BEA WebLogic SIP Server™

Configuring and Managing WebLogic SIP Server

Version 2.2
Revised: May 16, 2006

Copyright

Restricted Rights Legend

Trademarks and Service Marks

Contents

3. Configuring Data Tier Partitions and Replicas

4. Configuring Engine Tier Container Properties

5. Configuring Diameter Sh Client Nodes and Relay Agents

6. Capacity Planning for WebLogic SIP Server Deployments

7. Managing WebLogic SIP Server Network Resources

10.Logging SIP Requests and Responses

11.Avoiding and Recovering From Server Failures

12.Configuring SNMP

A. Upgrading Deployed SIP Applications

B. Upgrading Software and Converged Applications

CHAPTER 1

Overview of the WebLogic SIP Server Architecture

The following sections provide an overview of the WebLogic SIP Server 2.2 architecture:

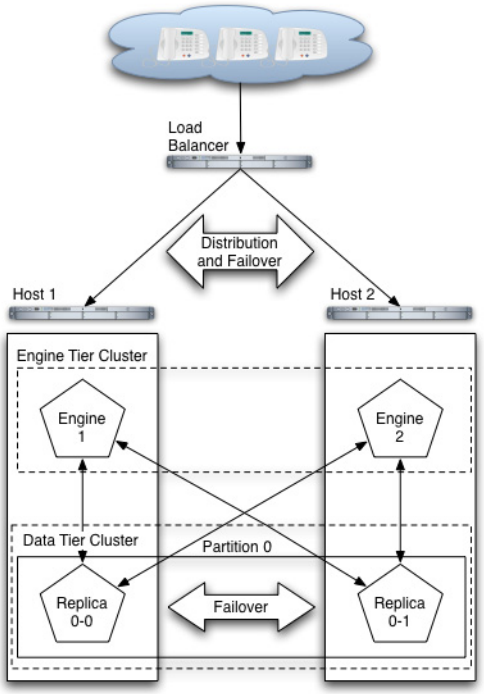
Goals of the WebLogic SIP Server Architecture

WebLogic SIP Server 2.2 is designed to provide a highly scalable, highly available, performant server for deploying SIP applications. The WebLogic SIP Server 2.2 architecture is simple to manage and easily adaptable to make use of available hardware. The basic architecture consists of these components:

- Load Balancer
- Engine Tier
- Data tier

Figure 1-1 shows the components of a basic WebLogic SIP Server installation. The sections that follow describe each component of the architecture in more detail.

Figure 1-1 WebLogic SIP Server 2.2 Architecture



Load Balancer

Although it is not provided as part of the WebLogic SIP Server product, a load balancer (or multiple load balancers) is an essential component of any production WebLogic SIP Server installation. The primary goal of a load balancer is to provide a single public address that distributes incoming SIP requests to available servers in the WebLogic SIP Server engine tier. Distribution of requests ensures that WebLogic SIP Server engines are fully utilized.

Most load balancers have configurable policies to ensure that client requests are distributed according to the capacity and availability of individual machines, or according to any other load policies required by your installation. Some load balancers provide additional features for managing SIP network traffic, such as support for routing policies based on source IP address, port number, or other fields available in SIP message headers. Many load balancer products also provide additional fault tolerance features for telephony networks, and can be configured to

consistently route SIP requests for a given call to the same engine server on which the call was initiated.

In a WebLogic SIP Server installation, the load balancer is also essential for performing maintenance activities such as upgrading individual servers (WebLogic SIP Server software or hardware) or upgrading applications without disrupting existing SIP clients. The Administrator modifies load balancer policies to move client traffic off of one or more servers, and then performs the required upgrades on the unused server instances. Afterwards, the Administrator modifies the load balancer policies to allow client traffic to resume on the upgraded servers.

BEA provides detailed information for setting up load balancers with the WebLogic SIP Server engine tier for basic load distribution. See "Configuring Load Balancer Addresses" on page 7-2 to configure a load balancer used with WebLogic SIP Server and "Upgrading Software and Converged Applications" on page B-1 to use a load balancer to perform system and application upgrades.

