Oracle® Communications Converged Application Server

Administrator Guide

Release 8.2
G34027-02
August 2025

Oracle Communications Converged Application Server Administrator Guide, Release 8.2

G34027-02

Contents

Preface

Revision History

1 Converged Application Server Configuration

2 Configuring Infrastructure Components

4 Reference

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.

My Oracle Support

My Oracle Support (https://support.oracle.com) is your initial point of contact for all product support and training needs. A representative at Customer Access Support (CAS) can assist you with My Oracle Support registration.

Call the CAS main number at 1-800-223-1711 (toll-free in the US), or call the Oracle Support hotline for your local country from the list at http://www.oracle.com/us/support/contact/index.html. When calling, make the selections in the sequence shown below on the Support telephone menu:

1. Select 2 for New Service Request.
2. Select 3 for Hardware, Networking, and Solaris Operating System Support.
3. Select one of the following options:
   - For technical issues such as creating a new Service Request (SR), select 1.
   - For non-technical issues such as registration or assistance with My Oracle Support, select 2.

You are connected to a live agent who can assist you with My Oracle Support registration and opening a support ticket.

My Oracle Support is available 24 hours a day, 7 days a week, 365 days a year.

Emergency Response

In the event of a critical service situation, emergency response is offered by the Customer Access Support (CAS) main number at 1-800-223-1711 (toll-free in the US), or call the Oracle Support hotline for your local country from the list at http://www.oracle.com/us/support/contact/index.html. The emergency response provides immediate coverage, automatic escalation, and other features to ensure that the critical situation is resolved as rapidly as possible.

A critical situation is defined as a problem with the installed equipment that severely affects service, traffic, or maintenance capabilities, and requires immediate corrective action. Critical situations affect service and/or system operation resulting in one or several of these situations:

- A total system failure that results in loss of all transaction processing capability

- Significant reduction in system capacity or traffic handling capability
- Loss of the system's ability to perform automatic system reconfiguration
- Inability to restart a processor or the system
- Corruption of system databases that requires service affecting corrective actions
- Loss of access for maintenance or recovery operations
- Loss of the system ability to provide any required critical or major trouble notification

Any other problem severely affecting service, capacity/traffic, billing, and maintenance capabilities may be defined as critical by prior discussion and agreement with Oracle.

Locate Product Documentation on the Oracle Help Center Site

Oracle Communications customer documentation is available on the web at the Oracle Help Center (OHC) site, http://docs.oracle.com. You do not have to register to access these documents. Viewing these files requires Adobe Acrobat Reader, which can be downloaded at http://www.adobe.com.

1. Access the Oracle Help Center site at http://docs.oracle.com.
2. Click **Industries**.
3. Under the Oracle Communications sub-header, click the **Oracle Communications documentation** link.
   The Communications Documentation page appears. Most products covered by these documentation sets appear under the headings "Network Session Delivery and Control Infrastructure" or "Platforms."
4. Click on your Product and then Release Number.
   A list of the entire documentation set for the selected product and release appears.
5. To download a file to your location, right-click the **PDF** link, select **Save target as** (or similar command based on your browser), and save to a local folder.

Access to Oracle Support

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

Revision History

Table 1 Revision History

Date	Revision
July 2025	- Initial release
August 2025	- Corrects cluster name

1 Converged Application Server Configuration

The Converged Application Server is based on Oracle WebLogic Server, 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 the Converged Application Server, such as tasks related to networking, security, and cluster configuration for the engine and the Coherence cache.

HTTP server configuration and other basic configuration tasks such as server logging are addressed in the Oracle WebLogic Server documentation. See "Overview of WebLogic Server System Administration" in *Understanding Oracle WebLogic Server* to get started.

Overview of Configuration and Administration Tools

You can apply configuration changes using the Remote Console, using the REST API, or from the command line using the WLST utility. Changes to certain SIP Servlet container properties require a restart of the engine for the change to take affect. In such cases, a *Server or App Restart Required* icon appears in the Remote Console.

> **Note**
>
> The administrator must restart the server when fields with *Server or App Restart Required* are modified.

The following sections contain more information about the configuration tools:

- WebLogic Remote Console
- WebLogic Scripting Tool (WLST)
- REST API
- Additional Configuration Methods

Remote Console

You can use the Remote Console to view or edit the Converged Application Server. The Converged Application Server configuration and monitoring is provided through the Edit Tree and the Monitoring Tree.

Under the Edit Tree or the Configuration View Tree, you can access most of the settings by clicking **Custom Resources**, and then **sipserver**, and then **SIP Server**.

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 WLST information, see:

- For information about WLST, see "Using the WebLogic Scripting Tool" in *Understanding the WebLogic Scripting Tool*.
- For information about WLST commands, see "WLST Command and Variable Reference" in *WLST Command Reference for WebLogic Server*.

REST API

You can use the REST API to retrieve runtime metrics; create, modify, or delete address-of-record entries in the Location Service; and manage configuration settings.

For information about the REST API, see the *Developer Guide*.

Additional Configuration Methods

Most Converged Application Server configuration is performed using either the Remote 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.
- **coherence.xml** identifies servers that participate in SIP state storage, and also defines the number of threads and partitions available in the state storage service.
- **diameter.xml** defines Diameter nodes and Diameter protocol applications used in the domain.

See 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 Java API Reference*.

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 to perform batch configuration. See "Configuring Converged Application Server Container Properties" 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 Server. See *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.

Getting Started

This chapter describes how to start and stop servers in an Converged Application Server domain.

Accessing the Remote Console

> **Note**
>
> Before you can access the Remote Console, you must apply the April 2025 Critical Patch Update.

The Remote Console enables you to configure and monitor core WebLogic Server functionality as well as the SIP Servlet container functionality provided with Converged Application Server. Follow the steps in the Get Started section of the *Oracle WebLogic Remote Console Online Help* to install the Remote Console and connect it to a provider.

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 Converged Application Server instance. To learn how to use WLST, see "Using the WebLogic Scripting Tool" in *Understanding the WebLogic Scripting Tool*.

Before using WLST to configure a domain, set your environment to add required classes to your classpath. Use either a domain environment script or the **setDomainEnv.sh** script located in `<domain_home>/bin` where *domain_home* is the domain home that you set during installation.

Configuration MBeans for the SIP Servlet Container

`ConfigManagerRuntimeMBean` manages access to and persists the configuration MBean attributes described in the **com.bea.wcp.sip.management.descriptor.beans** package of the *Converged Application Server Java API Reference*. 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`.

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. The example below 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 1-1 Template WLST Script for Accessing ConfigManagerRuntimeMBean

```
# Connect to the Administration Server
connect('username','password','t3://localhost:7001')
# Start an edit session
edit()
startEdit()
# --MODIFY THIS SECTION AS NECESSARY--
# Edit SIP Servlet container configuration MBeans
cd('mydomain:DomainConfig=mydomain,Location=myserver,Name=myserver,SipServer=myserver,Typ
e=Proxy')
set('RoutingPolicy','domain')
# Commit changes
save()
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.

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

Example 1-2 WLST Commands for Creating and Deleting MBeans

```
connect('username','password','t3://localhost:7001')
edit()
startEdit()
cd('CustomResources/sipserver/Resource/sipserver')
cmo.destroyOverload()
cmo.createProxy()
save()
activate()
```

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. The example below modifies the previous example to add a new URI attribute to the **LoadBalancer** MBean. See also the *Oracle Converged Application Server Java API Reference* for a full reference to the **URIHelper** class.

Example 1-3 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=ConfigManagerRu
ntime')
cmo.startEdit()
cd('mydomain:DomainConfig=mydomain,Location=myserver,Name=sipserver,Type=SipServer')
cmo.createProxy()
cd('mydomain:DomainConfig=mydomain,Location=myserver,Name=sipserver,SipServer=sipserver,T
ype=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=ConfigManagerRu
ntime')
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 Remote Console, see Log Messages in the *Oracle WebLogic Remote Console Online Help*.

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. In the Edit Tree, expand **Custom Resources**, and then **sipserver**, and then **SIP Server**.
2. Click the **Message Debug** node.
3. Toggle the **Enable Debug** slider to its on position.
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**.
6. Click the Shopping Cart and then **Commit Changes**.
7. Restart the WebLogic Server.

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 for suggestions about maximizing JVM performance in a production domain.

> ⚠ **Caution**
>
> When you configure a domain with multiple 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 for more information.

Because a typical Converged Application Server domain contains numerous engines, 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 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 the engine servers.
   You generally start each SIP Coherence server by using either the **startManagedWebLogic.sh** script installed with the Configuration Wizard, or a custom startup script. The **startManagedWebLogic.sh** script requires that you specify the name of the server to start up and the URL of the Administration Server for the domain. For example:

   ```
   startManagedWebLogic.sh engine0-0 t3://adminhost:7001
   ```

Following the above startup sequence ensures that all Managed Servers use the latest SIP Servlet container and Coherence cache configuration.

Startup Command Options

The following table lists startup options available to Converged Application Server. For more information about these and other options, see "WLST Command and Variable Reference" in *WLST Command Reference for WebLogic Server*.

Table 1-1 Startup Command Options

Application	Startup Option	For More Information
Installer	-Djava.io.tmpdir	See the discussion about Temporary Disk Space Requirements in the *Fusion Middleware System Requirements and Specifications*.
SIP Servlet Application Router	-Djavax.servlet.sip.ar.spi.SipApplicationRouterProvider	See Configuring a Custom Application Router in *Converged Application Server Developer Guide*.
SIP Servlet Application Router	-Djavax.servlet.sip.dar.configuration	See Using the Default Application Router in *Converged Application Server Developer Guide*.
Converged Application Server	-Dweblogic.management.discover	See Restarting an Administration Server on the Same System.

Table 1-1 (Cont.) Startup Command Options

Application	Startup Option	For More Information
Converged Application Server	-Dweblogic.RootDirectory	See Restarting an Administration Server on Another System.
Converged Application Server	–Dwlss.dialog.index.enabled	See Join and Replaces Header Support in *Converged Application Server Developer Guide*.
Converged Application Server	-Dwlss.local.serialization	See Optimizing Memory Utilization and Performance with Serialization in *Converged Application Server Developer Guide*.
Converged Application Server	-Dwlss.sip.session.count.log_interval	See Configuring the License Tracking as Startup Command Options.
Converged Application Server	-Dwlss.sip.session.count.start_time	See Configuring the License Tracking as Startup Command Options.
Converged Application Server	-Dwlss.send100ForNonInviteTransaction	See the description about Sending Provisional Responses to Non-Invite Requests in *Converged Application Server Developer Guide*.
Converged Application Server	-Dwlss.udp.lb.masquerade	See information about Network Address Translation Options in *Converged Application Server Concepts*.
Converged Application Server	-Dwlss.udp.listen.on.ephemeral	See information about Single-NIC Configurations with TCP and UDP Channels in *Converged Application Server Concepts*.

Reverting to the Original Boot Configuration

When you boot the Administration Server for a Converged Application Server domain, the server parses the current container configuration in **sipserver.xml**. It generates a copy of the initial configuration in a file named **sipserver.xml.booted** in the *Domain_home***/config/custom** directory, where *Domain_home* is the directory in which the Converged Application Server domain resides. This backup 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.

Configuring Converged Application Server Container Properties

This section describes how to configure SIP container features in the engine of an Oracle Communications Converged Application Server deployment.

Configure General SIP Application Server Properties

Loading SIP applications to the Converged Application Server in the Remote Console is similar to loading any application to WebLogic server. You use the Deployments node in the Edit Tree 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 if necessary modify the default values for the general settings.

1. In the Edit Tree, click **Custom Resources**, and then **sipserver**, and then **SIP Server**.

2. Use the fields in the **General** subtab 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 on-screen field descriptions.
3. Click **Save** to save your configuration changes.
4. Click the shopping cart and then **Commit Changes**.
5. Restart the server.

Adding Servers to the Cluster

If you have configured a replicated domain using the domain configuration wizard, Converged Application Server instances include the default **BEA_ENGINE_TIER_CLUST** cluster. You can assign additional managed servers to each cluster as needed when performance requirements in your environment require them.

For more information on clustering, see Understanding WebLogic Server Clustering in *Administering Clusters for Oracle WebLogic Server*.

Configuring Timer Processing

As engine servers add new call state data to the SIP call-state store, they maintain data structures to track the SIP protocol timers and application timers associated with each call. Engine servers periodically poll the SIP Coherence call-state store to determine which timers have expired, given the current time. By default, multiple engine polls to the call-state store are staggered to avoid contention on the timer tables. Engine servers then process all expired timers using threads allocated in the **wlss.timer** work manager.

Configuring Timer Affinity (Optional)

With the default timer processing mechanism, a given engine processes all timers that are currently due to fire, regardless of whether 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, each engine server periodically polls the SIP call-state store for processed timers. When polling the SIP call-state store, 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 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 call-state store.

To enable timer affinity:

1. From the Edit Tree of the Remote Console, select **Custom Resources**, and then **sipserver**, and then **SIP Server**
2. Toggle the slider for **Enable Timer Affinity**.
3. Click **Save** to save your configuration changes.
4. Click the shopping cart and then **Commit Changes** to apply your changes to the engine 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 servers must accurately synchronize their system clocks to a common time source, to within one or two milliseconds. Large differences in system clocks cause severe problems such as:

- SIP timers firing prematurely on servers with fast clock settings.
- Poor distribution of timer processing among engine servers. For example, one engine server might process all expired timers, whereas other engine 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 server with a system clock 250 milliseconds faster than other servers will process more expired timers than other engine servers, will cause retransmits to begin in half the allotted time, and may force messages to time out prematurely.

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 use 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 in your system. Engines access the SIP call-state store using the Coherence cluster configured in the domain.

> **Note**
>
> If you configure the Coherence cluster to use Unicast addressing, you must configure the engines to use either explicit listen addresses or explicit well-known addresses to allow all cluster domain servers to locate each other.

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 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 Server servers, you must define a channel on each engine for each one. When a particular network interface on the engine 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 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. Enable IPv6 for SIP traffic by configuring a network channel with an IPv6 address. You must configure an IPv6 SIP channel on each engine server that will support IPv6 traffic.

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. You can configure a single engine with both an IPv4 and IPv6 channel to support multiple, separate networks.

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

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 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. From the Edit Tree of the Remote Console, select **Custom Resource**, and then **sipserver**, and then **SIP Server**.
2. Under the General section, toggle **Enable DNS Server Lookup**.
3. Click **Save** to save your changes.

4. Click the shopping cart and then **Commit 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 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 port number of the destination, using the information provided in the **sent-by** field and the **Via** the header.

Configuring Network Channels for SIP or SIPS

When you create a 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 channel using the instructions in "Creating a New SIP or SIPS Channel".

To reconfigure a channel:

1. From the Remote Console, select **Environment**, and then **Servers**, and then a specific server, and then **Channels**.
2. Select the channel to delete and click **Delete**.

To reconfigure an existing channel:

1. Select the channel's link from **Name** column of the channel list (for example, the default **sip** channel).
2. Edit the **Listen Address** or **Listen Port** fields to correspond to the address of a NIC or logical address on the associated engine server.

> **Note**
>
> The channel must be disabled before you can modify the listen address or listen port. Disable the channel by deselecting the **Enabled** check box.

3. 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.
4. Edit the advanced channel attributes as necessary (see "Creating a New SIP or SIPS Channel" for details.)

To save your changes:

1. Click **Save**.
2. Click the shopping cart and click **Commit Changes**.
3. Restart the server.

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. From the Remote Console, select **Environment**, and then **Servers**.
2. Select your server to configure and click **Channels**.
3. Click **New** to configure a new channel.
4. 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. A SIP channel cannot support only UDP or only TCP transport on the configured port.
   - **Listen Address:** Enter the IP address or DNS name for this channel. On a DNS server, 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.
5. If required, click Show Advaned Fields and set the advanced properties:
   - **Enabled**: This attribute specifies whether to start the new channel.
   - **Tunneling Enabled**: This attribute specifies whether tunneling through HTTP should be enabled for this network channel. This value is not inherited from the server's configuration.
   - **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.
   - **Outbound Enabled**: This attribute cannot be unchecked, because all SIP and SIPS channels can originate network connections.
6. Click **Save**.

7. Click the shopping cart and then **Commit Changes**.

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 Remote Console. Instead, you must use a text editor or a WLST script 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 V1.2 as the supported protocol. Oracle recommends the TLS V.1.2 protocol for the best security. TLS1 configures the channel to send and accept only TLS V1.2 messages. Peers must respond with a TLS V1.2 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 the following example.

Example 1-4 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-properties>
</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 system, that IP address cannot be used when populating **Via**, **Contact**, and **Record-Route** headers.

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 SIP channel and specify **0.0.0.0** (or `::` for IPv6 networks) as the listen address. 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 host name of the server, and localhost. If multiple addresses are mapped to the host name, 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 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, 2024 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.

Example 1-5 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>
```

Configuring Listen Addresses for Servers

Each server in the domain is a member in the Coherence cluster, and the default Coherence configuration uses a generated well-known address list based on server listen addresses. You must use explicit listen addresses with the domain servers for Coherence to correctly form a cluster.You can set up explicit listen addresses using the domain creation wizard or, after creating a domain, by using the Remote Console and following these instructions:

1. From the Remote Console, select **Environment**, and then **Servers**.
2. Select your server to configure.
3. In the **General** tab, enter a unique DNS name or IP address in the Listen Address field.
4. Click **Save**.
5. Click the shopping cart and click **Commit Changes**.
6. Restart the server.

Configuring Coherence Cluster Addressing

If you do not want to use explicit listen addresses with domain servers or want to isolate Coherence cluster communication to its own network, you can configure Coherence cluster addressing to use it's own addressing scheme, using one of the following cluster modes.

- Multicast with multicast address, port and time to live. Multicast communication can make more efficient use of the network in some circumstances, but also might not work in all environments.
- Unicast addressing, specifying explicit well-known addresses (WKAs) and explicit Unicast listen ports for servers.

The default setting is Unicast addressing together with a well-known address list generated from the domain server listen addresses

For more details, see "Configuring and Managing Coherence Clusters" in *Administering Clusters for Oracle WebLogic Server*.

Configuring Maximum Content Length

By default, the maximum value for the Content-Length header is 327675 (5 * 65535). To set a different value, set the value of the `wlss.max.content.size` property.

Using the Engine Cache

This chapter describes how to enable the Oracle Communications Converged Application Server engine cache for improved performance with SIP-aware load balancers.

Overview of Engine Caching

A Converged Application Server engine cluster manages call-state data in several partitions in the memory of each engine server. Each call-state entry resides in one such partition on a specific engine server in the cluster. In many cases the engine server requesting the call-state entry is not the same engine server where it is stored. Engine servers fetch and write data in the SIP call-state store as necessary. Each call state data partition can have one or more backup copies in another server to provide automatic failover in the event that a SIP call-state store server fails or shuts down for some reason.

Converged Application Server also provides the option for engine servers to cache a portion of the call-state data locally. When a local cache is used, an engine server first checks its local cache. If the cache contains the required data, and the local copy of the data is up-to-date (compared to the SIP call-state store copy), the engine locks the call state in the SIP call-state store 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 call-state store.

The engine cache stores only the call state data that has been most recently used by engine servers. Call state data is moved into an engine's local cache as necessary 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 cache is not configurable.

Using a local cache is most beneficial when a SIP-aware load balancer manages requests to the engine cluster. With a SIP-aware load balancer, all of the requests for an established call are directed to the same engine 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 severs (having different cache contents).

Configuring Engine Caching

By default, engine caching is enabled. To disable partial caching of call state data in the engine, 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 cache is not configurable.

Monitoring and Tuning Cache Performance

The **SipPerformanceRuntime** MBean monitors the behavior of the engine cache.

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 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**). The actual value depends on the call state size used by applications and the size of the applications themselves.

In addition, keep the following points in mind when using engine caching:

1. The engine cache is less useful if SIP aware load balancers are not used.
2. The engine cache is *not* used for timer processing, so if an application fires many timers, cache benefits decrease.
3. The cache alters the garbage collection characteristics of the engine, since there is more long-lived state.

For SIP performance monitoring information, see Converged Application Server Monitoring and Overload Protection.

For more information on the methods of the **SipPerformanceRuntime** MBean, see its interface description in the **com.bea.wcp.sip.managment.runtime** package in the *Oracle Converged Application Server Java API Reference*.

Configuring Coherence

This chapter describes the implementation and configuration of Oracle Coherence in Oracle Converged Application Server.

