (starting at 0) in the text of the name for administrative purposes. For example, "partition-0."
`server-name`	Specifies the name of a Converged Application Server instance that manages call state in this partition. You can define up two three servers per `partition` element. Multiple servers in the same partition maintain the same call state data, and are referred to as *replicas*. Oracle recommends including the number of the server (starting with 0) and the number of the partition in the server name for administrative purposes. For example, "replica-0-0."

23 Diameter Configuration Reference (diameter.xml)

This chapter describes the Diameter configuration file, **diameter.xml**:

Overview of diameter.xml

The **diameter.xml** file configures attributes of a Diameter node, such as:

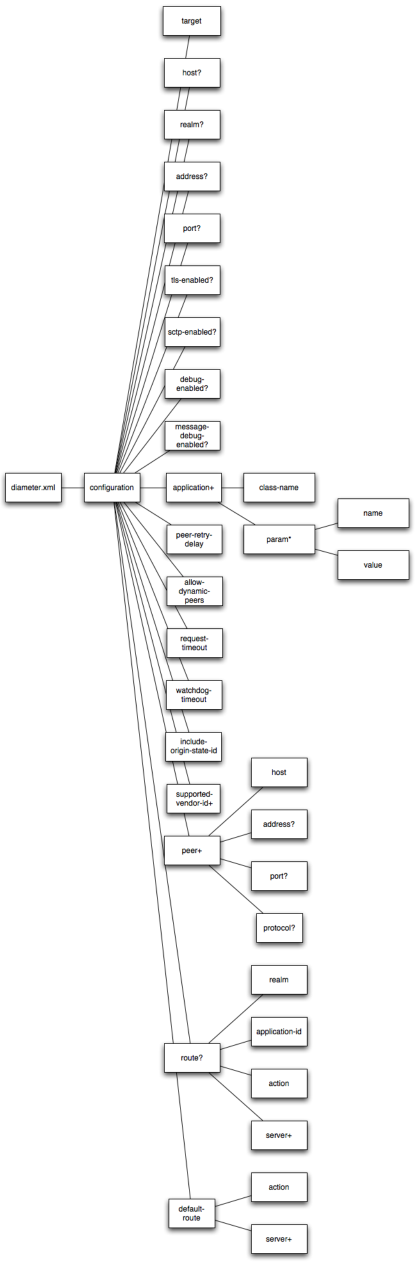
- The host identity of the Diameter node
- The Diameter applications that are deployed on the node
- Connection information for Diameter peer nodes
- Routing information and default routes for handling Diameter messages.

The Diameter protocol implementation reads the configuration file at boot time. **diameter.xml** is stored in the ***DOMAIN_home*/config/custom** subdirectory where ***DOMAIN_home*** is the root directory of the Oracle Communications Converged Application Server domain.

Graphical Representation

Figure 23–1 shows the element hierarchy of the **diameter.xml** file.

Figure 23–1 Element Hierarchy of diameter.xml



Editing diameter.xml

You should never move, modify, or delete the **diameter.xml** file during normal operations.

Oracle recommends using the Administration Console to modify **diameter.xml** indirectly, rather than editing the manually with a text editor. Using the Administration Console ensures that the **diameter.xml** document always contains valid XML.

You may need to manually view or edit **diameter.xml** to troubleshoot problem configurations, repair corrupted files, or to roll out custom Diameter node configurations to a large number of machines when installing or upgrading

Converged Application Server. When you manually edit **diameter.xml**, you must reboot Diameter nodes to apply your changes.

Caution: Always use the Diameter node in the Administration Console or the WLST utility, as described in Chapter 3, "Configuring Converged Application Container Properties"to make changes to a running Converged Application Server deployment.

Steps for Editing diameter.xml

If you need to modify **diameter.xml** on a production system, follow these steps:

1. Use a text editor to open the *DOMAIN_home*/**config/custom/diameter.xml** file, where *DOMAIN_home* is the root directory of the Converged Application Server domain.
2. Modify the **diameter.xml** file as necessary. See "XML Element Description" for a full description of the XML elements.
3. Reboot or start servers to have your changes take effect.
4. Test the updated system to validate the configuration.

XML Schema

The XML schema file (**wcp-diameter.xsd**) is bundled within the **wlssdiameter.jar** library, installed in *WL_home***sip/server/lib**, where *WL_home* is the path to the directory where WebLogic Server is installed.

Example diameter.xml File

See Chapter 14, "Configuring Diameter Client Nodes and Relay Agents," for examples of **diameter.xml** configuration files.

XML Element Description

The following sections describe each XML element in **diameter.xml**.

configuration

The top level `configuration` element contains the entire diameter node configuration.

target

Specifies one or more target Converged Application Server instances to which the node configuration is applied. The target servers must be defined in the **config.xml** file for your domain.

host

Specifies the host identity for this Diameter node. If no `host` element is specified, the identity is taken from the local server's host name. Note that the host identity may or may not match the DNS name.

Note: When configuring Diameter support for multiple Sh client nodes, it is best to omit the `host` element from the **diameter.xml** file. This enables you to deploy the same Diameter Web Application to all servers in the engine tier cluster, and the host name is dynamically obtained for each server instance.