Engine Tier

The engine tier is a cluster of WebLogic SIP Server instances that hosts the SIP Servlets that provide features to SIP clients. In many configurations, server instances in the engine tier host only SIP Servlets. SIP session information is not persisted in the engine tier, but is obtained by querying the data tier, which also provides replication and failover services for SIP session data.

The primary goal of the engine tier is to provide maximum throughput and low response time to SIP clients. As the number of calls, or the average duration of calls to your system increases, you can easily add additional server instances to the engine tier to manage the additional load.

Note that although the engine tier consists of multiple WebLogic SIP Server instances, you manage the engine tier as a single, logical entity; SIP Servlets are deployed uniformly to all server instances (by targeting the cluster itself) and the load balancer need not maintain an affinity between SIP clients and servers in the engine tier.

Notes: WebLogic SIP 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 "Tuning JVM Garbage Collection for Production Deployments" on page F-1 for suggestions about maximizing JVM performance in a production domain.

Because the engine tier relies on data tier servers in order to retrieve call state data, BEA recommends using dual Network Interface Cards (NICs) on engine and data tier machines to provide redundant network connections.

Data tier

The data tier is a cluster of WebLogic SIP Server instances that provides a high-performance, highly-available, in-memory database for storing and retrieving the session state data for SIP Servlets. The goals of the data tier are as follows:

- To provide reliable, performant storage for session data required by SIP applications in the WebLogic SIP Server engine tier.
- To enable administrators to easily scale hardware and software resources as necessary to accommodate the session state for all concurrent calls.

Within the data tier, session data is managed in one or more "partitions" where each partition manages a fixed portion of the concurrent call state. For example, in a system that uses two partitions, the first partition manages one half of the concurrent call state (for example, sessions A through M) while the second partition manages another half of the concurrent call states (sessions N through Z). With three partitions, each partition manages a third of the call state, and so on. Additional partitions can be added as necessary to manage a large number of concurrent calls.

Within each partition, multiple servers can be added to provide redundancy and failover should other servers in the partition fail. When multiple servers participate in the same partition, the servers are referred to as "replicas" because each server maintains a duplicate copy of the partition's call state. For example, if a two-partition system has two servers in the first partition, each server manages a replica of call states A through M. If one or more servers in a partition fails or is disconnected from the network, any available replica can automatically provide call state data to the engine tier. In WebLogic SIP Server 2.2, the data tier can have a maximum of three replicas, providing two levels of redundancy.

See "Configuring Data Tier Partitions and Replicas" on page 3-1 for more information about configuring the data tier for high availability. See "Determining Performance Goals" on page 6-2 for information about planning the hardware resources required in the data tier.

Note: Because the engine tier relies on data tier servers in order to retrieve call state data, BEA recommends using dual Network Interface Cards (NICs) on engine and data tier machines to provide redundant network connections.

Example Hardware Configuration

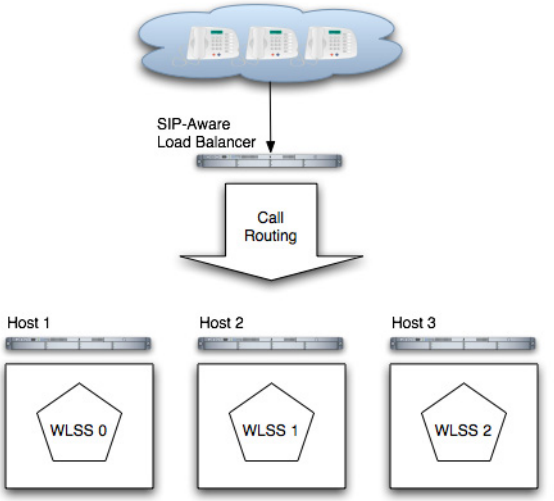
WebLogic SIP Server's flexible architecture enables you to configure engine and data tiers in a variety of ways to support high throughput and/or provide high availability. See "Capacity Planning for WebLogic SIP Server Deployments" on page 6-1 for detailed information about scaling the engine and data tiers to suit the needs of your organization.

Alternate Configurations

Not all WebLogic SIP Server requirements require the performance and reliability provided by multiple servers in the engine and data tiers. On a development machine, for example, it is generally more convenient to deploy and test applications on a single server, rather than a cluster of servers.