Converged Application Server uses Coherence for the following purposes:

- Cluster-wide engine communication and state management
- Application call-state storage and management for concurrent SIP calls

About Coherence Engine Communication and State Management

The Domain Creation Wizard automatically creates a default Coherence cluster for managing Converged Application Server information when it sets up new domains. The default cluster includes the engine servers and the administrative server in your environment.

Engine Communication and State Management

The Domain Creation Wizard automatically creates a default Coherence cluster for managing Converged Application Server information when it sets up new domains. The default cluster includes the engine servers and the administrative server in your environment.

Use the Remote Console to Create a Coherence Cluster.

1. In the **Edit Tree**, go to **Environment**, then **Coherence Clusters**.
2. Click **New**.
3. Enter a name for the Coherence cluster and click **Create**.
4. Select a **Clustering Mode** and then adjust the Coherence general properties according to the clustering mode you selected.
5. **Optional:** If you want to specify operational settings that are not available through the provided MBeans, you can upload a cluster configuration file with your supplemental settings. Click **Import Configuration**, then, in the **Custom Cluster Configuration File Name** dialog box, enter the location of the cluster configuration file, relative to the domain configuration directory.

   > **Note**
   >
   > Avoid configuring the same operational settings in both an external cluster configuration file and through the MBeans.

6. Click **Save**.

Each engine server and the Administration server acts as a managed Coherence server.

To configure Coherence settings for individual engine servers and the Administration Server:

1. In the **Edit Tree**, go to **Environment**, then **Coherence Clusters**.

2. Select your cluster to configure.
3. Configure the Coherence parameters.
4. Click **Save**.
5. Click the shopping cart and then click **Commit Changes**.

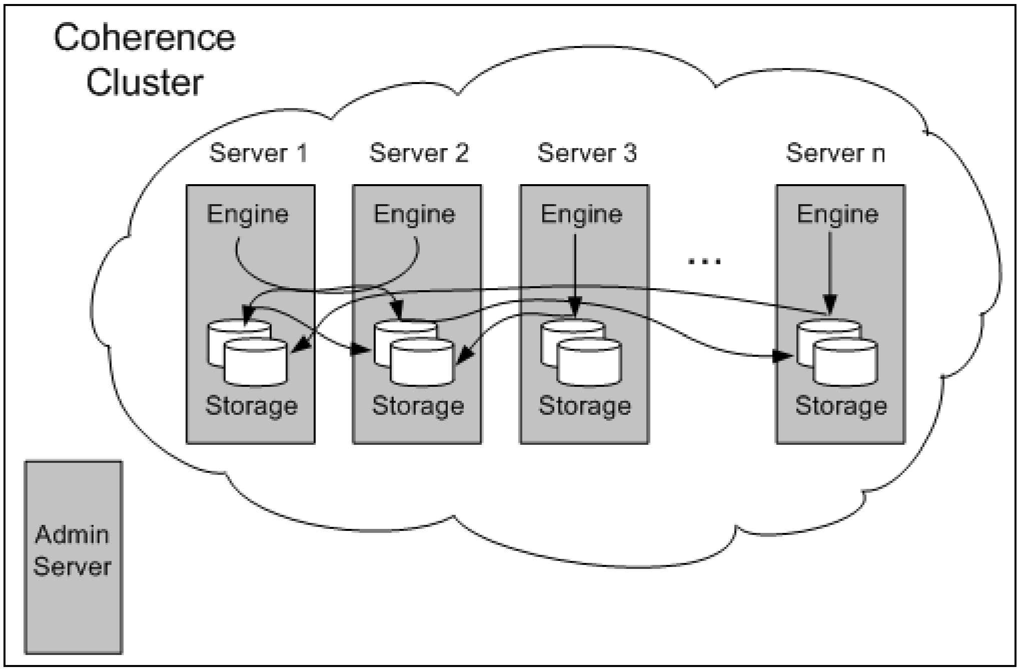
About Call-State Storage and Management for SIP Calls

The Coherence call-state storage facility for Converged Application Server is built on the distributed cache service of WebLogic Server. In each managed server in the domain cluster, Coherence combines logic and processing with state-storage data. Coherence writes data to the primary partition cache-storage server and it, in turn, writes a backup copy to the configured number of backup copies.

See "Understanding Distributed Caches" in *Developing Applications with Oracle Coherence* for an explanation of Coherence distributed caches.

The image below illustrates an administration server with a Coherence cluster for call-state storage.

Figure 1-1 Coherence Cluster for Call-State Storage



The Coherence call-state storage facility includes the following features:

- Built-in support for dynamically adding or removing nodes
- Partitions that migrate dynamically, eliminating the need to configure replica servers and their partitions
- Enhanced data serialization with Portable Object Format (PoF)
- Proven node death detection for fail-over and split brain handling
- Flexible configuration

- Advanced network protocol that leverages UDP and supports multi-cast to optimize network usage
- Graceful migration of partitions from one node to another during startup and shutdown, limiting the impact on ongoing traffic and reducing the risk of overload

Configuring Coherence Call-State Storage

The **coherence.xml** custom resource file specifies a subset of the configuration options that control call-state storage. The **config.xml** file specifies the custom resource file as ***$domain_home*/config/custom/coherence.xml**. The entry in the **config.xml** file looks like this:

```
<custom-resource>
    <name>coherence</name>
    <target>BEA_ENGINE_TIER_CLUST</target>
    <descriptor-file-name>custom/coherence.xml</descriptor-file-name>
    <resource-class>com.bea.wcp.sip.management.descriptor.
        resource.CoherenceStorageResource</resource-class>
    <descriptor-bean-class>oracle.occas.management.descriptor.beans.
        storage.CoherenceStorageBean</descriptor-bean-class>
</custom-resource>
```

The following parameters describe the **coherence.xml** file. They define a default call-state storage domain.

```
<?xml version='1.0' encoding='UTF-8'?>
<coherence-storage>
  <cache-config>
    <thread-count>20</thread-count>
    <partition-count>257</partition-count>
  </cache-config>
</coherence-storage>
```

Modifying the Call-State Storage Configuration

To view and modify SIP call-state storage parameters:

1. From the **Edit Tree** of the Remote Console, select **Custom Resource**, and then **coherence**, and then **Call State Storage**, and then **Cache Config**.
2. Enter values for **Thread Count** or **Partition Count** or both.
3. Click **Save**.

The following table describes the rules that apply to the Thread Count and Partition Count parameters:

Table 1-2 Call State Storage Configuration Parameters

Parameter	Type	Validation Rule	Restart Server?	Notes
Thread Count	integer	-1 to 32767	Yes	-1 = caller thread; 0 = service thread; otherwise, thread pool
Partition Count	integer	1 to 32767	Yes (all at the same time)	Must be prime number

The values are saved in the *domain_home***/config/custom/coherence.xml** file where *domain_home* is the root directory of the Converged Application Server domain.

You can also set call-state storage parameters using WLST. See "Using WLST (JMX) to Configure Converged Application Server" for more information.

Monitoring Coherence Call-State Storage

This functionality is not supported in the present release.

Use the Coherence CLI to get details about Coherence.

Coherence CLI

The Coherence command line interface, cohctl, is a lightweight tool, in the tradition of tools such as kubectl, which can be scripted or used interactively to manage and monitor Coherence clusters. You can use cohctl to view cluster information such as services, caches, members, etc, as well as perform various management operations against clusters.

The Coherence CLI is installable on a variety of platforms and architectures including macOS, Linux and Windows. For more details, see Coherence CLI Installation.

Check the installation process for your operating system. For Linux or macOS, run the following command:

```
curl -sL https://raw.githubusercontent.com/oracle/coherence-cli/main/scripts/
install.sh | bash
```

The 'add cluster' command adds a new connection to a Coherence cluster. For more details, see Clusters.

```
cohctl add cluster local -u http://<IP>:7001/management/coherence/latest/
clusters
```

The Coherence CLI contains many commands to manage and monitor Coherence clusters. Given below are 2 commands that can be helpful in fetching details about the cluster named “local” and the service “callstate”. See the Commands reference.

The output of `cohctl describe cluster local` is shown below.

```
CLUSTER
-------
Cluster Name:       Coherence-Default
Version:            14.1.2.0.0
Cluster TotalSize:  2
License Mode:       Development
Departure Count:    0
Running:            true

MACHINES
--------
MACHINE  PROCESSORS       LOAD  TOTAL MEMORY  FREE MEMORY  % FREE  OS
ARCH     VERSION
n/a              11  1.8305664     36,864 MB       691 MB   1.88%  Mac OS X
aarch64  15.3.2

MEMBERS
-------
```

```
Total cluster members: 2
Storage enabled count: 1
Departure count:       0

Cluster Heap - Total: 1,024 MB Used: 324 MB Available: 700 MB (68.4%)
Storage Heap - Total: 512 MB Used: 169 MB Available: 343 MB (67.0%)


SERVICES
--------
SERVICE NAME                     TYPE              MEMBERS  STATUS HA
STORAGE  SENIOR  PARTITIONS  STATUS
"fabric:FabricPartitionedCache"  DistributedCache        1  ENDANGERED
1       2         257  StatusHA is ENDANGERED
"fabric:FabricReplicatedCache"   ReplicatedCache         1  ENDANGERED
1       2           1  StatusHA is ENDANGERED
InvocationService                Invocation              1  n/a
0       2           -  n/a
ReplicatedCache                  ReplicatedCache         1  ENDANGERED
1       2           1  StatusHA is ENDANGERED
callstate                        DistributedCache        1  ENDANGERED
1       2         257  StatusHA is ENDANGERED
flowstate                        DistributedCache        1  ENDANGERED
1       2         257  StatusHA is ENDANGERED

PERSISTENCE
-----------
Total Active Space Used: 0 MB
Total Backup Space Used: 0 MB

SERVICE NAME                     STORAGE COUNT  PERSISTENCE MODE  ACTIVE
SPACE  BACKUP SPACE  AVG LATENCY  MAX LATENCY  SNAPSHOTS  STATUS
"fabric:FabricPartitionedCache"              1  on-demand                 0
MB          0 MB      0.000ms          0ms          0  Idle
callstate                                    1  on-demand                 0
MB          0 MB      0.000ms          0ms          0  Idle
flowstate                                    1  on-demand                 0
MB          0 MB      0.000ms          0ms          0  Idle

CACHES
------
Total Caches: 6, Total primary storage: 0 MB

SERVICE
CACHE                                                   COUNT  SIZE
"fabric:FabricPartitionedCache"  "fabric-storage_near-occas-stats-
FabricKey:SimpleData"      0  0 MB
"fabric:FabricPartitionedCache"  fabric-ngstats-
SYSTEM                                       0  0 MB
"fabric:FabricReplicatedCache"   fabric-
members                                              0  0 MB
callstate
CallState                                                   0  0 MB
callstate
CallState.idx                                               0  0 MB
callstate
```

```
CallState.meta                                              0  0 MB

TOPICS
------
```

The output of `cohctl describe service callstate` is shown below.

```
SERVICE DETAILS
---------------
Name                                :  callstate
Type                                :  [DistributedCache]
Backup Count                        :  [1]
Backup Count After Writebehind      :  [1]
Event Backlog                       :  0
Event Count                         :  1
Event Interceptor Info              :  [[Interceptors: [EventsHelper]
ExceptionCount: 0 LastException: ]]
Indexing Total Millis               :  7
Join Time                           :  [2025-04-02T10:09:30.494+00:00]
Member Count                        :  [1]
Messages Local                      :  55
Messages Received                   :  55
Messages Sent                       :  64
Outgoing Transfer Count             :  0
Owned Partitions Backup             :  [0]
Owned Partitions Primary            :  [257]
Partitions All                      :  [257]
Partitions Endangered               :  [257]
Partitions Unbalanced               :  [0]
Partitions Vulnerable               :  [257]
Persistence Environment             :
[SafeBerkeleyDBEnvironment(ActiveDirectory=null, SnapshotDirectory=/Users/
srsvn/my_data/Projects/deployment/latest_installation/user_projects/domains/
replicated_domain/coherence/snapshots/Coherence-Default/callstate)]
Persistence Latency Average         :  map[average:0 count:1 max:0 min:0
sum:0]
Persistence Latency Max             :  0
Persistence Mode                    :  [on-demand]
Persistence Snapshot Space Available:  [0]
Persistence Snapshot Space Total    :  [0]
Quorum Status                       :  [allowed-actions=distribution,
restore, read, write, recover]
Refresh Time                        :  [2025-04-02T11:32:04.935+00:00]
Request Average Duration            :  map[average:3 count:1 max:3 min:3
sum:3]
Request Max Duration                :  16
Request Pending Count               :  0
Request Pending Duration            :  0
Request Timeout Count               :  0
Request Timeout Millis              :  [0]
Request Total Count                 :  25
Running                             :  map[true:1]
Senior Member Id                    :  [2]
Statistics                          :  [Cpu=1078ms (0.0%), Messages=55,
Throughput=51.02041msg/sec, AverageActiveThreadCount=2.0179546E-7, Tasks=3,
AverageTaskDuration=0.33333334ms, MaximumBacklog=0]
Status HA                           :  [ENDANGERED]
Storage Enabled                     :  map[true:1]
Task Average Duration               :  map[average:0.3333333432674408 count:1
max:0.3333333432674408 min:0.3333333432674408 sum:0.3333333432674408]
Task Backlog                        :  0
Task Count                          :  3
Task Hung Count                     :  0
Task Hung Duration                  :  0
Task Hung Threshold Millis          :  [30000]
Task Max Backlog                    :  0
Task Timeout Count                  :  0
Task Timeout Millis                 :  [3000]
Thread Abandoned Count              :  0
Thread Average Active Count         :  map[average:2.0179545856535697e-07
count:1 max:2.0179545856535697e-07 min:2.0179545856535697e-07
sum:2.0179545856535697e-07]
Thread Count                        :  [20]
Thread Count Update Time            :  [1970-01-01T00:00:00.000+00:00]
Thread Idle Count                   :  20
Thread Pool Sizing Enabled          :  map[false:1]
Transport Backlogged Connection List:  [[]]
Transport Status                    :  [SocketMessageBus(tmb://
192.168.0.103:63586.56838, state=OPEN, connections active=1/1)]

SERVICE MEMBERS
---------------
NODE ID  THREADS  IDLE  THREAD UTIL  MIN THREADS  MAX THREADS
      2       20    20        0.00%           -1           -1

SERVICE CACHES
--------------
Total Caches: 3, Total primary storage: 0 MB

SERVICE    CACHE           COUNT  SIZE
callstate  CallState           0  0 MB
callstate  CallState.idx       0  0 MB
callstate  CallState.meta      0  0 MB

PERSISTENCE FOR SERVICE
-----------------------
Total Active Space Used: 0 MB
Total Backup Space Used: 0 MB

NODE ID  PERSISTENCE MODE  ACTIVE SPACE  BACKUP SPACE  AVG LATENCY  MAX
LATENCY
      2  on-demand                 0 MB          0 MB      0.000ms
0ms

PERSISTENCE COORDINATOR
-----------------------
Coordinator Id  :  2
Idle            :  true
Operation Status:  Idle
Snapshots       :  []
```

```
DISTRIBUTION INFORMATION
------------------------
Scheduled Distributions:  No distributions are currently scheduled for this
service.

PARTITION INFORMATION
---------------------
Service                     :  callstate
Strategy Name               :  SimpleAssignmentStrategy
Average Partition Size KB   :  0
Average Storage Size KB     :  0
Backup Count                :  1
Cluster                     :  Coherence-Default
Coordinator Id              :  2
Fair Share Backup           :  0
Fair Share Primary          :  0
HA Status                   :  ENDANGERED
HA Status Code              :  0
HA Target                   :  There are no distribution analysis results.
Last Analysis Time          :  1970-01-01T05:30:00.000+05:30
Location                    :  AdminServer
Max Load Node Id            :  0
Max Partition Size KB       :  0
Max Storage Size KB         :  0
Partition Count             :  257
Remaining Distribution Count:  0
Responsibility              :  DistributionCoordinator
Service Machine Count       :  0
Service Node Count          :  1
Service Rack Count          :  0
Service Site Count          :  0
Type                        :  PartitionAssignment
```

Configuring Server Failure Detection

This chapter describes how to configure Oracle Communications Converged Application Server to improve failover performance when a server becomes physically disconnected from the network.

Overview of Failover Detection

To achieve a highly-available production system, the Converged Application Server uses the Oracle Coherence distributed cache service to retrieve and write call-state data. The cache service consists of a number of partitions that are spread across the servers that are running in the cluster. Each partition has a primary copy of call-state storage assigned to one server in the cluster, and a backup copy assigned to another server in the cluster. This means that a call state that is required to process a request may reside on a remote server and possibly even a remote machine.

The Converged Application Server architecture depends on the Coherence cache service to detect when a server has failed or becomes disconnected. When an engine cannot access or write call-state data because a server is unavailable, the Coherence cache service detects this and reassigns the lost server's partitions to another server in the cluster and ensures a new backup copy is made available on a different server, if one is running.

Coherence Cluster Overview

The Coherence cache service uses its own cluster communication protocol, known as Tangosol Cluster Management Protocol (TCMP), to invoke remote servers, detect server failure and achieve high availability. This protocol uses an optimized algorithm to quickly detect that a server has become physically disconnected from the network. This algorithm, and the configuration options that are available to modify its behavior, are described in detail in the Oracle Coherence documentation. See the following documentation for more information on Coherence and its distributed cache service.

- "Introduction to Coherence Clusters" in *Developing Applications with Oracle Coherence*
- "Understanding Distributed Caches" in *Developing Applications with Oracle Coherence*

See "Configuring Coherence" and "SIP Coherence Configuration Reference (coherence.xml)" for additional information on configuring Coherence for the Converged Application Server.

Split-Brain Handling

The Converged Application Server relies to a large extent on Oracle Coherence to detect and handle a split-brain condition. A split-brain condition can occur, for example, when connectivity is restored between two or more parts of a cluster that had been isolated from each other.

After a split-brain failure, causing two or more network partitions to be created, each such network partition will contain a set of engines that will reform themselves into a smaller cluster (or possibly a single server waiting to form a new cluster with newly started members).

Each such cluster will, while the network still is partitioned, continue to operate as if the other engines have been shut down. The clusters will now have promoted the oldest member in the cluster to a cluster senior member, responsible for managing the cluster state.

When the network is repaired, and all clusters become aware of each other again, the senior members in each cluster will communicate to decide which single cluster should survive. This may in certain situations take a couple of minutes before reaching a final conclusion, but will eventually resolve as follows:

1. If one cluster is larger than all of the others, it will survive and all other engines will be shut down.
2. If two or more equally large clusters exist that are larger than all the other clusters, the cluster with the older senior member will survive and all other engines will be shut down.

When Coherence detects a split-brain condition, its behavior is controlled primarily through the options related to death detection in the cluster-related configuration. For more information see "Configuring Death Detection" in *Developing Applications with Oracle Coherence*.

Coherence Configuration

You can use the following three mechanisms to modify Coherence configuration options:

- The default Coherence cluster configuration file
- The system properties
- The **tangosol-coherence-override.xml** file

> ⚠ **Warning**
>
> No servers in the domain can be running when you make changes to the Coherence configuration. Also, the configuration must be the same for all servers in the domain or unexpected behavior can result.

Cluster Configuration File

The default Coherence cluster configuration file, **Custom-Default.xml**, resides in the following location:

```
$DOMAIN_HOME/config/coherence/Coherence-Default/
```

where *$DOMAIN_HOME* is the root directory for the domain.

The following table describes the default configuration options that you can specify.

Table 1-3 Coherence Cluster Configuration File Options

Option	Element Name	System Property Name	Default Value
TCP-ring IP-timeout	<tcp-ring-listener><ip-timeout>	tangosol.coherence.ipmonitor.pingtimeout	5
TCP-ring IP-attempts	<tcp-ring-listener><ip-attempts	tangosol.coherence.ipmonitor.pingtattempts	2
Service Guardian Timeout	<service-guardian><timeout-milliseconds>	tangosol.coherence.guard.timeout	305000
Packet Delivery Timeout	<packet-delivery><timeout-milliseconds>	tangosol.coherence.packet.timeout	300000

You can override these default configuration options either by modifying the corresponding system properties or creating an override configuration file, called **tangosol-coherence-override.xml**, which you add to the system CLASSPATH variable on all servers.

See the following Coherence documentation for information on which configuration options you can override and for information on how to use the override configuration option:

- Configuring a Coherence Clusterin *Administering Clusters for Oracle WebLogic Server*
- Death Detection Recommendations in *Administering Oracle Coherence*
- Configuring Death Detection in *Developing Applications with Oracle Coherence*
- Understanding the XML Overrride Feature in *Developing Applications with Oracle Coherence*
- Coherence Operational Configuration Reference in *Developing Applications with Oracle Coherence*

Avoiding and Recovering From Server Failures

This chapter describes the Oracle Communications Converged Application Server failure prevention and recovery features, and includes the configuration artifacts that are required to restore different portions of a Converged Application Server domain.