---

realm

Specifies the realm name for which this Diameter node has responsibility. You can run multiple Diameter nodes on a single host using different realms and listen port numbers. The HSS, Application Server, and relay agents must all agree on a realm name or names. The realm name for the HSS and Application Server need not match.

If you omit the `realm` element, the realm named is derived using the domain name portion of the host name, if the host name is fully-qualified (for example, host@oracle.com).

address

Specifies the listen address for this Diameter node, using either the DNS name or IP address. If you do not specify an address, the node uses the `host` identity as the listen address.

Note: The `host` identity may or may not match the DNS name of the Diameter node. Oracle recommends configuring the `address` element with an explicit DNS name or IP address to avoid configuration errors.

---

port

Specifies the TCP or TLS listen port for this Diameter node. The default port is 3868.

tls-enabled

This element is used only for standalone node operation to advertise TLS capabilities.

Converged Application Server ignores the `tls-enabled` element for nodes running within a server instance. Instead, TLS transport is reported as enabled if the server instance has configured a Network Channel having TLS support (a diameters channel). See "Creating TCP, TLS, and SCTP Network Channels for the Diameter Protocol" in Chapter 14, "Configuring Diameter Client Nodes and Relay Agents."

sctp-enabled

This element is used only for standalone node operation to advertise SCTP capabilities.

Converged Application Server ignores the `sctp-enabled` element for nodes running within a server instance. Instead, SCTP transport is reported as enabled if the server instance has configured a Network Channel having SCTP support (a diameter-sctp channel). See "Creating TCP, TLS, and SCTP Network Channels for the Diameter Protocol" in Chapter 14, "Configuring Diameter Client Nodes and Relay Agents."

debug-enabled

Specifies a boolean value to enable or disable debug message output. Debug messages are disabled by default.

message-debug-enabled

Specifies a boolean value to enable or disable tracing of Diameter messages. This element is disabled by default.

application

Configures a particular Diameter application to run on the selected node. Converged Application Server includes applications to support nodes that act as Diameter Sh, Ro, and Rf clients, Diameter relay agents, or Home Subscriber Servers (HSS). Note that the HSS application is a simulator that is provided only for development or testing purposes.

class-name

Specifies the application class file to load.

param*

Specifies one or more optional parameters to pass to the application class.

name Specifies the name of the application parameter.

value Specifies the value of the parameter.

peer-retry-delay

Specifies the number of seconds this node waits between retries to Diameter peers. The default value is 30 seconds.

allow-dynamic-peers

Specifies a boolean value that enables or disables dynamic peer configuration. Dynamic peer support is disabled by default. Oracle recommends enabling dynamic peers only when using the TLS transport, because no access control mechanism is available to restrict hosts from becoming peers.

request-timeout

Specifies the number of milliseconds to wait for an answer from a peer before timing out.

watchdog-timeout

Specifies the number of seconds used for the Diameter Tw watchdog timer.

include-origin-state-id

Specifies whether the node should include the origin state AVP in requests and answers.

supported-vendor-id+

Specifies one or more vendor IDs to be added to the `Supported-Version-Ids` AVP in the capabilities exchange.

peer+

Specifies connection information for an individual Diameter peer. You can choose to configure connection information for individual peer nodes, or allow any node to be dynamically added as a peer. Oracle recommends using dynamic peers only if you are using the TLS transport, because there is no way to filter or restrict hosts from becoming peers when dynamic peers are enabled.

When configuring Sh client nodes, the `peers` element should contain peer definitions for each Diameter relay agent deployed to your system. If your system does not use relay agents, you must include a peer entry for the Home Subscriber Server (HSS) in the system, as well as for all other engine tier nodes that act as Sh client nodes.

When configuring Diameter relay agent nodes, the `peers` element should contain peer entries for all Diameter client nodes that access the peer, as well as the HSS.

host

Specifies the host identity for a Diameter peer.

address

Specifies the listen address for a Diameter peer. If you do not specify an address, the host identity is used.

port

Specifies the TCP or TLS port number for this Diameter peer. The default port is 3868.

protocol

Specifies the protocol used by the peer. This element may be one of `tcp` or `sctp`.

route

Defines a realm-based route that this node uses when resolving messages.

When configuring Sh client nodes, you should specify a route to each Diameter relay agent node deployed in the system, as well as a `default-route` to a selected relay. If your system does not use relay agents, simply configure a single `default-route` to the HSS.

When configuring Diameter relay agent nodes, specify a single `default-route` to the HSS.

realm

The target realm used by this route.

application-id

The target application ID for the route.

action

An action type that describes the role of the Diameter node when using this route. The value of this element can be one of the following:

- none
- local
- relay
- proxy
- redirect

server+

Specifies one or more target servers for this route. Note that any server specified in the `server` element must also be defined as a `peer` to this Diameter node, or dynamic peer support must be enabled.

default-route

Defines a default route to use when a request cannot be matched to a configured route.

action

Specifies the default routing action for the Diameter node. See "action".

server+

Specifies one or more target servers for the default route. Any server you include in this element must also be defined as a `peer` to this Diameter node, or dynamic peer support must be enabled.