WebLogic SIP Server enables you to combine engine and data tier services on a single server instance when replicating call states is unnecessary. In a combined-tier configuration, the same WebLogic SIP Server instance provides SIP Servlet container functionality and also manages the call state for applications hosted on the server. Although the combined-tier configuration is most commonly used for development and testing purposes, it may also be used in a production environment if replication is not required for call state data. Figure 1-2 shows an example deployment of multiple combined-tier servers in a production environment.

Figure 1-2 Single-Server Configurations with SIP-Aware Load Balancer



Because each server in a combined-tier server deployment manages only the call state for the applications it hosts, the load balancer must be fully "SIP aware." This means that the load balancer actively routes multiple requests for the same call to the same WebLogic SIP Server instance. If requests in the same call are not pinned to the same server, the call state cannot be retrieved. Also keep in mind that if a WebLogic SIP Server instance fails in the configuration shown in Figure 1-2, all calls handled by that server are lost.

CHAPTER 2

Overview of WebLogic SIP Server Configuration and Management

The following sections provide an overview of how to configure and manage WebLogic SIP Server deployments:

Shared Configuration Tasks for WebLogic SIP Server and WebLogic Server

WebLogic SIP Server 2.2 is based on the award-winning WebLogic Server 8.1 application server, and many system-level configuration tasks are the same for both products. This manual addresses only those system-level configuration tasks that are unique to WebLogic SIP Server 2.2, such as tasks related to network and security configuration and cluster configuration for the engine and data tiers.

HTTP server configuration and other basic configuration tasks such as server logging, startup, and shutdown, are addressed in the WebLogic Server 8.1 Documentation.

WebLogic SIP Server Configuration Overview

The SIP Servlet container and call state data replication features of WebLogic SIP Server are implemented in an Enterprise Application (EAR), named `sipserver`, that is deployed on the WebLogic Server 8.1 product. The same `sipserver` application code is deployed to engine and data tier instances of WebLogic SIP Server. The `sipserver.xml` and `datatier.xml` configuration files included in the `sipserver` application determines the role of each server instance.

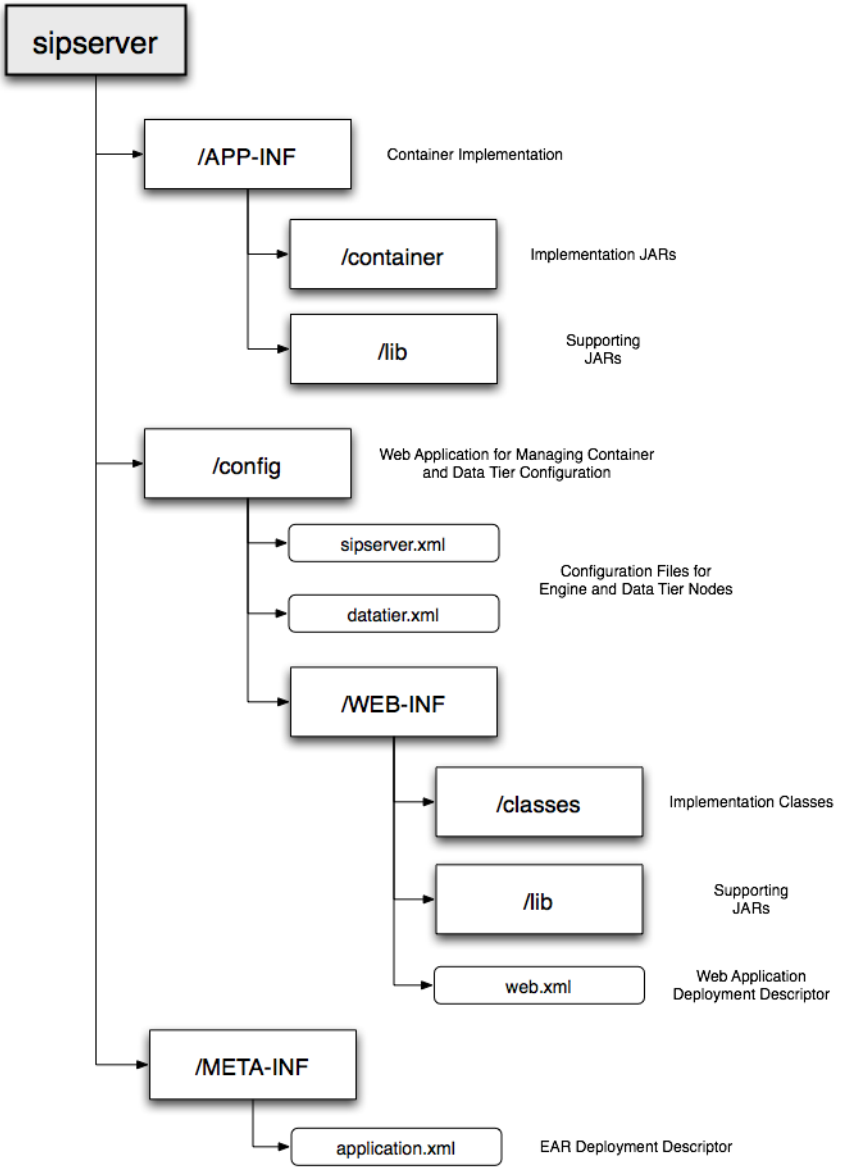
A server instance initiates data tier services only if the server name is designated as a data tier server in the `datatier.xml` configuration file. Servers that are not part of the data tier provide SIP Servlet container features, and container properties are configured based on entries in the `sipserver.xml` configuration file.