Failure Prevention and Automatic Recovery Features

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 malfunctions, 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 if an individual server fails.

Converged Application Server, and the underlying WebLogic Server platform, provide many features that protect against server failures. In a production system, use all available features to ensure uninterrupted service.

High Availability

High availability refers to a system design that eliminates or minimizes the amount of time that a system is inaccessible due to some type of system failure.

Converged Application Server achieves high availability primarily due to the features of the underlying Weblogic Server platform. These features include:

- WebLogic Server clusters that distribute the work load among the multiple instances of WebLogic Server running on the nodes in the cluster. In the event of failure, the session state of the failed WebLogic Server is available to other node that can continue the work. If the cluster is configured correctly, services can also migrate to another node in the event of failure. See Understanding Weblogic Server Clustering in *Administering Clusters for Oracle WebLogic Server* for more information.
- Coherence clusters that distribute data across members to ensure that data is always available. See Configuring and Managing Coherence Clusters in *Administering Clusters for Oracle WebLogic Server* for more information.
- Overload protection that enables WebLogic Server to detect and recover from overload conditions. See Avoiding and Managing Overload in *Administering Server Environments* for more information.
- Network channels that segregate traffic by type to use resources effectively. See Configuring Network Resources in *Administering Server Environments* for more information
- Work Managers that optimize and prioritize work based on rules and performance statistics. See Using Work Managers to Optimize Scheduled Work in *Administering Server Environments* for more information.

You can also use virtual machines (VMs) to mitigate system failure. An individual server has multiple points of potential failure, including CPU, RAM, network ports, and disk drives. A virtual machine, on the other hand, can satisfy its resource requirements from a pool of hardware resources so that a physical disk failure does not result in a failure of the virtual disk.

The virtual machine simply employs another available disk drive to compensate for the one that failed.

Overload Protection

Converged Application Server implements an overload framework which supports plug-in statistics collectors, plug-in event handlers, as well as multiple threshold settings and statistics collection algorithms. For more information, see "About Converged Application Server Overload Protection".

For general information on overload protection, see Avoiding and Managing Overload in *Administering Server Environments for Oracle WebLogic Server* for more information.

Redundancy and Failover for Clustered Services

You can increase the reliability and availability of your applications by using multiple servers and partitions in a dedicated cluster.

Server partitions store redundant copies of call state information, and automatically failover to one another should a partition or server fail.

See *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 restart servers that have failed. This improves the overall reliability of a domain, and requires no direct intervention from an administrator. For more information, see Using Node Manager to Control Servers in the *Administering Node Manager for Oracle WebLogic Server*.

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 without contacting its Administration Server to check for configuration updates is running in **Managed Server Independence (MSI)** mode. By default, MSI mode is enabled.

Geographic Redundancy for Regional Site Failures

In addition to server-level redundancy and failover capabilities, you can configure peer sites to protect against catastrophic failures, such as power outages, that can affect an entire domain. This configuration enables you to failover from one geographical site to another, avoiding complete service outages. For more information, see Configuring Geographically-Redundant Installations.

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. 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). Use the configuration archives 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 and 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. From the Edit Tree of the Remote Console, click **Environment**, and then **Domain**, and then under the General tab, select **Show Advanced Fields**.
2. Select **Configuration Archive Enabled**.
3. In the **Archive Configuration Count** box, enter the maximum number of configuration file revisions to save.
4. Click **Save**.
5. Restart the server.

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 creating a full snapshot of the domain at regular intervals. For example, you might want to create a snapshot when the following events occur:

- You first deploy the production system
- You add or remove deployed applications
- The configuration is tuned for performance
- Any other permanent change is made.

> **Note**
>
> The domain directory is present on the Administration Server and each Managed Server but the Administration Server has the master copy, which you must back up. You do not need to back up any files on a Managed Server.

The WebLogic `pack` command creates a template archive file (.jar) based on an existing WebLogic domain. For example, the following command creates a template file called **C:\oracle\user_templates\mydomain.jar**.

```
pack -domain=C:\oracle\user_projects\domains\mydomain -
template=C:\oracle\user_templates\mydomain.jar -template_name="My WebLogic Domain"
```

The name of the template is My WebLogic Domain.

See Creating Templates and Domains Using the Pack and Unpack Commands for information on using the `pack` and `unpack` commands.

Store the new archive in a source control system, preserving earlier versions should you need to restore the domain to an earlier point in time.

Backing Up Logging Servlet Applications

If you use Converged Application Server logging Servlets (see "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_home***\servers\AdminServer\data\ldap**

where *domain_home* is the domain's root directory and **servers\AdminServer\data\ldap** is the directory in which the Administration Server stores run-time 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 "Back Up LDAP Repository" section in *Administering Server Startup and Shutdown for Oracle WebLogic Server*.

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 and NAT configuration settings.
- NTP client configuration scripts used to synchronize the system clocks of engine servers.
- Host configuration files for each Converged Application Server system (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 without 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 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 system it was running on when the domain was started.

Restarting an Administration Server on the Same System

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**
>
> Ensure 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 their running state. 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 malfunction, 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 run-time changes to attributes configurable while a server is running (dynamic attributes).

Restarting an Administration Server on Another System

If a system malfunction prevents you from restarting the Administration Server on the same system, you can recover management of the running Managed Servers as follows:

1. Install the Converged Application Server software on the new system (if this has not already been done).apply any patches that had been applied to the failed server.
2. Apply any patches that had been applied to the failed server.
3. Use the `unpack` command to create a WebLogic domain from the template that you created when you backed up the domain. See Storing the Domain Configuration Offline. 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.
4. Make your configuration and security data available to the new administration system by copying them from backups or by using a shared disk. Refer to Backing Up Security Data.
5. Restart the Administration Server on the new system.
   Ensure 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 system 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. You must configure Node Manager and the Managed Server to support automated restarts, as described in the discussion on "How Node Manager Restarts a Managed Server" in the *Administering Node Manager for Oracle WebLogic Server*.

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.

To start 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".

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 Coherence 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 Coherence cache 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 Coherence cache 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 Coherence cache's in-memory replication functionality. To improve latency performance when using an RDBMS, the Coherence cache 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 Coherence cache selects one engine to process all call state writes (or deletes) to the database. Any available server 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 cache, if your domain uses a SIP-aware load balancer to manage connections to the engine tier. See "Using the Engine 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 Coherence cache 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 Coherence cache.

> **Note**
>
> RDBMS persistence is designed only to reduce the RAM requirements in the Coherence cache for large, long-lived call states. The persisted data cannot be used to restore a failed engine.

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.

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 contents of the `callstate.sql` SQL script are shown below.

Example 1-6 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. Connect to the database using the user name, password and database name that you specified when you installed the database software. 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
   ```

Configure JDBC Resources

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

1. From the **Edit Tree** of the Remote Console, click **Services**, and then **Data Sources**.
2. Click **New** and provide a name for this data source.
3. Set **JNDI Name** to wlss.callstate.datasource.
4. Under Targets, select the name of your SIP engine cluster (for example, BEA_ENGINE_TIER_CLUST) and move it into the Chosen list.
5. Set **Data Source Type** to **Generic Data Source**.
6. Fill in the database fields according to your previously created database.
7. 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. For more information on JDBC drivers, see "Using JDBC Drivers with WebLogic Server" in *Administering JDBC Data Sources for WebLogic Server*.
8. Click **Create**.
9. Under the Transaction tab, configure any transaction properties required for your database. For more information on JDBC transaction options, see "Configure Transaction Options" in *Administering JDBC Data Sources for WebLogic Server*.
10. Under the Connection Pool tab, fill in the fields using connection information for the database you want to use. For more information, see "Configure Connection Properties" in *Administering JDBC Data Sources for WebLogic Server*.
11. Click **Save**, click the shopping cart, and then **Commit Changes**.
12. Restart the server.

Configuring Persistence Options (Primary and Secondary Sites)

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

1. From the **Edit Tree** of Remote Console, select **Custom Resources**, and then **sipserver**, and then **SIP Server**, and then **Persistence**.
2. Configure the Persistence attributes as follows:
   - **DB Enabled**: Check to enable call states to be stored in an RDBMS.
   - **Default Handling**: Select **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.
     For information on configuring geographical redundancy, see "Configuring Geographically-Redundant Installations".

3. Click **Save**, and then the shopping cart, and then **Commit Changes**.

Using Persistence Hints in SIP Applications

Converged Application Server provides a simple API to provide "hints" as to when the Coherence cache 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 below.

Example 1-7 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();
}
```

Configuring Geographically-Redundant Installations

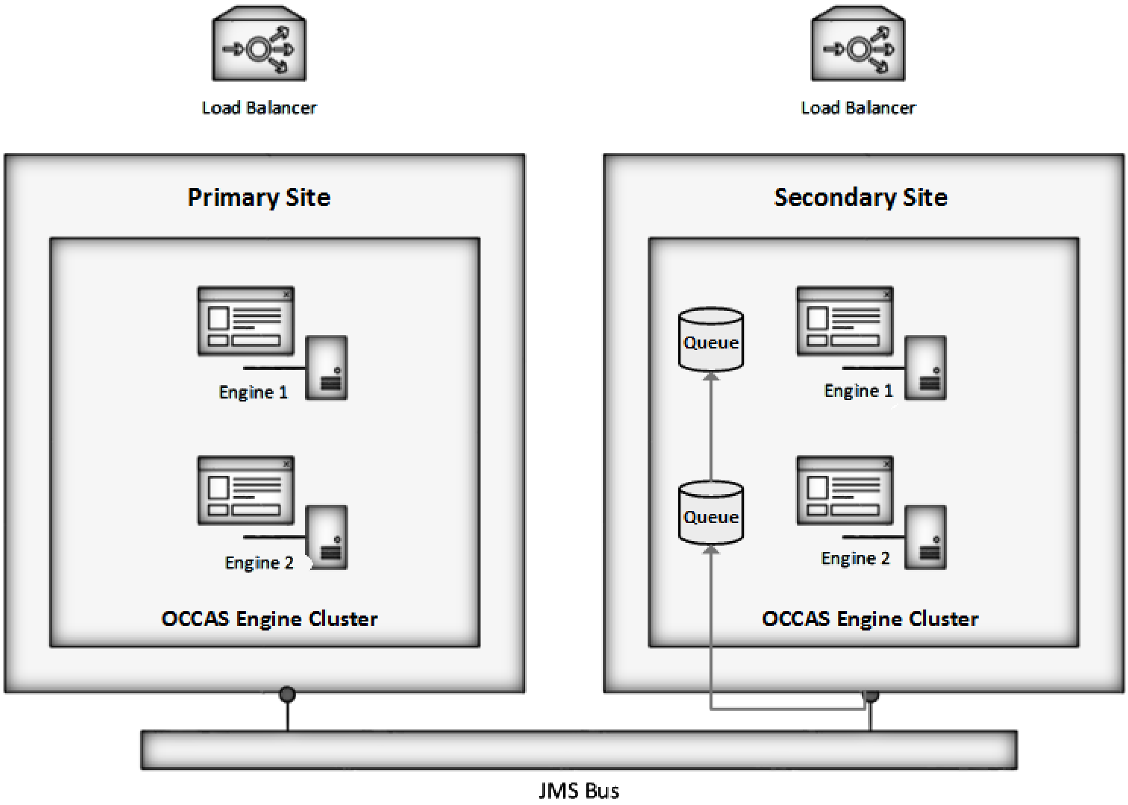
This chapter describes how to replicate call state transactions across multiple, regional Oracle Communications Converged Application Server installations.

Geographic Redundancy

Geographic 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 1-2 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 Coherence cache, and becomes eligible for replication to a secondary site.
3. A single engine in the Coherence cache 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 1-4 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 load balancing 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 Coherence cache, 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 engine in the Coherence cache 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.

Table 1-4 (Cont.) Geographic Redundancy Flow

Normal Operation	Failover
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.
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 ~20KB on the wire.

- 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.
- 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).
- `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).

Coherence Call State Storage

The call state can be distributed across multiple partitions spread across different servers within the cluster.

Call states are assigned to a partition based on the call state's ID. By default there is only 1 backup, but you can enable a second backup partition by adding `-Dwlss.cache.backupcount=2` to the `startWebLogic.sh` script in the `bin` directory.

To help identify your cluster member's identity, you can set the site name and rack name. These parameters are under **Environment**, and then **Servers**, and then your server, and then the **Advanced** tab, and then the **Coherence** subtab. After setting the site name or rack name, restart the WebLogic server.

- Site Name – the name of the geographic site that hosts the cluster member. The server's domain name is used if no name is specified. For WAN clustering, this value identifies the datacenter where the member is located.
- Rack Name – the name of the location within a geographic site that the member is hosted at and is often a cage, rack, or bladeframe identifier.

Both the site name and the rack name can be used as the basis for intelligent routing, load balancing, and disaster recovery planning (that is, the explicit backing up of data on separate geographic sites). The site name and rack name also help determine where to back up data when using distributed caching and the default partition assignment strategy. Lastly, the names are useful for displaying management information (for example, JMX) and interpreting log entries.

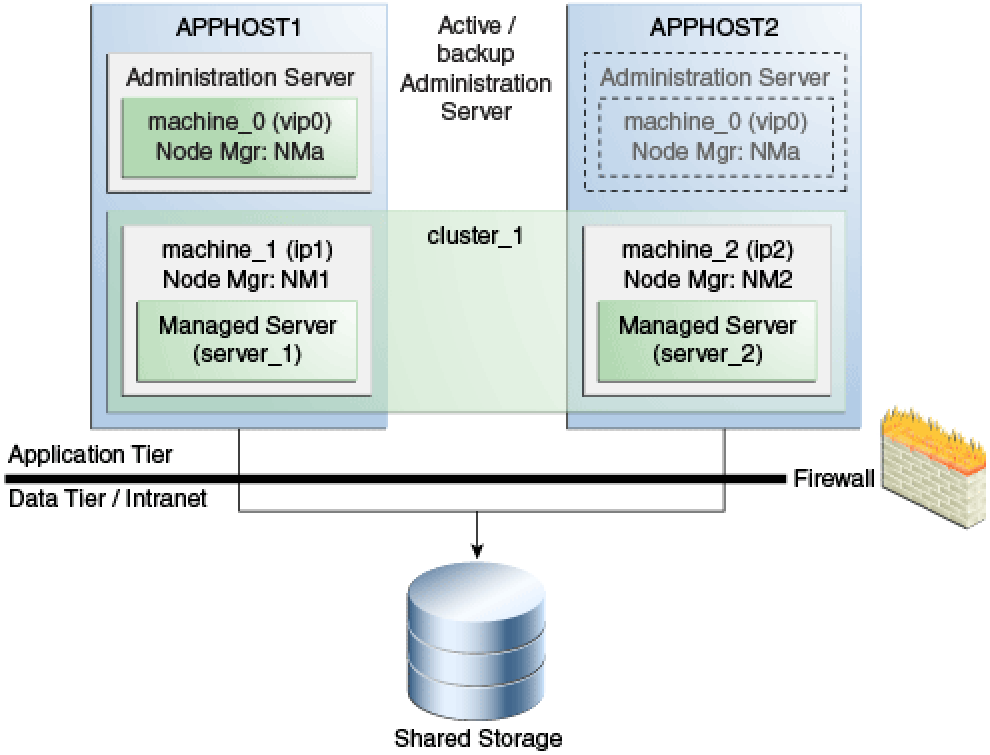
Admin Server HA

For an Admin Server to be highly available, use the following best practices:

- Use a floating virtual IP.
- Set up automatic failover with Node Manager.
- Use shared storage for the domain home directories of the active and standby nodes.

These practices allow for the quickest failover because there is no dependency on DNS.

Figure 1-3 Admin Server HA



Using Geographically-Redundant SIP Engines

The basic call state replication functionality available in the Coherence cache provides excellent failover capabilities for a single site installation. However, the active replication performed within the Coherence cache requires high network bandwidth in order to meet the latency performance needs of most production networks. This bandwidth requirement for Coherence Cluster spanning multiple sites may be less economical for bare metal deployments and more economical for cloud deployments. Refer to the Coherence documentation to see the data replication methods that Coherence supports.

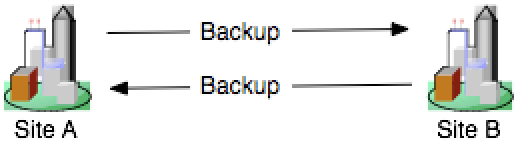
The Converged Application Server geographic persistence feature enables you to replicate call state transactions across multiple Converged Application Server installations (multiple Administrative domains or "sites"). A geographically-redundant configuration minimizes dropped calls in the event of a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

Oracle recommends using the JMS Queue and RDBMS methods for replicating data between Coherence clusters.i

Example Domain Configurations

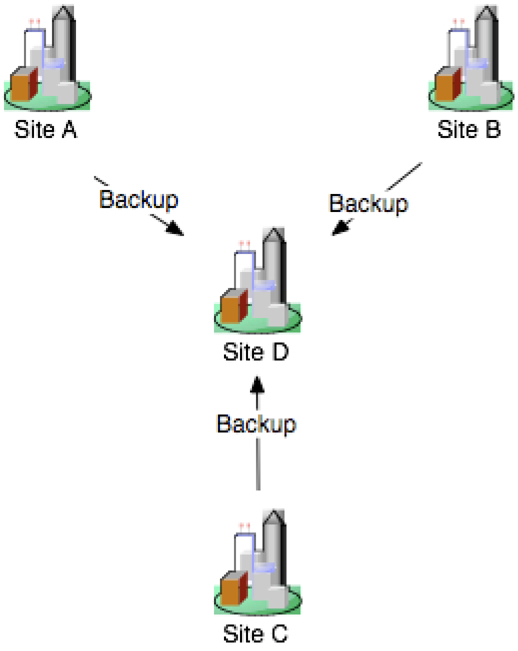
A secondary Converged Application Server domain that persists data from another domain may itself process SIP traffic, or it may exist solely as an active standby domain. In the most common configuration, two sites are configured to replicate each other's call state data, with each site processing its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

Figure 1-4 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 1-5 Alternate Geographically-Redundant Configuration



When using geographic persistence, a single engine 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, engines use a message listener to monitor the distributed queue to receive messages and write the data to its own Coherence cache. If the secondary site uses an RDBMS to store long-lived call states (recommended), then the call state data entries are written into the RDBMS and removed from the in-memory call state cache.

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.

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 1-5 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 JMS Servers and modules required for replicating data. 4. Configure persistence options to: define the unique regional site ID; identify the secondary site's URL; and enable replication hints. 5. Optionally configure cross domain security settings.	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. 5. Optionally configure cross domain security settings.

> **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.

Follow the instructions in "Configuring Geographic Redundancy" to create the resources.

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 geographic 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. Both the primary and secondary sites must have available JMS Servers and specific JMS module resources in order to replicate call state data between sites.
4. Optionally, configure cross domain security for both primary and secondary sites.

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 Site Only)

The primary site must configure the correct persistence settings in order to enable replication for geographic redundancy. Follow these steps to configure persistence:

1. From the **Edit Tree** of the Remote Console, click **Custom Resources**, and then **sipserver**, and then **SIP Server**, and then **Persistence**.
2. Configure the Persistence attributes as follows:
   - **DB Enabled**: Check to enable call states to be stored in an RDBMS.
   - **Geo Enabled**: Check to enable geographic redundancy.
   - **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**: This setting is deprecated. Leave it blank.

3. Click **Save**, and then the shopping cart, and then **Commit Changes**.
4. If you've set **Geo Site ID**, restart the server.

Configuring JMS Resources Options (Primary Site Only)

Follow these steps to configure JMS resources for the primary site only:

Create a JMS Server

1. From the **Edit Tree** of the Remote Console, click **Services**, and then **JMS Servers**, and then **New**.
2. Enter a unique name for the JMS Server and click **Create**.
3. In the **Target** tab, select the name of the engine cluster in the installation.
4. Click **Save**, and then the shopping cart, and then **Commit Changes**.

Create a JMS Module

1. Click **Services**, and then **JMS Modules**, and then **New**.
2. Enter a **Name** for the new JMS Module, for example *geo-redundancy*, and click **Create**.
3. In the **Targets** tab, select all the servers in the cluster, add them to the Chosen field.
4. Click **Save**, and then the shopping cart, and then **Commit Changes**.