24

Profile Service Provider Configuration Reference (profile.xml)

This chapter describes a profile provider configuration file, **profile.xml**:

- Overview of profile.xml
- Graphical Representation
- Editing profile.xml
- XML Schema
- Example profile.xml File
- XML Element Description

Overview of profile.xml

The **profile.xml** file configures attributes of a profile service provider, such as:

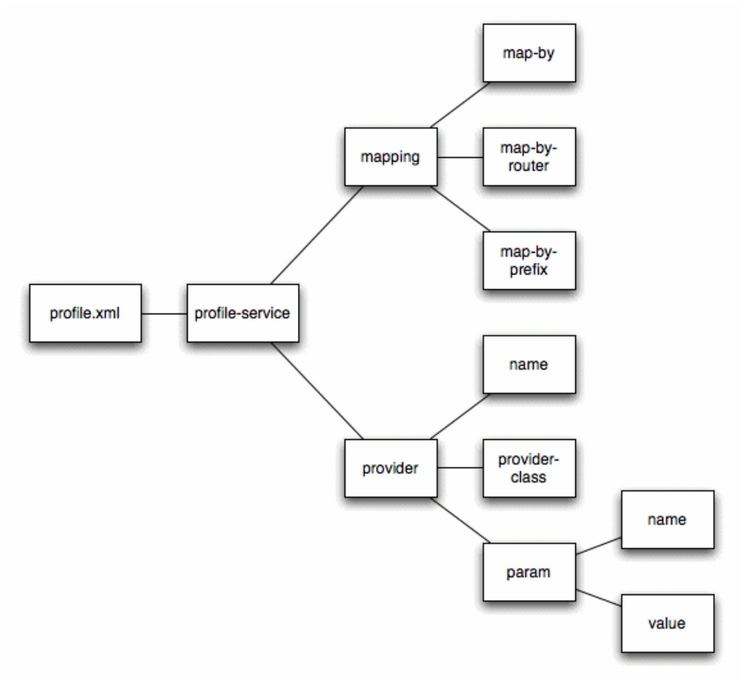
- The name of the provider
- The class name of the provider implementation
- Optional arguments passed to the provider
- Mapping rules for using the provider

The **profile.xml** file is located in the *CAS_home***/config/custom** subdirectory where *CAS_home* is the root directory of the Oracle Communications Converged Application Server domain.

Graphical Representation

Figure 24–1 shows the element hierarchy of the **profile.xml** file.

Figure 24–1 Element Hierarchy of profile.xml



Editing profile.xml

Oracle recommends using the Administration Console profile service extension to modify **profile.xml** indirectly, rather than editing the file manually using a text editor or other XML editing application. Using the Administration Console ensures that the **profile.xml** document always contains valid XML. See "Configuring Profile Providers Using the Administration Console" in the *Converged Application Server Application Developer's Guide*.

You may need to manually view or edit **profile.xml** to troubleshoot problem configurations, repair corrupted files, or to roll out custom profile provider configurations to a large number of machines when installing or upgrading Converged Application Server. When you manually edit **profile.xml**, you must reboot servers to apply your changes.

Steps for Editing profile.xml

If you need to modify **profile.xml** on a production system, follow these steps:

1. Use a text editor to open the *CAS_home***/config/custom/profile.xml** file, where *CAS_home* is the root directory of the Converged Application Server domain.
2. Modify the **profile.xml** file as necessary. See "XML Element Description" for a full description of the XML elements.
3. Save your changes and exit the text editor.
4. Reboot or start servers to have your changes take effect:
5. Test the updated system to validate the configuration.

XML Schema

The full schema for the **profile.xml** file is bundled within the **profile-service-descriptor-binding.jar** library, located in *WL_home*/**sip/server/lib**, where *WL_home* is the path to the directory where WebLogic Server is installed.

Example profile.xml File

See "Developing Custom Profile Providers" in the *Converged Application Server Application Developer's Guide* for examples of **profile.xml** configuration files.

XML Element Description

The following sections describe each XML element in **profile.xml**.

profile-service

The top level `profile-service` element contains the entire profile service configuration.

mapping

Specifies how requests for profile data are mapped to profile provider implementations.

map-by

Specifies the technique used for mapping documents to providers:

- `router` uses a custom router class, specified by `map-by-router`, to determine the provider.
- `prefix` uses the specified `map-by-prefix` entry to map documents to a provider.
- `provider-name` uses the specified `name` element in the `provider` entry to map documents to a provider.

map-by-prefix

Specifies the prefix used to map documents to profile providers when mapping by prefix.

map-by-router

Specifies the router class (implementing `com.bea.wcp.profile.ProfileRouter`) used to map documents to profile providers with router-based mapping.

provider

Configures the profile provider implementation and startup options.

name

Specifies a name for the provider configuration. The `name` element is also used for mapping documents to the provider if you specify the `provider-name` mapping technique.

provider-class

Specifies the profile provider class (implementing `com.bea.wcp.profile.ProfileServiceSpi`).

param

Uses the `name` and `value` elements to specify optional parameters to the provider implementation.