The `sipserver` EAR is deployed in exploded archive format, and is automatically copied to the top level of the domain directory when you create a domain using the Configuration Wizard (for example, `c:\bea\user_projects\domains\mydomain\sipserver`). Figure 2-1, "sipserver Web Application Contents," on page 2-3 summarizes the basic structure of the `sipserver` application.

Only one copy of the deployment files are required for the engine and data tier servers in the domain, because the same `sipserver.xml` and `datatier.xml` file are used by both clusters.

WARNING: Never modify, redeploy, or undeploy the `sipserver` implementation application on a running production server. Always use the SIP Servers node in the Administration Console or the WLST utility, as described in "Configuring Engine Tier Container Properties" on page 4-1, to make changes to a running WebLogic SIP Server deployment.

Figure 2-1 sipserver Web Application Contents



Configuration Implementation

The `config` Web Application in `sipserver` provides the logic for parsing the `sipserver.xml` and `datatier.xml` files and applying configuration changes, as well as the implementation of the Administration Console extensions for configuring SIP features. Because the `sipserver.xml` and `datatier.xml` configuration files are included within a Web Application, changing the configuration for a WebLogic SIP Server deployment involves modifying these files.

Configuration changes to SIP Servlet container properties can be applied dynamically to a running server by using the Administration Console SIP Servers node or from the command line using the WLST utility. Configuration for data tier nodes cannot be changed dynamically, so you must reboot data tier servers in order to change the number of partitions or replicas.

A special MBean included in the SIP Server implementation, `ConfigManagerRuntimeMBean`, handles locking and modifying the `sipserver.xml` configuration file in response to JMX commands, as well as applying configuration changes to running servers. Therefore, when you edit a WebLogic SIP Server configuration using the Administration Console or using JMX-based utilities, `ConfigManagerRuntimeMBean` manages updates to your configuration files transparently. See "Configuring Engine Tier Container Properties" on page 4-1. If you want to modify a configuration file outside of JMX, you must do so while WebLogic SIP Server is shut down. See Engine Tier Configuration Reference (sipserver.xml) and Data Tier Configuration Reference (datatier.xml) in the *Configuration Reference Manual*.

Diameter Configuration

The Diameter protocol implementation and configuration files are deployed as a Web Application separate from the `sipserver` EAR. The Diameter Web Application configures a specific Diameter protocol application to provide Diameter node functionality. WebLogic SIP Server provides the Diameter protocol applications to support the following node types:

- Diameter Sh interface client node
- Diameter relay node
- HSS simulator node (suitable for testing and development only, not for production deployment)

The Diameter Web Application is deployed only to servers that need to act as Diameter client nodes or relay agents, or to servers that want to provide HSS simulation capabilities. The actual function of the server depends on the configuration defined in the `diameter.xml` file.