Create a Connection Factory

1. Under **JMS Modules**, and your new module, select the **Connection Factory** and then **New**.
   If your new module does not appear under JMS Modules, refresh the **Services** dropdown by clicking away from **Services** and clicking back on **Services**.
2. Enter a **Name** for the connection factory, click **Create**, and enter *wlss.callstate.backup.site.connection.factory* as the **JNDI Name**.
3. Click **Save**, click the shopping cart, and click **Commit Changes**.

Create a Foreign Server

1. In the left column, select the **Foreign Server** resource type and click **New**.

   > **Note**
   >
   > Your Foreign Server must be targeted to all servers in the engine cluster.

2. Give the server a name, click **Create**, and enter a value for the **JNDI Connection URL** field. Use a comma-separated list for a cluster. For example, for a single server *t3://site-2-admin:7001* or for a cluster *t3://site-2-engine1:8001,site-2-engine2:8051*.
3. Click **Save**, click the shopping cart, and click **Commit Changes**.
4. Click **Services**, and then **JMS Modules**, and then your module, and then **Foreign Servers**, and then your foreign server, and then **Foreign Destinations**, and then click **New**.
5. Enter a Name for the foreign destination, and enter *wlss.callstate.backup.site.peer.queue* for the **Local JNDI Name**, and *wlss.callstate.backup.site.queue* for the **Remote JNDI Name**, and then click **Create**.

6. Click **Save**, click the shopping cart, and click **Commit Changes**.
7. In the **Foreign Connection Factories** resource, click **New**.
8. Enter a Name for the foreign connection factory, and enter *wlss.callstate.backup.site.peer.connection.factory* for the **Local JNDI Name**, and *wlss.callstate.backup.site.connection.factory* for the **Remote JNDI Name**, and click **Create**.
9. Click **Save**, click the shopping cart, and click **Commit Changes**.

Create an Unrestricted Connection Factory

1. Click **JMS Modules**, click your module name, and click **Connection Factories**.
2. Click **New**, give your new connection factory a name, and click **Create**.
3. In the **JNDI Name**, enter the name `wlss.callstate.backup.site.queue`.
4. Click **Save** and navigate to the **Client** tab.
5. Set the **Client ID Policy** to **Unrestricted**.
6. Click **Save**, click the shopping cart, and click **Commit Changes**.

Configuring Persistence Options (Secondary Sites)

The secondary site must configure the correct persistence settings in order to enable replication for geographic redundancy. Follow these steps to configure persistence:

1. From the **Edit Tree** of Remote Console, click **Custom Resources**, and then **sipserver**, and then **SIP Server**, and then **Persistence**.
2. Configure the Persistence attributes as follows:
   - **DB Enabled**: Check to enable call states to be stored in an RDBMS. For information on configuring RDBMS call state storage, see "Storing Long-Lived Call State Data in an RDBMS".
   - **Geo Enabled**: Check to enable geographic redundancy.
   - **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**: This setting is deprecated. Leave it blank.
3. Click **Save**, click the shopping cart, and then click **Commit Changes**.

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. In the **Edit Tree** of the Remote Console, click **Services**, and then **JMS Servers**, and then **New**.
2. Enter a unique name for the JMS Server, and click **Create**.

3. In the **Target** tab, select the name of the engine cluster in the installation.
4. Click **Save**, and then click the shopping cart, and then click **Commit Changes**.
5. Click **Services**, then **JMS Modules**, and then **New**.
6. Enter a name for the JMS module and click **Create**.
7. In the **Target** tab, select the name of the engine cluster in the installation.
8. Click **Save**, and then click the shopping cart, and then click **Commit Changes**.
9. Select **Services**, and then **JMS Modules**, and then the new JMS module you created.
10. Click **Connection Factory**, and then click **New**, enter a name, and click **Create**.
11. Set the **JNDI Name** field to `wlss.callstate.backup.site.connection.factory`, and click **Save**.
12. In the Client tab, set **Client ID Policy** to **Unrestricted**, and click **Save**.
13. In the Load Balancing tab, de-select the **Server Affinity Enabled** option, and click **Save**.
14. Re-expand **Services**, and then select the **JMS Modules**.
15. Select the name of the JMS module you created in the right pane.
16. Click **New** to create another JMS resource, give it a name, and click **Create**.
17. Select the **Queue**, and then **New**, and then enter a name (such as *DistributedQueue-Callstate*), and click **Create**.
18. **JNDI Name**: Enter the name `wlss.callstate.backup.site.queue`.
19. Click **Save**, and then the shopping cart, and then **Commit Changes**.
20. Restart the server.

Configuring Cross Domain Security (Both Primary and Secondary Sites)

Oracle recommends, depending upon your requirements, that you enable cross domain security between your geographically redundant sites.

For information on cross domain security concepts and configuration details, refer to the following documents:

- Integration and Multi-Domain Best Practices in *Administering JMS Resources for Oracle WebLogic Server*
- Configuring Foreign Server Resources to Access Third-Party JMS Providers in *Administering JMS Resources for Oracle WebLogic Server*
- Simplified Access to Foreign JMS Providers in *Developing JMS Applications for Oracle WebLogic Server*
- Cross Domain Security in *Developing JTA Applications for Oracle WebLogic Server*
- Configuring Cross-Domain Security in *Administering Security for Oracle WebLogic Server*

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 Coherence cache, 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 engine in the Coherence cache 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 (referenced in the Foreign Server resource configuration specified for the primary site) 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. The example below 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 1-8 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 (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

To monitor replication across regional sites, administrators will have examine WebLogic behavior using a combination of WebLogic JMS and Coherence cache statistics.

Troubleshooting Replication

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.

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* 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 "Redeploying Applications in a Production Environment" in *Deploying Applications to Oracle WebLogic Server* 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 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.

Defining the Version in the Manifest

Both WAR and EAR deployments must specify a version identifier in the MANIFEST.MF file.

Example 1-9 Version Identifier in Manifest

```
Manifest-Version: 1.0
Created-By: 21.0.6 (Oracle Corporation)
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 Remote 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. The example 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 Remote Console or `weblogic.Deployer` utility to undeploy the correct deployment name.

Example 1-10 Sample WLST Session for Examining Session Count

```
connect()
custom()
cd
('examples:Location=myserver,Name=myserver_myserver_findme_findme,ServerRuntim
e=myserver,Type=SipApplicationRuntime')
ls()
-rw-   ActiveAppSessionCount                        0
-rw-   ActiveSipSessionCount                        0
-rw-   AppSessionCount                              0
```

```
-rw-   CachingDisabled                              true
-rw-   MBeanInfo
weblogic.management.tools.Info@5ae636
-rw-   Name
myserver_myserver_findme_findme
-rw-   ObjectName
examples:Location=myserver,Name=myserver_myserver_findme_findme,ServerRuntime=
myserver,Type=SipApplicationRuntime
-rw-   Parent
examples:Location=myserver,Name=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 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.

2

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.

> ⚠ **Warning**
>
> Configuring proxy registrar is deprecated in release 8.0 and later.

- Configuring the Proxy Registrar
- Configuring Diameter Client Nodes and Relay Agents

Configuring the Proxy Registrar

This chapter describes how to configure a proxy registrar and the permissible Proxy-Require options for the Sip Server in the Oracle Communications Converged Application Server deployment.

About Proxy Registrar Configuration

The Remote 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 Remote 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 *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. See sip-security.

  > **Note**
  >
  > You can also configure trusted hosts by using the Remote Console. See Using the Remote 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`.
- Security provider:
  You configure security providers by using the Remote 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.

  See the discussion of SIP servlet identity assertion in the *Converged Application Server Security Guide*.

Using the Remote Console to Configure Trusted Hosts

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

To add trusted hosts:

1. From the **Edit Tree** of the Remote Console, click **Custom Resources**, and then **sipserver**, and then **SIP Server**, and then **SIP Security**.
2. Enter any trusted hosts.
3. Click **Save**, click the shopping cart, and then click **Commit Changes**.

Configuring the Proxy Registrar

In this release, the Remote Console does not support configuring Proxy Registrar domains. All support for Proxy Registrars will be removed in the next release.

Configuring the Proxy-Required Options for the Sip Server Proxy

Provide the message header field values that the application supports in incoming messages. If the message header field of an incoming message contains a value from this configured list, then the message header is passed on to the application.

To provide the message header values for the Sip Server Proxy:

1. From the Remote Console, click **Custom Resources**, and then **sipserver**, and then **SIP Server**, and then **Proxy**.

2. In the **Proxy-Require Options** field, enter a comma-separated list of permissible values for the Proxy-Require headers. For example:
   `proxy,timer`
3. Click **Save**, click the shopping cart, and then click **Commit Changes**.

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 Getting Started With Oracle Internet Directory in *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. The list of commands varies depending on the components deployed to the server.

Table 2-1 Stand-alone 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.example.com privateId=alice`
			• `delete` – Deletes a communication identity from the system. For example:
			`publicIdentity delete publicId=sip:alice@test.example.com privateId=alice`

Table 2-1 (Cont.) Stand-alone Shell (Sash) Commands

Command	Description	Aliases	Subcommands

`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>`

`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`.

Table 2-1 (Cont.) Stand-alone Shell (Sash) Commands

Command	Description	Aliases	Subcommands

`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>]`

`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.

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.

Example 2-1 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.

> **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 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#`) by creating a Converged Application Server user known as *alice*.

1. Create a user using the `privateIdentity` command.

   ```
   privateIdentity add privateId=alice
   ```

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

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

3. Add an account for alice and use one of the optional commands 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.example.com password=<password>
   ```

   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 *Converged Application Server Security Guide*.

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

   > ⚠ **Warning**
   >
   > You can only create one user per Sash command. If you configure a single command that creates multiple users, only the final user will be created.

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

```
sash# privateIdentity add privateId=alice
sash# publicIdentity add publicId=sip:alice@test.example.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.example.com password=<password>
```

> ✓ **Tip**
>
> To create multiple users by creating Sash batch files, 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@example.com role="Location Service"
realm=example.com password=<password>
```

> **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@example.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.

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.

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.
- `--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.

Example 2-3 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
```

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`).

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 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.

The Converged Application Server supports multiple relays with a limitation that it tries to make connection to the first configured relay that is in the state I-OPEN. If first relay in the list is not in the I-OPEN state, then next node will be picked, and so on until it find a node in an I-OPEN state. Diameter load balancer is an implementation which will ensure that the load will be distributed sequentially to all the active relays using round robin algorithm.

> **Note**
>
> The Converged Application Server does not support the Dh interface.

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 2-2 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.

Table 2-2 (Cont.) Key Configuration Elements of the Diameter Domain

Server Name	Network Channel Configuration	Diameter Applications	Notes
`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: *Oracle_home***/wlserver/common/templates/wsl/diameterdomain.jar**

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. From the **Edit Tree** of the Remote Console, click **Environment**, and then **Servers**, and then **AdminServer**, and then **Channels**.
2. Click **New**, enter a name (such as Diameter TCP/TLS Channel), and click **Create**.
3. Under the **Channel General** subtab, fill in the fields as follows:
   - **Enabled**: Select this attribute to ensure that the new channel accepts network traffic.
   - **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 Address:** Enter the IP address or DNS name representing the external identity of this network channel.
   - **External Listen Port**: Re-enter the Listen Port value.
   - **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).

4. Optionally, click **Show Advanced Fields** and select values for the following fields:
   - **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.
   - **Idle Connection Timeout**: Change this 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.

5. Under the **Channel Security** subtab, set the following parameters:
   - **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.
6. Click **Save**, click the shopping cart, and click **Commit 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 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.

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>/config/custom`.

Where:

- *MW_home*—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*.
- *domain_name*—The name of the Diameter domain. In the following example, the domain name is "base_domain": `Oracle/Middleware/Oracle_Home/user_projects/domains/base_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. From the Remote Console, select **Custom Resources**, and then **diameter**, and then **Diameter**, and then **Configuration**.
2. Click **New** in the right pane to create a new Diameter configuration.
3. Fill in the fields of the Create a New Configuration page as described below, then click Create.

Table 2-3 Diameter Node General Configuration Properties

Property Name	Description
Configuration 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.

Table 2-3 (Cont.) Diameter Node General Configuration Properties

Property Name	Description
Address	List IP addresses or DNS names of the local interface(s) to be bound during connection setup. First address is the local primary address and others are alternate addresses. When the transport protocol is SCTP, all IP addresses will be associated with the remote SCTP endpoint. When the transport protocol is TCP or TLS, only the first address will be used. 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.
Port	Local port to be bound along with local interface(s) during connection setup. If the value is 0, the system assigns an ephemeral port.
Validate SCTP Peer Addresses	Enable this checkbox to validate the remote SCTP connection addresses of a Diameter Peer. If you enable this validation, only configured Peer Addresses are allowed in remote Peer Addresses offered during SCTP association setup. An SCTP association will be closed if any unknown remote Peer Address is present.
TLS Enabled	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.
Allow Dynamic Peers	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.
Maximum Request Attempts	Enter the maximum number of times to retry a request before giving up.
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.
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: • 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.
Load Balancing	Select this option if you want to enable load balancing. Load balancing is disabled by default.

4. Click **Create** to apply the configuration to target servers.

5. On the **Message Debug** tab, enable Message Debug if you want to enable tracing for Diameter messages processed by this node. Message tracing is disabled by default.
6. Click **Save**, and then the shopping cart, and then **Commit Changes**.

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 Remote Console using the page for a selected Diameter node. Follow these steps:

1. From the Remote Console, select **Custom Resources**, and then **diameter**, and then **Diameter**, and then **Configuration**.
2. Select the name of a Diameter node configuration in the right pane of the Console, and then click **Applications**.
3. Click **New** to configure a new Diameter application, or select an existing application configuration from the table.
4. Fill in the application properties as follows:
   - **Name**: Enter a name for the application configuration.
   - **Class Name**: Enter the classname of the application to deploy on this node.
   - **Params**: Enter optional parameters to pass to the application upon startup.
5. Click **Create** to create the new application configuration.
6. Click the shopping cart and then **Commit Changes**.

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.

The following example shows a sample node configuration for an Sh client node that uses a relay.

Example 2-4 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 to the client node that initiated the message, or 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.

The example below shows the sample **diameter.xml** configuration for a relay agent node.

Example 2-5 Diameter Relay Node Configuration

```
<?xml version='1.0' encoding='utf-8'?>
<diameter xmlns="http://www.bea.com/ns/wlcp/diameter/300"
          xmlns:sec="http://xmlns.oracle.com/weblogic/security"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xmlns:wls="http://xmlns.oracle.com/weblogic/security/wls">
  <configuration>
    <name>relay</name>
    <target>relay</target>
    <host>relay</host>
    <realm>oracle.com</realm>
    <!-- The local address and port are bound before initiating outbound 
connections
    to a remote Diameter server. Use 0.0.0.0 for all local addresses. Use 
port 0
    for ephemeral ports. -->
    <address>10.0.0.20,10.0.0.21</address>
    <port>0</port>
    <tls-enabled>false</tls-enabled>
    <debug-enabled>true</debug-enabled>
    <message-debug-enabled>true</message-debug-enabled>
    <message-debug>
      <message-debug-enabled>true</message-debug-enabled>
      <logging-enabled>true</logging-enabled>
      <file-min-size>500</file-min-size>
      <log-filename>diameter-messages.log</log-filename>
      <rotation-type>bySize</rotation-type>
      <number-of-files-limited>false</number-of-files-limited>
      <file-count>7</file-count>
      <rotate-log-on-startup>true</rotate-log-on-startup>
      <log-file-rotation-dir xsi:nil="true"></log-file-rotation-dir>
      <rotation-time>00:00</rotation-time>
      <file-time-span>24</file-time-span>
      <date-format-pattern>MMM d, yyyy h:mm:ss,SSS a z</date-format-pattern>
    </message-debug>
    <application>
      <name>RelayApplication</name>
      <class-name>com.bea.wcp.diameter.relay.RelayApplication</class-name>
    </application>
    <peer-retry-delay>30</peer-retry-delay>
    <allow-dynamic-peers>true</allow-dynamic-peers>
    <request-timeout>30000</request-timeout>
    <max-request-attempts>1</max-request-attempts>
    <watchdog-timeout>30</watchdog-timeout>
    <!-- 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>
      <protocol>tcp</protocol>
      <watchdog-enabled>false</watchdog-enabled>
    </peer>
    <peer>
      <host>hss</host>
      <address>localhost</address>
      <port>3870</port>
      <protocol>tcp</protocol>
      <watchdog-enabled>false</watchdog-enabled>
    </peer>
    <peer>
       <host>example.3gppnetwork.org</host>
        <!-- Remote Diameter server addresses that will listen for incoming
          connections from OCCAS Diameter client. Remote Diameter server
          port is same for all remote Diameter server addresses. -->
      <address>10.2.2.6, 10.2.3.6</address>
      <port>3869</port>
      <protocol>sctp</protocol>
      <watchdog-enabled>true</watchdog-enabled>
    </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. See "Working with Diameter Nodes" in *Converged Application Server Diameter Application Development Guide*.

Table 2-4 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**.
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**.

Example 2-6 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 Remote Console using the **Configuration** > **Peers** page for a selected Diameter node. Follow these steps:

1. From the Remote Console, select **Custom Resources**, and then **diameter**, and then **Diameter**, and then **Configuration**.
2. Select the name of a Diameter node configuration, and then **Peers**.
3. Click **New** to define a new peer entry.
4. Fill in the fields of the page as follows:
   - **Host**: Enter the peer node's host identity.
   - **Address**: Enter a comma-separated list of IP addresses or DNS names of the remote interface(s) for a Diameter peer. The first address is the primary remote address and others are alternate remote addresses. When the transport protocol is SCTP, all IP addresses will be associated with the remote SCTP endpoint. When the transport protocol is TCP or TLS, only the first address will be used.
     See "Validate SCTP Peer Address" for how the Converged Application Server behaves when a Diameter peer offers an IP address not in this list. If you do not specify an address, the host identity is used.
   - **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.
5. Click **Create** to create the new peer entry.
6. Click **Save**, and then the shopping cart, and then **Commit Changes**.

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 Remote Console using the **Configuration** and then **Routes** page for a selected Diameter node. Follow these steps:

1. From the Remote Console, select the **Diameter** node in the left pane of the Console.
2. Select the name of a Diameter node configuration in the right pane of the Console.
3. Select **Configuration**, then select the **Routes** tab.

4. Click **New** to configure a new Route.
5. 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.
   - **Load Balancing**: Select the checkbox to enable load balancing on this specific route.
6. Click **Finish** to create the new route entry.
7. Click **Activate Changes** to apply the configuration.

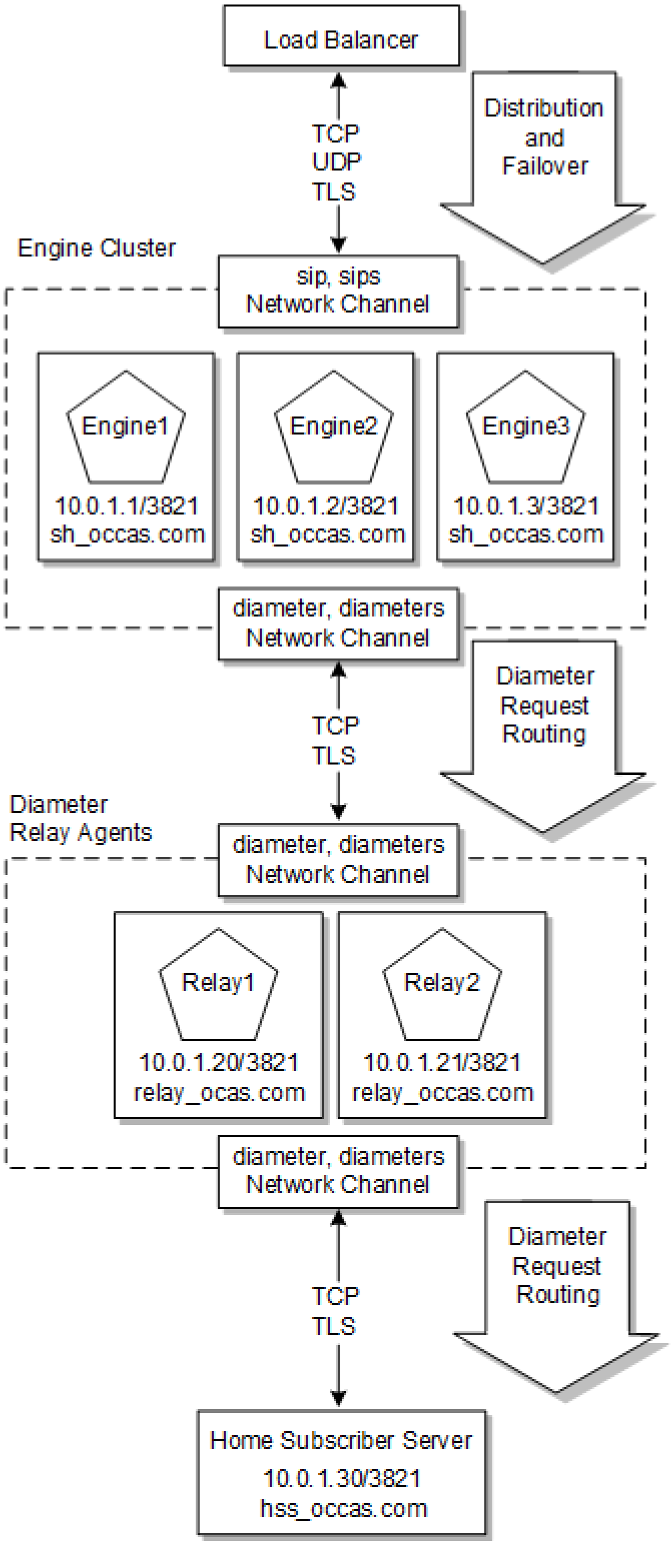
See Diameter Relay Node Configuration for an example **diameter.xml** node configuration containing a route entry.

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.
- Two servers that act as Diameter relay agents and forward diameter requests to an HSS.

Figure 2-1   Sample Diameter Domain



diameter.xml Configuration for Sample Engine Tier Cluster (Sh Clients)

The following example shows the contents of the **diameter.xml** file used to configure engine tier servers (Sh Clients) in the sample domain.

```
<?xml version='1.0' encoding='utf-8'?>
<diameter xmlns="http://www.bea.com/ns/wlcp/diameter/300" xmlns:sec="http://
xmlns.oracle.com/weblogic/security" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance" xmlns:wls="http://xmlns.oracle.com/weblogic/security/wls">
  <configuration>
    <name>clientnodes</name>
    <target>Engine1</target>
    <target>Engine2</target>
    <target>Engine3</target>
    <host>clientnodes</host>
    <realm>sh_occas.com</realm>
    <tls-enabled>false</tls-enabled>
    <debug-enabled>true</debug-enabled>
    <message-debug-enabled>true</message-debug-enabled>
    <message-debug>
      <message-debug-enabled>true</message-debug-enabled>
      <logging-enabled>true</logging-enabled>
      <file-min-size>500</file-min-size>
      <log-filename>diameter-messages.log</log-filename>
      <rotation-type>bySize</rotation-type>
      <number-of-files-limited>false</number-of-files-limited>
      <file-count>7</file-count>
      <rotate-log-on-startup>true</rotate-log-on-startup>
      <log-file-rotation-dir xsi:nil="true"></log-file-rotation-dir>
      <rotation-time>00:00</rotation-time>
      <file-time-span>24</file-time-span>
      <date-format-pattern>MMM d, yyyy h:mm:ss,SSS a z</date-format-pattern>
    </message-debug>
    <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-retry-delay>30</peer-retry-delay>
    <allow-dynamic-peers>true</allow-dynamic-peers>
    <request-timeout>30000</request-timeout>
    <max-request-attempts>1</max-request-attempts>
    <watchdog-timeout>30</watchdog-timeout>
    <peer>
      <host>Relay1</host>
      <address>10.0.1.20</address>
      <port>3821</port>
      <protocol>tcp</protocol>
      <watchdog-enabled>false</watchdog-enabled>
    </peer>
    <peer>
      <host>Relay2</host>
```

```
        <address>10.0.1.21</address>
        <port>3821</port>
        <protocol>tcp</protocol>
        <watchdog-enabled>false</watchdog-enabled>
      </peer>
      <default-route>
        <action>relay</action>
        <server>Relay1</server>
      </default-route>
  </configuration>
</diameter>
```

diameter.xml Configuration for Sample Relay Agents

The following example shows the **diameter.xml** file used to configure the relay agents.

```
<?xml version='1.0' encoding='utf-8'?>
<diameter xmlns="http://www.bea.com/ns/wlcp/diameter/300" xmlns:sec="http://
xmlns.oracle.com/weblogic/security" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance" xmlns:wls="http://xmlns.oracle.com/weblogic/security/wls">
  <configuration>
    <name>relaynodes</name>
    <target>Relay1</target>
    <target>Relay2</target>
    <host>relaynodes</host>
    <realm>relay_occas.com</realm>
    <tls-enabled>false</tls-enabled>
    <debug-enabled>true</debug-enabled>
    <message-debug-enabled>true</message-debug-enabled>
    <message-debug>
      <message-debug-enabled>true</message-debug-enabled>
      <logging-enabled>true</logging-enabled>
      <file-min-size>500</file-min-size>
      <log-filename>diameter-messages.log</log-filename>
      <rotation-type>bySize</rotation-type>
      <number-of-files-limited>false</number-of-files-limited>
      <file-count>7</file-count>
      <rotate-log-on-startup>true</rotate-log-on-startup>
      <log-file-rotation-dir xsi:nil="true"></log-file-rotation-dir>
      <rotation-time>00:00</rotation-time>
      <file-time-span>24</file-time-span>
      <date-format-pattern>MMM d, yyyy h:mm:ss,SSS a z</date-format-pattern>
    </message-debug>
    <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>
      <protocol>tcp</protocol>
      <watchdog-enabled>false</watchdog-enabled>
    </peer>
    <peer>
      <host>Engine2</host>
      <address>10.0.1.2</address>
      <port>3821</port>
      <protocol>tcp</protocol>
      <watchdog-enabled>false</watchdog-enabled>
    </peer>
    <peer>
      <host>Engine3</host>
      <address>10.0.1.3</address>
      <port>3821</port>
      <protocol>tcp</protocol>
      <watchdog-enabled>false</watchdog-enabled>
    </peer>
    <peer>
      <host>Relay1</host>
      <address>10.0.1.20</address>
      <port>3821</port>
      <protocol>tcp</protocol>
      <watchdog-enabled>false</watchdog-enabled>
    </peer>
    <peer>
      <host>Relay2</host>
      <address>10.0.1.21</address>
      <port>3821</port>
      <protocol>tcp</protocol>
      <watchdog-enabled>false</watchdog-enabled>
    </peer>
    <peer>
      <host>hss</host>
      <address>hssserver</address>
      <port>3870</port>
      <protocol>tcp</protocol>
      <watchdog-enabled>false</watchdog-enabled>
    </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).

3

Monitoring, Tuning, and Troubleshooting

This part provides information on monitoring, tuning, and troubleshooting Oracle Communications Converged Application Server.

This part contains the following chapters:

Monitoring, Tuning, and Troubleshooting Overview

This chapter provides a road map to the detailed monitoring, tuning, and troubleshooting chapters provided in this part.

Getting Started: Your System Stack

Before jumping into specific monitoring, tuning, and troubleshooting topics for Converged Application Server, you should step back and consider your entire system stack, from the lowest level components to the highest:

1. At the base of your system stack is the underlying hardware, or, in a virtualized environment, your virtual machine or hypervisor.
2. The next step up is your operating system.
3. Sitting on top of the operating system is your Java Virtual Machine (JVM).
4. Running within the JVM, is your Converged Application Server, based itself upon the WebLogic application server, and, within that, your Session Initiation Protocol (SIP) applications themselves.

While you may be most concerned about the performance of your SIP applications, you need to make sure that all levels of your system stack are optimized and running at peak performance. For instance, if your operating system is misconfigured, no amount of WebLogic tuning can improve the reliability of your SIP applications. Likewise, system hardware defects will stop everything dead in its tracks.

Once you know that the base of your system stack is stable, that you've got ample disk space and RAM, that your operating system is patched and not running any extraneous services, only then should you proceed up the stack with tuning recommendations. With a stable base, you can be certain that performance issues, should they occur, are localized to a particular top level software component.

The following sections will go through monitoring, tuning, and troubleshooting considerations for each level of the system stack, from lowest level to highest.

Hardware/VM Monitoring, Tuning, and Troubleshooting

Hardware or virtual machine monitoring and tuning is entirely dependent on your environment. Depending upon your requirements, some things to keep in mind include:

- General climate control monitoring for physical servers including temperature and humidity
- Temperature monitoring for CPU and power supplies
- Enclosure fan speed monitoring
- Hardware reliability elements such as error correcting RAM and RAID configurations as well as manageable network interface cards

Operating System and CPU Monitoring, Tuning, and Troubleshooting

Converged Application Server is certified to run on either Oracle Linux or Solaris operating systems, and there are many resources available covering monitoring, tuning and troubleshooting topics.

For Linux tuning and troubleshooting, see Monitoring the System and Optimizing Performance.

JVM Monitoring, Tuning and Troubleshooting

Since Converged Application Server itself runs within a JVM, you need to make sure that the JVM is correctly tuned and that you monitor it for any issues. For more information, see Monitoring, Tuning, and Troubleshooting the JVM. In that chapter, you'll find the following information:

- Profiling JVM Performance
- The Java Control+Break Handler
- Tuning JVM Garbage Collection for Production Deployments
- Avoiding JVM Delays Caused by Random Number Generation
- Troubleshooting Memory Leaks

Converged Application Server Monitoring, Tuning, and Troubleshooting

Next, you can attend to your Converged Application Server environment.

You can use Simple Networking Management Protocol (SNMP) to monitor your Converged Application Server environment. For information on enabling and using SNMP see Configuring Converged Application Server SNMP. In that chapter you'll find the following information:

- Configuring SNMP
- Understanding and Responding to SNMP Traps

Converged Application Server debugging and tuning topics are covered in Converged Application Server Debugging and Tuning, including the following topics:

- Recommended Debug Log Settings
- Server Performance Tuning Recommendations

Converged Application Server provides a monitoring console for SIP applications as well as flexible overload protection facilities which let you intercept and deal with SIP application performance issues before they threaten the stability of your environment. For more information, see Converged Application Server Monitoring and Overload Protection, which covers the following topics:

- SIP Server and Application Monitoring
- Other Ways to Monitor Converged Application Server
- Configuring Overload Protection

Converged Application Server offers flexible logging configuration which is helpful when debugging SIP application issues. See Logging SIP Requests and Responses, which covers the following topics:

- Defining Logging Servlets in sip.xml
- Configuring the Logging Level and Destination
- Specifying the Criteria for Logging Messages
- Specifying Content Types for Unencrypted Logging
- Enabling Log Rotation and Viewing Log Files
- trace-pattern.dtd Reference

For troubleshooting general WebLogic messages, see BEA-000001 to BEA-2163006 in *Oracle Fusion Middleware Error Messages*.

Monitoring the Sessions for License Limits

> **Note**
>
> The method of tracking license limits described in this section is deprecated and will be removed in the next release. See "Monitor Messages Per Second" for the method currently supported for all new deployments.

About the Monitoring of Licenses

As a system administrator, you can ensure that Converged Application Server is not exceeding the licensing limit for concurrent sessions per cluster in the system at any time.

In Converged Application Server, the number of concurrent sessions is the aggregate number of established virtual connections between two endpoints represented by subscriber devices or network switching equipment and traversing the licensed software at any one time.

A named user is an individual authorized by you to use the programs which are installed on a single server or multiple servers. This definition of a named user is valid regardless of whether the individual is actively using the programs at any given time. Additionally, Converged Application Server counts a non human operated device that can access the programs as a named user in addition to all individuals authorized to use the programs.

About the License Metrics

Converged Application Server supports the following when monitoring licenses:

- Cluster-based tracking

Total number of active sessions on a cluster

- Sip Sessions
- Diameter Sessions
- High water mark for sessions

Converged Application Server stores the high water mark values in the log file for each engine.

About the High Water Mark

A high water mark is the indicator which represents the highest value seen for a monitored entry in a specific period. Suppose that over the course of a week, the total count stored for a monitored entry goes from 1 to 10 to 5. The high water mark value for this entry is 10.

Converged Application Server provides the following metric for the high water mark metric:

- The high water mark for the latest specified interval or duration that was specified, such as the last hour or day.
- The high water mark since the start of the monitoring process.

To calculate the high water mark for the sessions in a cluster, Converged Application Server sums up all the active sessions across all engines in a cluster and saves the highest number of concurrent sessions. It performs this calculation for all engines synchronously.

About the Monitoring Process

When the monitoring of licenses is enabled, a dedicated polling thread is created whenever the Sip server of an engine starts (or restarts). This polling thread monitors the traffic from that starting point. At every interval, Converged Application Server stores the high water mark value for the previous interval and the high water mark value from the start of the monitoring. See Example of Log Entries for High Water Marks in a Sip Session.

If, for any reason, the server goes down in a standalone deployment, all the collected statistic is lost. In a cluster deployment, as long as one engine is alive, the high water mark value survives the event.

Setting Up the Logging Parameters

As an administrator, you specify the time when the monitoring is to begin and the length of the logging interval. Set up start time for the monitoring and the interval for each engine in the domain.

> **Note**
>
> When you have more than one engine in a cluster, these configurations must be identical for all the engines in that domain.

Configuring the License Tracking as Startup Command Options

To configure the settings as startup command options, specify the following for each engine in the domain.

- The local time when the monitoring is to start as:

  -Dwlss.sip.session.count.start_time=*start_time* where *start_time* is in the *HH:MM:SS* format, specifying the local time in hour, minute, and second with the 24-hour clock system. For example, the entry to start the logging for a server at 8:30 a.m. would be

```
-Dwlss.sip.session.count.start_time=08:30:00
```

By default, Converged Application Server commences its monitoring and logging process of an engine in a domain when the engine starts up.

- The time intervals for the log output:

  -Dwlss.sip.session.count.log_interval=*interval_seconds*

  Where, *interval_seconds* is the monitoring interval, in seconds. Converged Application Server commences its monitoring and logging process when the engine starts up.

  For example,

  ```
  -Dwlss.sip.session.count.log_interval=14400
  ```

  The entry 14400 is 4 hours. The high water mark entry is logged every 4 hours from the start of the logging, 08:30:00.

  If this interval is set to **0**, Converged Application Server does not monitor or save the log information.

About the Log Information

Converge Application Server stores all log entries for session high water mark of a cluster or standalone deployment in the server log file for each engine. The pathname to the *server_name***.log** file for each engine is:

*domain_name***/servers/***server_name***/logs/***server_name***.log**

where *domain_name* is the name of the directory in which you located the Converged Application Server domain and *server_name* is the name of the server.

Converged Application Server identifies the log information for the high water mark entries with the following entries:

- For Sip Sessions: `Concurrent SipSession Tracking:`
- For Diameter Sessions: `Concurrent Diameter Session Tracking:`

The high water mark entries are entered in the following way:

- The high water mark entry for the most recent interval:
  `high water mark of last interval:`*value*
- The high water mark entry since the start of the monitoring:
  `High water mark value of history:`*value*

The example below shows an excerpt from a high water mark log output for a Sip session, set to be logged every minute:

Example 3-1 Example of Log Entries for High Water Marks in a Sip Session

```
<Jan 11, 2016 12:06:00 PM PST> <Warning> <OCCAS> <BEA-000000> <Concurrent SipSession Tracking: |
AdminServer|High water mark of last interval:30|High water mark value of the history:30>
<Jan 11, 2016 12:07:00 PM PST> <Warning> <OCCAS> <BEA-000000> <Concurrent SipSession Tracking: |
AdminServer|High water mark of last interval:10|High water mark value of the history:30>
<Jan 11, 2016 12:08:00 PM PST> <Warning> <OCCAS> <BEA-000000> <Concurrent SipSession Tracking: |
AdminServer|High water mark of last interval:12|High water mark value of the history:30>
<Jan 11, 2016 12:09:00 PM PST> <Warning> <OCCAS> <BEA-000000> <Concurrent SipSession Tracking: |
AdminServer|High water mark of last interval:20|High water mark value of the history:30>
```

```
<Jan 11, 2016 12:10:00 PM PST> <Warning> <OCCAS> <BEA-000000> <Concurrent SipSession Tracking: |
AdminServer|High water mark of last interval:40|High water mark value of the history:40>
<Jan 11, 2016 12:11:00 PM PST> <Warning> <OCCAS> <BEA-000000> <Concurrent SipSession Tracking: |
AdminServer|High water mark of last interval:0|High water mark value of the history:40>
<Jan 11, 2016 12:12:00 PM PST> <Warning> <OCCAS> <BEA-000000> <Concurrent SipSession Tracking: |
AdminServer|High water mark of last interval:00|High water mark value of the history:40>
```

To find the high water mark value on any given day for a server, access the **logs** directory under the server name in your installation and enter the following grep command:

```
$ grep "Concurrent SipSession Tracking" engine1.log*
```

In this command, *engine1* is the name of the server.

Monitor Messages Per Second

The Converged Application Server allows you to monitor the messages per second (MPS) sent or received by your application to align with your licensed capacity. With a default 12-month historic MPS retention, you can audit usage trends and plan adjustments to your MPS license.

See the "MPS License Metric Definition" section of the License Document for a detailed definition of the MPS metric.

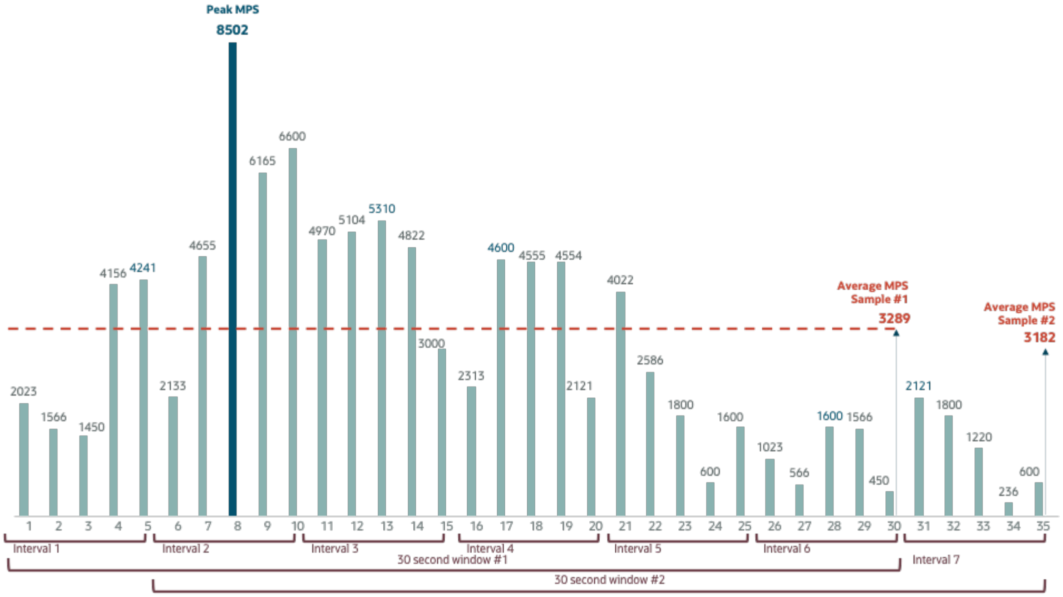
The following messages are not counted for the MPS metric:

- Messages between the Converged Application Server SIP Servlet applications in the same managed node (SIP Engine, JVM), regardless of:
  - Messages routed via the local network interface
  - Messages passed as java objects
- Messages sent to or received from observability and monitoring tools, continuous delivery tooling, configuration tools including REST API for configuration, and other tools used to monitor and operate the Converged Application Server environment
- Diameter Connection Control messages
- Messages rejected/dropped by the overload protection feature

Understand the following terms to understand how the licensed MPS is calculated.

- **MPS**—The number of messages per second received from or sent to external servers.
- **Peak MPS**—The highest Messages Per Second value during a 30-second window.
- **Sampling Rate**—Interval duration. By default, set to 5 seconds. The cumulative value of the number of messages for an interval duration is stored in an array. The array has six positions, hence creating a 30 second window.
- **Average MPS**—The average number of messages per second computed over a rolling 30 second window using the ceiling function (rounding to the next whole number).
  For example, given the sampling rate of 5 seconds, then the Converged Application Server calculates the average MPS over a 30 second sliding window. The first sample (of average MPS) covers 0 - 30 seconds, the second sample covers 5 - 35 seconds, the third sample covers 10 - 40 seconds, and so on.

  The following graph shows the **Peak MPS** and **Average MPS** for two consecutive 30 second windows. The **Peak MPS** is the same for both windows, namely 8502. The **Average MPS** however is different for each window: 3289 for the first window and 3182 for the second window.



- **Retention Window**—A 5 minute period during which the average MPS samples are retained in an internal buffer in memory. In total 60 average MPS values are stored in the internal buffer. From these 60 the highest value is stored for long term retention.
- **Peak Average MPS**—The highest number of average MPS values over the retention window.
  In the following example data set, the peak of the average MPS is 2542 within this 5 minute retention window. (All the other sample values are below 2542.)

Time	Sample	Average MPS
1 - 30 seconds	1	2534
5 - 35 seconds	2	2530
10 - 40 seconds	3	2515
15 - 45 seconds	4	2519
. . .	. . .	. . .
4 mins 25 sec - 4 mins 55 sec	59	2540
4 mins 30 sec - 5 mins	60	2542

  In this example, the next retention window starts at 5 mins 00 seconds. With the first averaged peak value written at 5 minutes 05 seconds using the before-mentioned array. Hence the values of the 5 previous intervals are used to determine the average peak for the 5 minutes 00 seconds to 5 mins 05 second interval and so on. The array is not emptied at the end or start of a retention window.
- **Licensed Peak Average MPS**—Your licensed MPS number.
  The Converged Application Server checks the retention window every 5 minutes for peak average MPS values that exceed your licensed peak MPS.

If the peak average MPS value exceeds the threshold limit, the Converged Application Server logs the alarm in the console log and updates the MPSConfig descriptor bean attribute **BreachInfo** with the message "Threshold limit crossed during period <FROMDATE> and <TODATE>".

ⓘ **Note**

After a restart or reboot of the Admin Server, the Average MPS count is inaccurate for the first 30 seconds and first retention window. The first Average MPS and the Peak Average MPS must be ignored. This also includes any MPS value threshold crossing alarms within the first retention window, which must be ignored as well.

The directory `<DOMAIN_HOME>/servers/logs/MPS` contains files called `MPS_<MM-dd-YYYY>.csv` that contains the start time, end time, and peak average MPS. This file is rotated daily. Files in this directory that are older than the **Historic MPS Persistency** value are purged.

ⓘ **Note**

Use the **Historic MPS Persistency** value to limit how much disk space is devoted to MPS logs.

Known Issue

The MPS monitoring function currently only records SIP and Diameter messages.

Configure MPS Monitoring

Enable MPS monitoring in the the Converged Application Server.

1. From the Remote Console, navigate to **Custom Resources**, and then **sipserver**, and then **SIP Server** and then **MPS Config**.
2. **Licensed Peak MPS**—Enter the number of peak average messages per second that you have been licensed for.

   A value of 0 disables the MPS limit.
3. **MPS Alert Threshold**—Enter the percentage threshold of **Licensed Peak MPS** which, once crossed, the Converged Application Server will generate an alert in the console log and update the BreachInfo attribute in the **MPSConfig** bean.

   The default value of 0 disables the MPS alert threshold. The range is 0 to 100.

   If your licensed peak MPS is 200 and your MPS alert threshold is 75, the Converged Application Server will generate an alarm if the peak average MPS is above 150 messages per second.
4. **Historic MPS Persistence**—Enter the number of months that the Converged Application Server should retain historic MPS values.

   The default value is 12. the Converged Application Server does not enforce an upper limit, so consider disk space limitations when setting this value.
5. **MPS Threshold Breach Info**—This field is updated when the average peak MPS crosses the licensed peak MPS.

   For example: "Threshold limit crossed during period Mon Jan 09 22:15:00 IST 2023 and Mon Jan 09 22:20:00 IST 2023"
6. Click **Save**, click the shopping cart, and click **Commit Changes**.

Monitoring, Tuning, and Troubleshooting the JVM

This chapter describes how to monitor and tune Java Virtual Machine (JVM) performance for Oracle Communications Converged Application Server engine servers.

Profiling JVM Performance

Java Flight Recorder and Java Mission Control together create a complete tool chain to continuously collect low level and detailed runtime information enabling after-the-fact incident analysis.

- **Java Flight Recorder** is a profiling and event collection framework built into the Oracle JDK that lets Converged Application Server administrators and developers to gather detailed low level information about how the Java Virtual Machine (JVM) and the Java application are behaving.
- **Java Mission Control** is an advanced set of tools that enables efficient and detailed analysis of the extensive of data collected by Java Flight Recorder. The tool chain enables developers and administrators to collect and analyze data from Java applications running locally or deployed in production environments.
- The **jcmd** utility is used to send diagnostic command requests to the JVM. It must be used on the same machine on which the JVM is running, and have the same effective user and group identifiers that were used to launch the JVM.

Using Java Flight Recorder

You can run multiple recordings concurrently and configure each recording using different settings; in particular, you can configure different recordings to capture different sets of events. However, in order to make the internal logic of Java Flight Recorder as streamlined as possible, the resulting recording always contains the union of all events for all recordings active at that time. This means that if more than one recording is running, you might end up with more information in the recording than you wanted. This can be confusing but has no other negative implications.

The easiest and most intuitive way to use JFR is through the Flight Recorder plug-in that is integrated into Java Mission Control. This plug-in enables access to JFR functionality through an intuitive GUI. For more information about using the JMC client to control JFR, see the Flight Recorder Plug-in section of the Java Mission Control help.

Using the Command Line

You can start and configure a recording from the command line using the -XX:StartFlightRecording option of the java command, when starting the application. To enable the use of JFR, specify the -XX:+FlightRecorder option. Because JFR is a commercial feature, you also have to specify the -XX:+UnlockCommercialFeatures option. The following example illustrates how to run the MyApp application and immediately start a 60-second recording which will be saved to a file named myrecording.jfr:

```
java -XX:+UnlockCommercialFeatures -XX:+FlightRecorder -
XX:StartFlightRecording=duration=60s,filename=myrecording.jfr MyApp
```

To configure JFR, you can use the -XX:FlightRecorderOptions option.

Using Diagnostic Command

You can also control recordings by using Java-specific diagnostic commands.

The simplest way to execute a diagnostic command is to use the `jcmd` tool (located in the Java installation directory). To issue a command, you have to pass the process identifier of the JVM (or the name of the main class) and the actual command as arguments to `jcmd`. For example, to start a 60-second recording on the running Java process with the identifier 5368 and save it to `myrecording.jfr` in the current directory, use the following:

```
jcmd 5368 JFR.start duration=60s filename=myrecording.jfr
```

To see a list of all running Java processes, run the `jcmd` command without any arguments. To see a complete list of commands available to a running Java application, specify `help` as the diagnostic command after the process identifier (or the name of the main class). The commands relevant to Java Flight Recorder are:

- `JFR.start`

  Start a recording.
- `JFR.check`

  Check the status of all recordings running for the specified process, including the recording identification number, file name, duration, and so on.
- `JFR.stop`

  Stop a recording with a specific identification number (by default, recording 1 is stopped).
- `JFR.dump`

  Dump the data collected so far by the recording with a specific identification number (by default, data from recording 1 is dumped).

Note

These commands are available only if the Java application was started with the Java Flight Recorder enabled, that is, using the following options:

```
-XX:+UnlockCommercialFeatures -XX:+FlightRecorder
```

Configuring Recordings

You can configure an explicit recording in a number of other ways. These techniques work the same regardless of how you start a recording (that is, either by using the command-line approach or by using diagnostic commands).

Setting Maximum Size and Age

You can configure an explicit recording to have a maximum size or age by using the following parameters:

- `maxsize=`*`size`*

  Append the letter `k` or `K` to indicate kilobytes, `m` or `M` to indicate megabytes, `g` or `G` to indicate gigabytes, or do not specify any suffix to set the size in bytes.

- `maxage=`*`age`*

  Append the letter `s` to indicate seconds, `m` to indicate minutes, `h` to indicate hours, or `d` to indicate days.

If both a size limit and an age are specified, the data is deleted when either limit is reached.

Setting the Delay

When scheduling a recording. you might want to add a delay before the recording is actually started; for example, when running from the command line, you might want the application to boot or reach a steady state before starting the recording. To achieve this, use the `delay` parameter:

```
delay=delay
```

Append the letter `s` to indicate seconds, `m` to indicate minutes, `h` to indicate hours, or `d` to indicate days.

Setting Compression

Although the recording file format is very compact, you can compress it further by adding it to a ZIP archive. To enable compression, use the following parameter:

```
compress=true
```

Note that CPU resources are required for the compression, which can negatively impact performance.

Creating Recordings Automatically

When running with a default recording you can configure Java Flight Recorder to automatically save the current in-memory recording data to a file whenever certain conditions occur. If a disk repository is used, the current information in the disk repository will also be included.

Creating a Recording On Exit

To save the recording data to the specified path every time the JVM exits, start your application with the following option:

```
-XX:FlightRecorderOptions=defaultrecording=true,dumponexit=true,dumponexitpath=path
```

Set *path* to the location where the recording should be saved. If you specify a directory, a file with the date and time as the name is created in that directory. If you specify a file name, that name is used. If you do not specify a path, the recording will be saved in the current directory.

Creating a Recording Using Triggers

You can use the Console in Java Mission Control to set *triggers*. A trigger is a rule that executes an action whenever a condition specified by the rule is true. For example, you can create a rule that triggers a flight recording to commence whenever the heap size exceeds 100 MB. Triggers in Java Mission Control can use any property exposed through a JMX MBean as the input to the rule. They can launch many other actions than just Flight Recorder dumps.

Define triggers on the **Triggers** tab of the JMX Console. For more information on how to create triggers, see the Java Mission Control help.

Troubleshooting

You can collect a significant amount of diagnostic information from Java Flight Recorder by starting the JVM with one of the following options:

- `-XX:FlightRecorderOptions=loglevel=debug`
- `-XX:FlightRecorderOptions=loglevel=trace`.

Java Flight Recorder Command Reference

For a listing of commands you can use with the Java Flight Recorder, see "Command Reference" in *Java Platform, Standard Edition Java Flight Recorder Runtime Guide*.

Using Java Mission Control

This section describes using Java Mission Control.

Starting the Java Mission Control Client

The JMC client executable file is located in the `bin` directory of the Java SE Development Kit (JDK) installation path (`JAVA_HOME`). If the `JAVA_HOME/bin` directory is in the `PATH` environment variable, you can start the JMC client by entering `jmc` at the command-line prompt (shell). Otherwise, you have to specify the full path to the JMC executable:

- `JAVA_HOME/bin/jmc` (Linux)

Passing JVM Options To the JMC Launcher

JMC is a Java application, and the JMC client executable is a launcher for this application. JMC startup is controlled by options specified in the `jmc.ini` file, which is located in the `JAVA_HOME/bin` directory. Arguments to the `-vmargs` option in the `jmc.ini` file are options that are passed to the JVM running the JMC application. You can specify these options to control the way this JVM runs. If you do not want to modify the `jmc.ini` file, you can specify JVM options on the command line as arguments to the `-vmargs` option of the `jmc` command.

> **Note**
>
> If other options are specified for the `jmc` command, the `-vmargs` option must be specified last.

To start the JMC client with your own set of JVM options (overriding those specified in the `jmc.ini` file), run the following command (separate multiple arguments with spaces):

```
jmc -vmarg arguments
```

To start the JMC client with additional JVM options (appending them to those specified in the `jmc.ini` file), run the following command (separate multiple arguments with spaces):

```
jmc --launcher.appendVmargs -vmarg arguments
```

Using a Workspace Directory

If you want to copy your settings for the JMC client to another computer or another user, or use different predefined settings for different applications, add the `-data` command-line option and define a *workspace directory* when you start the JMC client:

```
jmc -data workspace-directory
```

Using the Java Mission Control GUI

For detailed information on using the Java Mission Control GUI client, see "Java Mission Control Client GUI" in *Java Platform, Standard Edition Java Mission Control User's Guide*.

Creating Thread and Heap Dumps Using jcmd

You can use the Java utility, **jcmd**, to diagnostic command requests directly to the JVM. For detailed information on using the jcmd utility, see "The jcmd Utility" in *Java Platform, Standard Edition Troubleshooting Guide*.

Creating a Heap Dump using jcmd

To create a heap dump using jcmd execute the following command, replacing Process_ID with the process ID of your JVM process and specifying a path and filename for the output file:

```
jcmd Process_ID GC.heap_dump /path/filename
```

Example 3-2   Creating a Heap Dump

```
jcmd 5216 GC.heap_dump ~/heapdumps/myheapdump.dprof
  5216:
  Heap dump file created
```

Creating a Thread Dump using jcmd

To create a thread dump using jcmd execute the following command, replacing Process_ID with the process ID of your JVM process:

```
jcmd Process_ID Thread.print
```

Example 3-3   Creating a Thread Dump

```
jcmd 5216 Thread.print
5216:
  2014-09-19 13:12:30
  Full thread dump Java HotSpot(TM) 64-Bit Server VM (24.45-b08 mixed mode):

  "Thread-21" daemon prio=6 tid=0x0000000016109800 nid=0x1d5c in Object.wait()
  [0x000000001c22f000] java.lang.Thread.State: TIMED_WAITING (on object monitor)
  at java.lang.Object.wait(Native Method)
```

Other jcmd Commands

This section provides example jcmd commands.

List All JVM Processes

Run `jcmd` without any parameters (or with -l) to list all JVM processes preceded by a process ID:

```
jcmd -l
   6848
   8120 sun.tools.jcmd.JCmd -l
   3108 weblogic.Server
```

List jcmd Commands for a Particular Process

Run `jcmd` *PID* `help` to list the jcmd commands available for that process. Replace *PID* with the process ID of your JVM process:

```
jcmd 3108 help
  The following commands are available:
  VM.native_memory
  VM.commercial_features
  ManagementAgent.stop
  ManagementAgent.start_local
  ManagementAgent.start
  Thread.print
  GC.class_histogram
  GC.heap_dump
  GC.run_finalization
  GC.run
  VM.uptime
  VM.flags
  VM.system_properties
  VM.command_line
  VM.version
  help
```

Get More Information on a jcmd Command

To get more information on a jcmd command run `jcmd help` *command_name* where *command_name* is the name of the jcmd command:

```
jcmd help GC.heap_dump
  GC.heap_dump
  Generate a HPROF format dump of the Java heap.

  Impact: High: Depends on Java heap size and content. Request a full GC
  unless the '-all' option is specified.

  Syntax : GC.heap_dump [options] <filename>

  Arguments:
      filename :  Name of the dump file (STRING, no default value)

  Options: (options must be specified using the <key> or <key>=<value> syntax)
  -all : [optional] Dump all objects, including unreachable objects
  (BOOLEAN, false)
```

jcmd Command Reference

For a listing of commands you can use with the jcmd utility, see "Command Reference" in *Java Platform, Standard Edition Java Flight Recorder Runtime Guide*.

The Java Control+Break Handler

On Oracle Solaris or Linux operating systems, the combination of pressing the Control key and the backslash (\) key at the application console (standard input) causes the Java HotSpot VM to print a thread dump to the application's standard output. On Windows, the equivalent key sequence is the Control and Break keys. The general term for these key combinations is the **Control+Break** handler.

On Oracle Solaris and Linux operating systems, a thread dump is printed if the Java process receives a QUIT signal. Therefore, the kill -QUIT pid command causes the process with the ID pid to print a thread dump to standard output.

Tuning JVM Garbage Collection for Production Deployments

This Section describes how to tune Java Virtual Machine (JVM) garbage collection performance for Oracle Communications Converged Application Server engine servers.

Goals for Tuning Garbage Collection Performance

Production installations of Converged Application Server generally require extremely small response times (under 50 milliseconds) for clients 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.

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 Converged Application Server installation, all long GC intervals must be avoided to maintain response time goals.

The sections that follow describe GC tuning strategies for Oracle's JVM that generally result in best response time performance.

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 *Middleware_Home***/user_projects/domains/***domain_name***/bin** directory, where *Middleware_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. The also contain a variety of Java configuration settings.
- **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.

Tuning Garbage Collection with Oracle JDK

When using Oracle'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 collector and the parallel young generation collector, restricting the new generation size to at most one third of the overall heap.

Oracle recommends using the Garbage-First (G1) garbage collector. See "Getting Started with the G1 Garbage Collector" for more information on using the Garbage-First collector.

The following example JVM settings are recommended for most production engine servers:

```
-server -Xms24G -Xmx24G -XX:+UseG1GC -XX:MaxGCPauseMillis=200 -XX:ParallelGCThreads=20 -
XX:ConcGCThreads=5 -XX:InitiatingHeapOccupancyPercent=70
```

For standalone installations, use the example settings:

```
-server -Xms32G -Xmx32G -XX:+UseG1GC -XX:MaxGCPauseMillis=200 -XX:ParallelGCThreads=20 -
XX:ConcGCThreads=5 -XX:InitiatingHeapOccupancyPercent=70
```

The above options have the following effect:

- **-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:+UseG1GC**: Use the Garbage First (G1) Collector.
- **-XX:MaxGCPauseMillis**: Sets a target for the maximum GC pause time. This is a soft goal, and the JVM will make its best effort to achieve it.
- **-XX:ParallelGCThreads**: Sets the number of threads used during parallel phases of the garbage collectors. The default value varies with the platform on which the JVM is running.
- **-XX:ConcGCThreads**: Number of threads concurrent garbage collectors will use. The default value varies with the platform on which the JVM is running.
- **-XX:InitiatingHeapOccupancyPercent**: Percentage of the (entire) heap occupancy to start a concurrent GC cycle. GCs that trigger a concurrent GC cycle based on the occupancy of the entire heap and not just one of the generations, including G1, use this option. A value of 0 denotes 'do constant GC cycles'. The default value is 45.

Avoiding JVM Delays Caused by Random Number Generation

The library used for random number generation in Oracle'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 system before returning a result.

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 need not continue. If the command does not return immediately, configure the `rngd` daemon to feed data to the kernel's random number entropy pool:

```
rngd -r /dev/urandom -o /dev/random -f -t .1
```

> **Note**
>
> You may have to experiment with the value of the **-t** parameter. For more information on the **rngd** daemon, run the **man rngd** command from a shell to display this manual page.

Troubleshooting Memory Leaks

If your application's execution time becomes longer and longer, or if the operating system seems to be performing slower and slower, this could be an indication of a memory leak. In other words, virtual memory is being allocated but is not being returned when it is no longer needed. Eventually the application or the system runs out of memory, and the application terminates abnormally.

For more information on diagnosing Java memory leaks, see "Troubleshooting Memory Leaks" in *Java Platform, Standard Edition Troubleshooting Guide*.

In addition, you can also use the Eclipse Memory Analyzer Tool (MAT) during development to discover memory leaks as well as reduce memory consumption.

For details on the Eclipse MAT, see `http://www.eclipse.org/mat/`.

Configuring Converged Application Server SNMP

This chapter describes how to configure and manage SNMP services with Oracle Communications Converged Application Server.

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

In this release, the Remote Console does not support configuring SNMP.

To learn when the Remote Console supports configuring SNMP, monitor the Releases page.

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.

Trap Descriptions

This section describes the Converged Application Server SNMP traps.

overloadControlActivated, overloadControlDeactivated

Converged Application Server engines 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.

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 Remote Console. If you specify a session rate control, you cannot monitor the session rate using the Remote Console. (The Remote Console only displays the current number of SIP sessions, not the rate of new sessions generated.)

serverStopped

This trap indicates that the WebLogic Server instance is now down. 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 Remote 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 engines 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 engines 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.

Converged Application Server Debugging and Tuning

This chapter describes how to debug and tune Oracle Communications Converged Application Server.

Debugging Issues in the Runtime Environment

At times, issues can arise in your runtime environment such as when a call session fails or a server fails because the server configuration did not load correctly. You can resolve the issues in the runtime environment with the help of the debug attributes that Converged Application Server supports.

About the Runtime Debug Process

When you encounter an issue in the runtime environment, review the debug attributes that Converged Application Server supports in the Remote Console. To further diagnose an issue, select one or more of the relevant debug attributes that would help to reproduce the scenario.

You can isolate the debug process by enabling the selected debug attributes in one server only. Or you can attempt to view the behavior by enabling the selected attributes in all servers.

After enabling the relevant debug attributes in the Remote Console, rerun the scenario.

By default, Converged Application Server prints the debug log information to standard output stream, **stdout**. To parse the debug information, pipe the **stdout** data to a file. When the issue is resolved, be sure to disable the debug flag settings in the Remote Console. You can redirect the Java virtual machine (JVM) output to a log file. For more information, see the description about "Redirecting JVM Output" in *Oracle Fusion Middleware Configuring Log Files and Filtering Log Messages for Oracle WebLogic Server.*

If you pipe **stdout** content to a file, manage the rotated log files in your production or development environment. For more information about rotating log files, see "Configuring Log File Rotation" in *Converged Application Server Developer's Guide.*

> ⚠ **Warning**
>
> Debug flags can result in gigabytes of log output in heavy traffic cases. Make sure that you use targeted debug flags as described in Debug Attributes.
> Revert the setting for the debug attributes as soon as the issue you are tracing is reproduced.

About the Debug Attributes Configuration Method

Only use the Remote Console to enable or disable debug attributes in Converged Application Server.

Converged Application Server provides the **serverdebug.xml** file for your reference. This file is located in the *domain_home***/config/custom** directory, where *domain_home* represents the directory in which Converged Application Server domain is created.

> ⚠ **Caution**
>
> - Do not modify the **serverdebug.xml** file manually.
> - Do not enter a debug setting as a startup command option.

Recommended Debug Log Settings

The table below lists the various scenarios that you may encounter and the recommended debug attributes to select and enable in the Remote Console for information on debugging each issue.

Table 3-1 Converged Application Server Debug Attributes

Debug Attribute	Issues for which this Attribute Provides Information
wlss.Admin	Details about Sip Server startup and the loading of its modules
wlss.AppRouter	Application router issues
wlss.CallState	Call state issues
wlss.coherenceStore	Information related to Oracle Coherence
wlss.concurrent	Concurrent service issues
wlss.Deployment	Application deployment issues
wlss.Diameter	Diameter protocol handling within the container issues
wlss.Dns	Issues related to Domain Naming Service
wlss.Filters	Filter module that filters SIP message issues
wlss.Geo	Geo redundancy handling issues
wlss.Headers	Sip (message) header issues
wlss.History	Issues related to call state history
wlss.instrumentation	Issues related to diagnostics, SIP or Diameter message properties
wlss.MHDebug	Message handler issues
wlss.MPS	Messages per second
wlss.RuntimeRest	Issue related to runtime MBean
wlss.Security	Security-related issues
wlss.SipEngine	Issues related to SIP servers
wlss.SipEngineConfig	Issues related to the loading of a server configuration
wlss.SipRequest	Issues related to SIP requests
wlss.SipSession	Issues related to SIP sessions

Table 3-1 (Cont.) Converged Application Server Debug Attributes

Debug Attribute	Issues for which this Attribute Provides Information
wlss.Status	Performance and garbage collection issues
wlss.Status.Timer	Timer statistics related to collections performance
wlss.Store	Call state and other cache issues
wlss.Timer	Internal timer issues
wlss.Traffic	Issues related to Sip traffic information
wlss.Transaction	Issues related to client and server transactions created within the container to process the call flow
wlss.Transport	Transport-level information for UDP, TCP, or TLS protocols.
wlss.Wrapping	Issues related to Enterprise JavaBeans (EJB) business object wrapping

Issues that Require the Enabling of Multiple Debug Attributes

Several issues require the enabling of multiple flags. The most common issues are described here.

SIP Specific Issues Involving Calls

In general for any SIP specific issue involving calls, enable the following debug flags on the engine servers and retrieve the server and **stdout** logs:

- wlss.Transaction
- wlss.SipSession

Transport-level Issues

In general for transport-level issues, enable the following debug flags on the engine servers and retrieve the server and **stdout** logs:

- wlss.Transport
- wlss.SipSession

Server Does not Process SIP Messages

For SIP specific issues, a Wireshark trace indicates that a SIP message reached the server but the message was not processed. Enable the following debug flags on the engine servers and retrieve the server and **stdout** logs:

- wlss.Admin
- wlss.MHDebug
- wlss.Transaction
- wlss.SipSession

Locking and Timer-related Issues

For locking and timer-related issues, enable following debug flags in combination or one at a time:

- wlss.CallState
- wlss.SipEngine
- wlss.Timer

Message Validation Issues

For message handling details, enable following debug flags in combination or one at a time:

- wlss.SipRequest
- wlss.Headers

Enabling the Runtime Debug Attributes

To debug and resolve issues at run time, you can enable the appropriate debug attributes for one or all servers through the Remote Console.

To set the debug attributes through the Remote Console:

1. From the Edit Tree of the Remote Console, select **Custom Resources**, and then **serverdebug**, and then **SIP Server Debug**, and then **Servers**, and then a specific server (such as the **AdminServer**).
2. Select the check box next to the debug attribute that you want to enable or disable. For information about the attributes, see "Recommended Debug Log Settings".
   In the Remote Console, the initial string "wlss." is removed from the presentation of the attribute.

For more information, see the description about "Define debug settings" in *Oracle Fusion Middleware Oracle WebLogic Remote Console Online Help.*

Server Performance Tuning Recommendations

The following recommendations can help improve Converged Application Server performance.

- Disable the **Domain log broadcaster** at each engine server Logging page.

  Using JConsole, for each of your Converged Application Server managed servers, select **Logging**, then **Advanced**, and then **Domain log broadcaster**, and set **Severity level** to **Off**.
- Increase the **Shared Capacity For Work Managers** to 5000000 for each engine.

  Using JConsole, select **Environment**, then **Servers**, then **Configuration**, and then *OCCAS_engine* **Overload**, where *OCCAS_engine* is the server name of one of your Converged Application Server engines.
- Increase the following tuning values in the Work Manager Settings:
  - set **wlss.transport.capacity** to **5000000**
  - set **wlss.timer.capacity** to **150000**
  - set **wlss.timer.maxthreads** to **200**

  In JConsole, find the Work Managers by selecting your Converged Application Server environment, then selecting each Work Manager in turn.
- In **config.xml** for the Converged Application Server Admin Server, increase the engines server socket-readers from 2 to 10.

Find config.xml in *domain_home*/**config**, where *domain_home* is the root directory of the domain.

- Manage SIP Application Session Timeout
- Specifying the Minimum and Maximum Thread Pool Size

Manage SIP Application Session Timeout

To prevent a caller or callee from being logged out during a session, make sure that the SIP application session timeout is conservatively set based upon the upper bound of a call duration. To set the session timeout, add the following entry to the **sip.xml**, where *n* determines the timeout in minutes:

```
<session-config>
  <javaee:session-timeout>n</javaee:session-timeout>
</session-config>
```

The session timeout is set to 3 minutes by default. As a rule of thumb, if the maximum call duration is, for example, 3 minutes, a rational setting for the session timeout would be 5 to 7 minutes.

Max Application Session Timeout

While the SIP application session timeout is dictated by the **session-timeout** element in **sip.xml** as described above, the maximum application session lifetime is configured at the container level by the **max-application-session-lifetime** element in **sipserver.xml**. The **max-application-session-lifetime** element essentially limits the maximum value a deployer can set for the **session-timeout** element in **sip.xml**. Configuring this element helps resource management by ensuring that a deployer is forced to set a timeout value within specific boundaries. If a value is not specified for **max-application-session-lifetime**, a deployer can set any value in **sip.xml**.

Specifying the Minimum and Maximum Thread Pool Size

Depending upon the number of concurrent users for your environment, you should adjust the value of the **SelfTuningThreadPoolMinSize** and **SelfTuningThreadPoolMaxSize** server parameter.

The self-tuning thread pools start with a default size, which grows and shrinks automatically as required. The default size for the administration server is 15, and 100 for engines in each cluster. You can increase the number of threads to improve throughput, but the minimum will never fall below the default. However, an excessive number of threads increases memory use, and could cause garbage collection related performance issues. Oracle recommends 200 as a working minimum and 400 as a working maximum. If you have a low number of concurrent users, you can use a lower value.

To configure **SelfTuningThreadPoolMinSize** and **SelfTuningThreadPoolMaxSize**:

1. From the Remote Console, select **Environment** and then select **Servers**.
2. Select the server name from the Servers table.
3. Click on the **Advanced** tab, and then the **Tuning** subtab.
4. Click the **Show Advanced Fields** checkbox.
5. Enter a value for **Self Tuning Thread Minimum Pool Size**.
6. Enter a value for **Self Tuning Thread Maximum Pool Size**.

7. Click **Save**, click the shopping cart, and then click **Commit Changes**.
8. Restart the server.

Files for Troubleshooting

The following Converged Application Server log and configuration files are frequently helpful for troubleshooting problems. Your technical support contact generally requests the following files from you:

- *domain_home***/config/custom/coherence.xml**
- *domain_home***/config/coherence/Coherence-Default/Coherence-Default.xml**
- *domain_home***/config/coherence/Coherence-Default/Custom-Default.xml**
- *domain_home***/config/config.xml**
- *domain_home***/config/custom/sipserver.xml**
- *domain_home***/***server_name***/*.log** (Server and message logs)
- Located in the **/WEB-INF** subdirectory of the application
- Located in the **/WEB-INF** subdirectory of the application

By default, *domain_home* represents the directory in which Converged Application Server domain is created and *server_name* is the name of the server.

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

Backwards Compatibility with TO and FROM System Headers

In JSR289/RFC3261, you could modify TO and FROM fields. But JSR116/RFC2543 changed TO and FROM parameters into system headers that can't be modified.

For backwards compatibility, use the Boolean flag **wlss.enable_modify_to_from** to modify the TO and FROM headers in a request in a proxy servlet.

The fields can't be modified directly from the request object. Instead, the address object for the TO and FROM fields must be retrieved and the getFrom() and getTo() methods of SIPServletRequest. For example:

```
sipServletRequest.getFrom().setDisplayName("new user");
sipServletRequest.getFrom().setExpires(1000);
sipServletRequest.getFrom().setQ(0.1f);
URI myUri = sipServletRequest.getFrom().getURI():
sipServletRequest.getFrom().getURI().setParameter("newparam", "newvalue");
SipURI mySipUri = (SipURI) sipServletRequest.getFrom().getURI();
mySipUri.setUser("newuser");
```

By default, **wlss.enable_modify_to_from** is disabled (set to false).

Converged Application Server Monitoring and Overload Protection

This chapter describes Oracle Communications Converged Application Server monitoring as well as overload protection and how it is configured.

About Monitoring and Overload Protection

Converged Application Server provides two interrelated systems that you can use together to ensure your environments remain within functional boundaries:

- SIP Server and Application Monitoring Console
- SIP Overload Protection

The first system, SIP Server and Application Monitoring Console, provides you with a window into the performance of your SIP servers and deployed SIP applications. Using the console, you can review the real time performance of your servers and applications, and spot possible bottlenecks and impending failure conditions.

The second system, SIP Overload Protection, enables you to act upon the data you see in the SIP Server and Application Monitoring Console. Using the SIP Overload Protection interface, you can set flexible traps and thresholds, and statistical algorithms to gracefully handle many types of performance issues before they endanger the health of your environment.

SIP Server and Application Monitoring

Converged Application Server provides a console interface for monitoring your Session Initiation Protocol (SIP) servers and SIP applications.

To access the monitoring interface, do the following:

1. From the Remote Console, select the **Monitoring Tree**.
2. In the Monitoring Tree, you can select the following nodes:
   - **Sip Server Runtime**: Provides general monitoring data on configured SIP servers and provides server performance information.
   - **Sip Application Runtime**: Provides performance information on deployed SIP applications.

Call State Storage is not supported in this release. Instead, you can use the Coherence CLI to gather information about SIP call state.

The following sections provide details on each monitoring subtab.

SIP Server Runtime

The SIP Server Runtime node of the Monitoring Tree provides runtime server and performance statistics over a period of time for each configured SIP server. The period (default 60 seconds) and sample frequency (default 10 seconds) are noted at the bottom of the pane.

Table 3-2 SIP Performance Monitoring Data

Datum	Description
Server	The name of the SIP server instance.
Start Time	The time at which the SIP server instance was started.
Application Session Count	The number of active SIP application sessions.
SIP Session Count	The number of active SIP sessions.
Destroyed Application Session Count	The number of destroyed application sessions.
Destroyed SIP Session Count	The number of destroyed SIP sessions.
Messages Received	The number of SIP messages received.
Messages Rejected	The number of rejected SIP messages.
Messages Processed	The total number of SIP messages processed.
Cluster Id	The Converged Application Server cluster ID.
SIP Throughput	The SIP message throughput.
Succeeded SIP Trans	The number successful SIP transactions.
Failed SIP Trans	The number of failed SIP transactions.

SIP Application Runtime

The SIP Application Runtime node of the Monitoring Tree provides runtime session information for SIP applications deployed on each configured SIP server.

Table 3-3 SIP Applications Data

Datum	Description
Engine	The Converged Application Server engine on which the SIP application is deployed.
Name	The name of the SIP application.
SIP Session Count	The number of active SIP sessions.
Application Session Count	The number of active application sessions.
Destroyed SIP Session Count	The number of destroyed SIP sessions.
Destroyed Application Session Count	The number of destroyed application sessions.

Other Ways to Monitor Converged Application Server

In addition to using the monitoring functionality in the WebLogic console, you can also monitor Converged Application Server using the WebLogic Scripting Tool (WLST), Java Management Extensions (JMX) as well as the WebLogic Diagnostic Framework (WLDF). The next sections provide additional details.

Monitoring Applications with the WebLogic Scripting Tool

The WebLogic Scripting Tool (WLST) is a command-line scripting environment that you can use to create, manage, and monitor WebLogic domains. It is based on the Java scripting interpreter, Jython. In addition to supporting standard Jython features such as local variables, conditional variables, and flow control statements, WLST provides a set of scripting functions (commands) that are specific to WebLogic Server.

You can use WLST to retrieve information that WebLogic Server instances produce to describe their run-time state. For more information, see "Getting Runtime Information" in *Understanding the WebLogic Scripting Tool*.

Developing Custom Management Utilities with JMX

To integrate third-party management systems with the WebLogic Server management system, WebLogic Server provides standards-based interfaces that are fully compliant with the Java Management Extensions (JMX) specification. You can use these interfaces to monitor WebLogic Server MBeans, to change the configuration of a WebLogic Server domain, and to monitor the distribution (activation) of those changes to all server instances in the domain.

To get started creating custom JMX management utilities, see *Developing Custom Management Utilities Using JMX for Oracle WebLogic Server*.

WebLogic Server Diagnostic Framework

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 engines, as well as deployed SIP Servlets. For details, see Using the WebLogic Server Diagnostic Framework (WLDF).

About Converged Application Server Overload Protection

Converged Application Server implements an overload framework which supports plug-in statistics collectors, plug-in event handlers, as well as multiple threshold settings and statistics collection algorithms.

About the Overload Protection Framework

Converged Application Server overload protection statistics collectors and event handlers are installed as Statistics Provider Interface (SPI) plug-ins. Only a single instance of each statistics collector and event handler can be instantiated as utility functions in the SPI.

Multiple thresholds can be configured for each statistics collector, and, when activated upon an incoming SIP session, samples are collected at a user-configurable interval, and statistics results are calculated according to a user-configurable algorithm. The results of the statistics calculations are then used to execute particular actions depending upon the comparison of those results with a user-configurable threshold value.

Configuring Overload Protection

Execute the following steps in order, since the later configurations have dependencies upon the earlier steps.

Using the Remote Console, you:

1. Configure a new event handler. See "About Event Handlers".
2. Configure actions for the event handler. See "About Actions".

3. Configure a statistics collector. See "About Statistics Collectors".
4. Configure a threshold, which includes a threshold statistics value, as well as sampling intervals, number of samples to collect at each interval (or real-time sampling), an algorithm to calculate the collected samples, as well as actions for upward and downward breaches of the threshold. See "About Thresholds".

About Event Handlers

A Converged Application Server overload protection *event handler* plugs in to the SPI, and is discovered when the overload protection framework is initialized. When a particular event handler is discovered, only one instance is created and managed by the framework. Each event handler must implement one or more *actions*. When a threshold-breaching event occurs, the framework executes the actions defined for the event handler.

Each event handler can accept an optional event-handler scoped set of user configurable key/value pairs, which are passed to the event handler's `activate()` method as parameters.

Configuring an Event Handler

To configure an overload protection event handler:

1. From the Remote Console, click **Custom Resources**, and then **sipserver**, and then **SIP Server**, and then **Overload Protections**, and then **Event Handlers**.
2. In the Event Handlers table, click **New**.
   Enter the following information:
   - **Event Handler Name**: *Required*. Enter a name for the event handler, for example: `com.oracle.sendSnmpTrap`

     Table 3-4 Default Event Handlers

Event Handler	Description
com.bea.wcp.sip.engine.server.olp.handler.ControlTrafficHandler	Used for a new call setup on a SIP container and either reject or accept call traffic.
com.bea.wcp.sip.engine.server.olp.handler.SendSNMPTrapHandler	Used to send SNMP traps.

   - **Attributes**: *Optional*. Specify key/value attribute pairs separated by semicolons, for example: c
     Attributes are passed to the event handler as parameters when the event is triggered.

   > **Note**
   >
   > The **com.bea.wcp.sip.engine.server.olp.handler.SendSNMPTrapHandler** event handler supports a **snmp-trap-message** attribute. Its default value is *overloadControlActivated*. No attributes are supported for the **com.bea.wcp.sip.engine.server.olp.handler.ControlTrafficHandler** event.

3. Click **Create** to save your configuration changes.
4. Click the shopping cart, and then click **Commit Changes**.

About Actions

Once you have defined an event handler, you must define one or more actions for the event handler to take when a threshold breaching event occurs. As with event handlers, actions are also plugged into the overload protection framework using the SPI, and are discovered when the framework is initialized, and, when discovered, only one instance is created and managed by the framework.

Each action can accept an optional action-scoped set of user configurable key/value pairs, which are passed to the actions `activate()` method as parameters.

Supported out of the box action types are listed in Default Action Types.

Configuring an Action

To configure an overload protection action:

1. From the Remote Console, click **Custom Resources**, and then **sipserver**, and then **SIP Server**, and then **Overload Protections** and then **Actions**.
2. In the Actions table, click **New**.
   Enter the following information:
   - **Action Name**: *Required*. Enter a name for the action, for example:

     ```
     TrafficReject
     ```

   - **Event Handler**: *Required*. Choose the name of an event handler you have created. For information on configuring an event handler, see "About Event Handlers".
     If you provide an invalid event handler name, the Remote Console will not throw an exception. However, the engine will fail to start up.
   - **Action Type**: *Required*. Enter an Action Type supported by the Event Handler, for example:

     ```
     reject-traffic
     ```

     Table 3-5 Default Action Types

Action Type	Description
accept-traffic	Used by the event handler, **com.bea.wcp.sip.engine.server.olp.handler.ControlTrafficHandler** . After an overload condition has cleared, accepts SIP session traffic.
reject-traffic	Used by the event handler, **com.bea.wcp.sip.engine.server.olp.handler.ControlTrafficHandler** . When an overload condition occurs, rejects SIP session traffic. SIP session traffic will continue to be rejected until an accept-traffic action is triggered.
default	Used by the event handler, **com.bea.wcp.sip.engine.server.olp.handler.SendSNMPTrapHandler**.

   - **Attributes**: *Optional*. Specify key/value attribute pairs separated by semicolons, for example:

     ```
     attribute1=21;attribute2=64
     ```

     Attributes are passed when the action is triggered.

> **Note**
>
> Support for attributes is dependent upon the implementation of the particular action. None of the default Action Types support any attributes.

3. Click **Create** to save your configuration changes.
4. Click the shopping cart, and then click **Commit Changes**.

About Statistics Collectors

Statistics collectors are also plugged into the overload protection framework using the SPI, and are discovered when the framework is initialized. When a particular statistics collector framework is discovered, only one instance is created and managed by the framework.

Each statistics collector consists of a *name*, a *type* and optional *attributes*. The collector name is referred to when defining a *threshold* as described in "Configuring a Threshold". The overload protection framework retrieves statistics samples using the statistics collector's `getStats()` method to which the optional attributes are passed as parameters.

Supported out of the box statistics collectors are described in Default Statistics Collector Types.

Configuring a Statistics Collector

To configure an overload protection statistics collector:

1. From the Remote Console, click **Custom Resources**, and then **sipserver**, and then **SIP Server**, and then **Overload Protections** and then **Statistics Collector**.
2. In the Statistics Collector table, click **New**.
   Enter the following information:
   - **Statistics Collector Name**: *Required*. Enter a name for the action, for example: `MBeanStatsCollector`
   - **Statistics Collector Type**: *Required*. Enter an Action Type supported by the Event Handler, for example: `mbean-stats`
     The following table lists the Statistics Collector Types supplied with Converged Application Server.

     Table 3-6 Default Statistics Collector Types

Statistics Collector Type	Description
queue-length	Uses the sum of the length of the transport and timer work manager queue lengths.
mbean-stats	Uses an MBean counter as a statistics example.
memory-usage	Returns the call state memory usage from Coherence.
active-diameter-session	Returns the number of active Diameter sessions.

   - **Attributes**: *Optional* except for the **mbean-stats** collector type. Specify key/value attribute pairs separated by semicolons, for example: `attribute1=21;attribute2=64`
     Attributes are passed when the action is triggered.

> **Note**
>
> The mbean-stats collector lets you use an MBean counter for statistics samples. When configuring the collector, the attributes **object-name** and **attribute-name** must be set so that the collector can find the attribute value of the particular MBean.
>
> For the **object-name** attribute, a variable *${server_name}* can be used that will be replaced with name of managed server on which the statistics collector is running.
>
> The following example shows a configuration retrieving the **ServerAppSessionCount** from the **SipServerRuntime** MBean on the current server.
>
> ```
> object-name="com.bea:ServerRuntime=${server_name},Name=$
> {server_name},Type=SipServerRuntime";attribute-name=ServerAppSessionCount
> ```
>
> For a complete list of Converged Application Server MBeans, see the *Oracle Communications Converged Application Server Java API Reference*.

3. Click **Create** to save your configuration changes.
4. Click the shopping cart, and then click **Commit Changes**.

About Thresholds

An overload protection threshold consists of a threshold value, a collector, sampling settings, and two lists of overload protection actions defined for an event handler.

Thresholds work in two modes: a sampling mode with a configurable interval and number of samples, and a real-time mode. For both modes, statistics samples are collected and calculated according to an selectable algorithm and compared to the threshold value. Each threshold has two events, **UP_EVENT** and **DOWN_EVENT**. When the threshold is breached upwards, the **UP_EVENT** event is triggered and when it is breached downwards, the **DOWN_EVENT** event is triggered.

For each event, you can configure a list of event handler actions. When an event is triggered, the overload protection framework will execute each action associated with the threshold event.

Configuring a Threshold

To configure an overload protection Threshold:

1. From the Remote Console, click **Custom Resources**, and then **sipserver**, and then **SIP Server**, and then **Overload Protections** and then **Thresholds**.
2. In the Thresholds table, click **New**.
   Enter the following information:
   - **Threshold Name**: *Required*. Enter a name for the action, for example:

     ```
     queueLengthThreshold
     ```

   - **Threshold Value**: *Required*. Enter the level of the threshold. This is the value that the threshold must *exceed* to trigger an event, for example:

     ```
     10.0
     ```

> **Note**
>
> The Threshold Value cannot be greater than 100.

- **Sampling Mode**: *Required*. Choose either **real-time** or **sampling** from the drop down list. In **real-time** mode, statistics are compared against the Threshold Value when every initial SIP message is received. No calculations are supported.
- **Sampling Interval**. *Required* when **sampling** mode is selected. Enter the interval at which samples should be taken in milliseconds, for example:

  ```
  1000
  ```

- **Sampling Number**. *Required* when **sampling** mode is selected. Enter the number of samples to be taken at each Sampling Interval, for example:

  ```
  5
  ```

- **Algorithm Name**: *Required*. Choose an appropriate algorithm to calculate samples.

Table 3-7 Algorithm Types

Algorithm Name	Description
percntile	Calculates the Pth percentile value of the samples. When **PERCNTILE** is selected, an **Algorithm Parameter** value must be provided.
average	Calculates the average of the samples (sum of samples divided by number of samples).
value	The straight value of the last sample.
rate	The sample rate calculated as (*last sample - first sample*)/(*sampling interval*).

- **Algorithm Parameter**: *Required* when the **PERCNTILE** algorithm is selected. Enter a percentile value that the threshold must match, for example:

  ```
  65
  ```

- **Enable**: *Optional*. Check Enable to enable the Threshold.

3. Choose the Actions to be executed when a threshold is breached upwards (if any) by moving an Action from the Available list to the Chosen list.
4. Choose the Actions to be executed when a threshold is breached downwards (if any) by moving an Action from the Available list to the Chosen list.
5. Click **Create** to save your configuration changes.
6. Click the shopping cart, and then **Commit Changes**.

Example: Configuring Overload Protection Based upon Session Rate

In the following example you create an overload protection scheme based upon the session rate. You begin by creating an event handler of the type com.oracle.trafficControl to react to traffic control events. Next, you create two actions that the event handler will initiate, one to reject SIP session traffic and another to accept SIP session traffic. You then create a statistics collector that reads counter information from the SipServerRuntime MBean, and you finally create a threhold that takes 5 samples every 1000 milliseconds and reacts on an upwards/downwards breach of a particular threshold value you set.

Once configured, when your threshold value is breached upwards, SIP traffic will be rejected until the threshold value is again breached downwards.

To configure a session rate overload protection scheme:

1. From the Remote Console, click **Custom Resources**, and then **sipserver**, and then **SIP Server**, and then **Overload Protections** and then **Event Handlers**.
2. In the Event Handlers table, click **New**, and enter **com.bea.wcp.sip.engine.server.olp.handler.ControlTrafficHandler** for the **Event Handler Name**.
3. Click **Create**.
4. Click **Custom Resources**, and then **sipserver**, and then **SIP Server**, and then **Overload Protections** and then **Actions**.
5. In the Actions table, click **New** and enter the following information:
   - **Action Name**: Enter **TrafficReject**.
   - **Event Handler**: Enter com.oracle.trafficControl.
   - **Action Type**: Enter **reject-traffic**.
6. Click **Create** to save your configuration changes.
7. In the Actions table, click **New** and enter the following information:
   - **Action Name**: Enter **TrafficAccept**.
   - **Event Handler**: Enter com.oracle.trafficControl.
   - **Action Type**: Enter **accept-traffic**.
8. Click **Custom Resources**, and then **sipserver**, and then **SIP Server**, and then **Overload Protections** and then **Statistics Collector**.
9. In the Statistics Collectors table, click **New** and enter the following information:
   - **Statistics Collector Name**: Enter **com.bea.wcp.sip.engine.server.olp.collector.MBeanCollector**.
   - **Statistics Collector Type**: Enter **mbean-stats**.
   - **Attributes**: Enter:

     ```
     object-name="com.bea:ServerRuntime=${server_name},Name=$
     {server_name},Type=SipServerRuntime";attribute-name=ServerAppSessionCount
     ```
10. Click **Create** to save your configuration changes.
11. Click **Custom Resources**, and then **sipserver**, and then **SIP Server**, and then **Overload Protections** and then **Thresholds**.
12. In the Thresholds table, click **New** and enter the following information:
    - **Threshold Name**: Enter **SessionRate**.
    - **Threshold Value**: Enter the threshold value you wish to use for the maximum number of sessions.

      > ⓘ **Note**
      >
      > The Threshold Value cannot be greater than 100.

    - **Sampling Mode**: Select **sampling** from the drop down list.

- **Sampling Interval**: Enter **1000** to take a sample every 1000 milliseconds.
- **Sampling Number**: Enter **5** to take 5 samples at each sampling interval.
- **Algorithm Name**: Select **rate** from the drop down list.
- **Statistics Collector**: Enter **com.bea.wcp.sip.engine.server.olp.collector.MBeanCollector**.
- Check **Enable**.

13. For **Actions for upward breach**, enter **TrafficReject**.
14. For **Actions for downward breach**, enter **TrafficAccept**.
15. Click **Create**.
16. Click the shopping cart, and then click **Commit Changes**.

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 engines, 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 the *Configuring and Using the Diagnostics Framework for Oracle WebLogic Server* 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:

- **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/Oracle_Home/wlserver/server/lib/console-ext/diagnostics-
console-extension.jar ~/ORACLE_HOME/Middleware/Oracle_Home/user_projects/domains/
base_domain/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 Remote 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, from the **Edit Tree** of the Remote Console, select the **Diagnostics**, and then **WLDF System Resources**, and then **New**. Create a new module with the watch rules and notifications required for monitoring your servers. Next, from the **Monitoring Tree**, select **Diagnostics**, and then **System Resource Controls**, and then the newly created resource. 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:

- Call state and timer statistics.
- Work manager statistics.

Instrumentation

> ⚠ **Warning**
>
> Not supported in this release.

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 below, as well as the WebLogic Server dye flags USER and ADDR. See Configuring the DyeInjection Monitor to Manage Diagnostic Contexts in *Oracle Fusion Middleware Configuring and Using the Diagnostics Framework for Oracle WebLogic Server* for more information.

Table 3-8 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.

Table 3-9 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.

Table 3-9 (Cont.) Converged Application Server Diagnostic Monitors

Monitor Name	Monitor Type	Scope	Pointcuts
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.
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

> **Warning**
>
> Not supported in this release.

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. From the **Edit Tree** of the Remote Console, select **Diagnostics**, and then **WLDF System Resources**.
2. Click **New**, provide a name, and click **Create**.
3. Under **WLDF System Resources**, select the new module.
4. In the **Targets** tab, select the server or servers to target, and click **Save**.
5. In the **Policy Configuration** tab, select **Enabled**, and click **Save**.
6. Add the DyeInjection monitor to the module:
   1. Click **Add/Remove**.
   2. Select the name of a monitor from the Available list (for example, DyeInjection), and use the arrows to move it to the Chosen list.
   3. Click **OK**.
   4. Select the newly-created monitor from the list of available monitors.
   5. 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
      ```

   6. Click **Save**.
7. Add one or more delegate monitors to the module:
   1. Return to the **Configuration** > **Instrumentation** tab for the new module.
   2. Click **Add/Remove**.
   3. 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.
   4. Click **OK**.
   5. Select the newly-created monitor from the list of available monitors.
   6. 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.
   7. Select the check box to EnableDyeFiltering.
   8. 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.
   9. Click **Save.**

> **Note**
>
> You can repeat the above steps to create additional delegate monitors.

8. If your domain is running in Production mode, click **Activate Changes**.

Configuring Application-Scoped Monitors

> **Warning**
>
> Not supported in this release.

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. The example below shows a sample configuration file that uses the **occas/Sip_Servlet_Before_Service** monitor.

Example 3-4 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* for more information about creating the **weblogic-diagnostics.xml** configuration file.

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 below.

Example 3-5 Sample Logging Servlet

```
<servlet>
    <servlet-name>msgTraceLogger</servlet-name>
    <servlet-class>com.bea.wcp.sip.engine.tracing.listener.TraceMessageListenerImpl</
servlet-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. Pattern-matching

Variables and Sample Values lists the parameters and parameter values that you can specify as **init-param** entries. Sample Logging Servlet 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.

As SIP requests are processed or SIP responses generated, the logging Servlet compares the message with the defined filtering patterns. 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.

> **Note**
>
> By default Converged Application Server logs both requests and responses. If you define a **request-pattern.xml** file, the response within the SIP session will be logged automatically.

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 3-6 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, require a variable (`var` element) that identifies the portion of the SIP message to evaluate. The table below 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 3-10 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.example.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. The example below 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 3-7 Logging Criteria Specified as init-param Elements

```
<servlet>
      <servlet-name>invTraceLogger</servlet-name>
      <servlet-class>com.bea.wcp.sip.engine.tracing.listener.TraceMessageListenerImpl</
servlet-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.

The example below 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 3-8   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 Remote 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 3-9 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.

Example 3-10   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.

4
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:

- Engine Server Configuration Reference (sipserver.xml)
- SIP Coherence Configuration Reference (coherence.xml)
- Diameter Configuration Reference (diameter.xml)

Engine Server Configuration Reference (sipserver.xml)

This chapter describes the Oracle Communications Converged Application Server engine server configuration file, **sipserver.xml**.

Overview of sipserver.xml

The **sipserver.xml** file is an XML document that configures the SIP container features provided by a Converged Application Server instance in a server installation. The **sipserver.xml** file is stored in the *domain_home***/config/custom** subdirectory where *domain_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 Remote Console to modify **sipserver.xml** indirectly, rather than editing the file manually with a text editor. Using the Remote 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 many systems when installing or upgrading Converged Application Server. When you manually edit **sipserver.xml**, you must restart Converged Application Server instances to apply your changes.

> ⚠ **Caution**
>
> Always use the **SIP Server** node in the Remote Console or the WLST utility to make changes to a running Converged Application Server deployment. See Configuring Converged Application Server 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 *domain_home***/config/custom/sipserver.xm**l file, where *domain_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. Restart or start servers to have your changes take effect:

   > ⚠ **Caution**
   >
   > Always use the SIP Server node in the Remote Console or the WLST utility to make changes to a running Converged Application Server deployment. See Configuring Converged Application Server 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/***wlserver/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 servers process expired timers. By default (when enable-timer-affinity is omitted from **sipserver.xml**, or is set to **false**), an engine server that polls the SIP call-state store for expired timers might process all available expired timers. When enable-timer-affinity is set to **true**, engine servers polling the SIP call-state store 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" for more information.

message-debug

The **message-debug** element enables and configures access logging with log rotation for Converged Application Server. Use this element only in a development environment, because access logging logs all SIP requests and responses.

To perform more selective logging in a production environment, see 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. The table below 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 preserving any routing decision taken by Converged Application Server. This configuration 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. Applications use the X-BEA-Proxy-Policy custom header 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 to route to the outbound proxy.

Table 4-1 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.

Table 4-1 (Cont.) Nested proxy Elements

Element	Description
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.

The example below 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 4-1 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" for more information.

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" for more information.

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" for more information.

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" for more information.

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" for more information.

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.

> **Note**
>
> The value of max-application-session-lifetime must be equal or greater than the default session length of every deployed application.

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's cluster address or the local server address, then the message is routed internally to the local server instead of being sent through the network.

You may want to disable this optimization if you feel that routing internal messages could skew the load on engine servers, and you prefer to route all requests through 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 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 cluster configured for an upgrade. Converged Application Server uses this mapping to ensure that engine 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, Converged Application Server uses the **cluster-loadbalancer-map** to forward the request to the correct cluster.

Each **cluster-loadbalancer-map** entry contains the two elements.

Table 4-2 Nested cluster-loadbalancer-map Elements

Element	Description
cluster-name	The configured name of an engine cluster.
sip-uri	The internal SIP URI that maps to the engine cluster. This corresponds to a virtual IP address that you have configured in your load balancer. The internal URI forwards requests to the correct cluster version during an upgrade.

The example below shows a sample **cluster-loadbalancer-map** entry used during an upgrade.

Example 4-2 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 the section on upgrading production Converged Application Server software in the *Converged Application Server Installation Guide* 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 4-3   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 servers to cache a portion of the call-state data locally, to improve performance with SIP-aware load balancers. When a local cache is used, an engine 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 call-state store), the engine locks the call state in the SIP call-state store but reads directly from its cache.

By default the engine 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 Using the Engine 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 enables or disables writing call state data to an RDBMS, or to a remote, geographically-redundant Converged Application Server installation. For sites that use 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 sub-elements.

Table 4-3 Nested persistence Elements

Element	Description
default-handling	Determines whether Converged Application Server observes persistence hints for RDBMS persistence 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. 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 call-state store. 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.

Table 4-3 (Cont.) Nested persistence Elements

Element	Description
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 cluster of the remote installation. You can also specify a comma-delimited list of addresses corresponding to each engine server. The URLs must specify the t3 protocol.

The example below shows a sample configuration that uses RDBMS storage for long-lived call state and geographically-redundant replication. Call states are replicated to two engine servers in a remote location.

Example 4-4 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>
```

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 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 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 minimizes 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 re-creating new connections. Opened connections are reused in a round-robin fashion. Opened connections remain open until they are explicitly closed by the remote address.

Connection reuse pools are not used for incoming requests from a configured address.

To configure a connection reuse pool, you define the following four nested elements.

Table 4-4 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 reuses 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.

The example below shows a sample connection-reuse-pool configuration having two pools.

Example 4-5 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.)

User Agents (UAs) deployed on Converged Application Server typically obtain GRUUs through 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. 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 **SIP Server** node of the Remote Console.

enable-rport

This element determines whether 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 Remote Console. In the Remote Console, select **Custom Resources**, and then **sipserver**, and then **SIP Server**.

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 through the gateway port.

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.

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 from the Remote Console under **Custom Resources**, and then **sipserver**, and then **SIP Server**.

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. 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>
```

Setting this element to **true** does not affect **100 Trying** responses.

SIP Coherence Configuration Reference (coherence.xml)

This chapter describes the Coherence configuration file, coherence.xml, for Oracle Communications Converged Application Server.

Overview of coherence.xml

The **coherence.xml** configuration file identifies servers that manage the concurrent call state for SIP applications, and specifies distributed cache settings. See "Configuring Coherence" for information on configuring Coherence.

The **coherence.xml** file resides in the *domain_home***/config/custom** subdirectory where *domain_home* is the root directory of Converged Application Server domain.

Editing coherence.xml

You can edit **coherence.xml** using either the Remote Console or a text editor. Changes to the configuration cannot be applied to servers dynamically; you must restart servers to change the SIP server configuration.

XML Schema

The schema file is bundled within the **wlss-descriptor-binding.jar** library, installed in the *Middleware_Home***/wlserver/sip/server/lib** directory where *Middleware_Home* is the path to the directory where WebLogic Server is installed.

Example coherence.xml File

The default coherence.xml file is shown.

Example 4-6 Default coherence.xml File

```
<?xml version='1.0' encoding='UTF-8'?>
<coherence-storage>
  <cache-config>
    <thread-count>20</thread-count>
    <partition-count>257</partition-count>
  </cache-config>
</coherence-storage>
```

XML Element Description

The coherence.xml file describes the elements that govern the Coherence distributed cache service.

Table 4-5 coherence.xml File Elements

Element	Description
thread-count	Specifies the number of threads used in the call-state Coherence cache service used by the SIP server. Oracle recommends that this value be a positive integer but you can specify 0 or -1 to obtain specific behaviors. See the `thread-count` element description in "Cache Configuration Elements" in *Developing Applications with Oracle Coherence* for more information.
partition-count	Specifies the number of partitions used in the call-state Coherence cache service used by the SIP server. You must specify a positive integer and should specify a prime number. See the `partition-count` element description in "Cache Configuration Elements" in *Developing Applications with Oracle Coherence* for more information.

Diameter Configuration Reference (diameter.xml)

This chapter describes the Oracle Communications Converged Application Server 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 start time. **diameter.xml** is stored in the *domain_home***/config/custom** subdirectory where *domain_home* is the root directory of the Converged Application Server domain.

Editing diameter.xml

> ⚠ **Warning**
>
> You should never move, modify, or delete the **diameter.xml** file during normal operations.

Oracle recommends using the Remote Console to modify **diameter.xml** indirectly, rather than editing the file manually with a text editor. Using the Remote 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 restart Diameter nodes to apply your changes.

> ⚠ **Caution**
>
> Always use the Diameter node in the Remote Console or the WLST utility, as described in Configuring Converged Application Server 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 *OCCAS_home***/config/custom/diameter.xml** file, where *OCCAS_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. Restart 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***/wlserver/sip/server/lib**, where *WL_home* is the path to the directory where WebLogic Server is installed.

Example diameter.xml File

See 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. 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 omission enables you to deploy the same Diameter web application to all servers in the engine 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 comma-separated list of IP addresses or DNS names of the remote interface(s) for a Diameter peer. The first address is the primary remote address and others are alternate remote addresses. When the transport protocol is SCTP, all IP addresses will be associated with the remote SCTP endpoint. When the transport protocol is TCP or TLS, only the first address will be used.

See "Validate SCTP Peer Address" for how the Converged Application Server behaves when a Diameter peer offers an IP address not in this list.

If you do not specify an address, the host identity is used.

> **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 or SCTP port number for this Diameter peer. The default port is 3588. If the value is 0, the system assigns an ephemeral port.

validate-peer-address

Enable this checkbox to validate the remote SCTP connection addresses of a Diameter Peer. If you enable this validation, only configured Peer Addresses are allowed in remote Peer Addresses offered during SCTP association setup. An SCTP association will be closed if any unknown remote Peer Address is present.

Possible values are `true` or `false`.

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".

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".

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 Rx clients, Diameter relay agents, or Home Subscriber Servers (HSS). 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 and for all other engine 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 and the HSS.

host

Specifies the host identity for a Diameter peer.

address

Specifies comma-separated list of IP addresses or DNS names of the remote interface(s) for a Diameter peer. The first address is the primary remote address and others are alternate remote addresses. When the transport protocol is SCTP, all IP addresses will be associated with the remote SCTP endpoint. When the transport protocol is TCP or TLS, only the first address will be used.

See "Validate SCTP Peer Address" for how Converged Application Server behaves when a Diameter peer offers an IP address not in this list.

If you do not specify an address, the host identity is used.

port

Specifies the TCP or TLS or SCTP port number for this Diameter peer. The default port is 3588. If the value is 0, the system assigns an ephemeral port.

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 and 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:

- local

- relay
- proxy
- redirect

server+

Specifies one or more target servers for this route. 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 "route" for more information.

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.