The Diameter Web Application is structured as follows:

- `WEB-INF/config` contains the `diameter.xml` file that configures Diameter node functionality. This file uses a configuration schema similar to the one used in OpenDiameter (see the Diameter Configuration Reference (diameter.xml).)
- `WEB-INF/lib` contains the Diameter application implmentations packaged as a JAR file.

See "Configuring Diameter Sh Client Nodes and Relay Agents" on page 5-1 for instructions to configure the Diameter Web Application in a WebLogic SIP Server domain. See Using the IMS Sh Interface (Diameter) in *Developing Applications with WebLogic SIP Server* for more information about using the Sh profile API.

Understanding Staging Modes for the sipserver Application

The `sipserver` application provides both the SIP container implementation and the engine and data tier configuration for all servers in a WebLogic SIP Server domain. To ensure that all engine tier and data tier servers are configured in a consistent manner, you must deploy the same version of the `sipserver` application to all WebLogic SIP Server instances in the domain.

The *deployment staging mode* determines how an application's deployment files are made available to a Managed Server in the domain. WebLogic SIP Server uses two basic deployment staging modes for engine and data tier instances: nostage mode and stage mode. With nostage mode, the `sipserver` deployment files are not copied to engine and data tier server machines. Instead, all WebLogic SIP Server machines must be able to access the same `sipserver` deployment files from a central location. In a multiple-server, replicated environment, this means that the `sipserver` application must be published on a shared filesystem so that all server instances can access the files at startup time. If one or more servers cannot access they deployment files, they cannot deploy or configure SIP container services may be unable to communicate with other WebLogic SIP Server instances. Nostage mode is generally used for single-server domain deployments, or when the deployment files can be easily accessed over a shared filesystem (for example, a network share or a multi-homed machine).

With stage mode deployment, the Administration Server first copies the `sipserver` application deployment files to data and engine tier servers before deployment. The copied files reside in the staging directory for each Managed Server instance (the root server directory, by default). Stage mode is used for most multiple-server domains, and this mode is required if the `sipserver` deployment files cannot be easily shared among engine and data tier server machines.

The Configuration Wizard automatically configures servers in a replicated domain to use stage mode. Single-server (example) domains are to use nostage mode. See Changing a Server Staging

Mode or Staging Directory in the WebLogic Server 8.1 Documentation for information about how to verify or change a Managed Server's deployment staging mode.

Startup Sequence for a WebLogic SIP Server Domain

Note: WebLogic SIP 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 "Tuning JVM Garbage Collection for Production Deployments" on page F-1 for suggestions about maximizing JVM performance in a production domain.

WARNING: When you configure a domain with multiple engine and data tier servers, you must accurately synchronize all system clocks to a common time source (to within one or two milliseconds) in order for the SIP protocol stack to function properly. See Configuring NTP for Accurate SIP Timers in *Configuring and Managing WebLogic SIP Server* for more information.

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

1. **Start the Administration Server for the domain.** Start the Administration Server in order to provide the initial configuration to engine and data tier servers in the domain. The Administration Server can also be used to monitor the startup/shutdown status of each Managed Server. You generally start the Administration Server by using either the `startAdminServer.cmd` script installed with the Configuration Wizard, or a custom startup script.

2. **Start data tier servers in each partition.** The engine tier cannot function until servers in the data tier are available to manage call state data. Although all replicas in each partition need not be available to begin processing requests, at least one replica in each configured partition must be available in order to manage the concurrent call state. All replicas should be started and available before opening the system to production network traffic.

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

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

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

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

Methods and Tools for Performing Configuration Tasks

WebLogic SIP Server provides several mechanisms for changing the configuration of the SIP Servlet container:

- "Administration Console" on page 2-7
- "Upgrade Utility" on page 2-7
- "WebLogic Scripting Tool (WLST)" on page 2-8
- "Additional Configuration Methods" on page 2-8

Administration Console

WebLogic SIP Server provides an Administration Console extension that allows you to modify and monitor SIP Servlet container and data tier configuration properties using a graphical user interface. The Administration Console for WebLogic SIP Server is similar to the console available in WebLogic Server 8.1. All SIP Server configuration and monitoring is available via the SIP Server node in the left pane. See "Configuring Engine Tier Container Properties" on page 4-1 for more information about configuring the SIP Servlet container using the Administration Console.

Upgrade Utility

The WebLogic SIP Server upgrade utility, `wlss.UpgradeConfig`, helps you migrate an earlier WebLogic SIP Server configuration to a new WebLogic SIP Server 2.2 configuration. `wlss.UpgradeConfig` operates by taking an existing `sipserver.xml` configuration file and recreating the earlier configuration using the latest `sipserver.xml` schema. For more information about upgrading a configuration, see "Upgrading a WebLogic SIP Server 2.0.x Configuration to Version 2.2" on page D-1.

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 WebLogic SIP Server domain. "Configuring Engine Tier Container Properties" on page 4-1 provides instructions for modifying SIP Servlet container properties using WLST.

Additional Configuration Methods

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

Editing Configuration Files

You may also edit `sipserver.xml` or `datatier.xml` by hand, following the respective schemas described in Engine Tier Configuration Reference (sipserver.xml) and Data Tier Configuration Reference (datatier.xml) in the *Configuration Reference Manual*.

If you edit `sipserver.xml` by hand, you must manually reboot all servers to apply the configuration changes.

WARNING: Never redeploy or undeploy the `sipserver` implementation application on a running server. Always use the SIP Servers node in the Administration Console or the WLST utility, as described in "Configuring Engine Tier Container Properties" on page 4-1, to make changes to a running WebLogic SIP Server deployment.

Data tier properties, such as the number of call state partitions and replicas, can never be changed while data tier server instances are running. If you edit `datatier.xml`, the changes are not applied until the data tier servers are rebooted.

Custom JMX Applications

WebLogic SIP Server properties are represented by JMX-compliant MBeans, and access to these MBeans and `sipserver.xml` is managed through the special runtime MBean, `com.bea.wcp.sip.management.runtime.ConfigManagerRuntimeMBean`. You can therefore program JMX application to configure SIP container properties using WebLogic SIP Server MBeans.

The general procedure for modifying WebLogic SIP Server MBean properties using JMX is described in "Configuring Container Properties Using WLST (JMX)" on page 4-4 (WLST itself is a JMX-based application). For more information about the individual MBeans used to manage SIP container properties, see the WebLogic SIP Server Javadocs.

Administration Server Best Practices

The Administration Server in a WebLogic SIP Server 2.0.2 installation is required only for configuring, deploying, and monitoring J2EE services and applications; all SIP container configuration is performed using the container's `sipserver.xml` configuration file.

Note: If an Administration Server fails due to a hardware, software, or network problem, only management, deployment, and monitoring operations are affected. **Managed Servers do not require the Administration Server for continuing operation; J2EE applications and SIP features running on Managed Server instances continue to function even if the Administration Server fails.**

BEA recommends the following best practices for configuring Administration Server and Managed Server instances in your WebLogic SIP Server domain:

- Run the Administration Server instance on a dedicated machine. The Administration Server machine should have a memory capacity similar to Managed Server machines, although a single CPU is generally acceptable for administration purposes.
- Increase the threads available in the `weblogic.admin.RMI` and `weblogic.admin.HTTP` execute queues to match the number of managed servers in your system.
- Configure all Managed Server instances to use Managed Server Independence. This feature allows the Managed Servers to restart even if the Administration Server is unreachable due to a network, hardware, or software failure. See Replicating a Domain's Configuration Files for Managed Server Independence in the WebLogic Server 8.1 documentation.
- Configure the Node Manager utility to automatically restart all Managed Servers in the WebLogic SIP Server domain. See Configuring, Starting, and Stopping Node Manager in the WebLogic Server 8.1 documentation.

Should an Administration Server instance or machine fail, remember that 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 WebLogic SIP Server instances). On Managed Servers, WebLogic SIP Server traps are generated even in the absence of the Administration Server.

To resume normal management activities, restart the failed Administration Server instance as soon as possible.

Adding threads to weblogic.admin.RMI and weblogic.admin.HTTP

You must increase the default size of the `weblogic.admin.RMI` and `weblogic.admin.HTTP` execute queues to ensure that an Administration Server can configure and monitor the large number of Managed Server instances deployed in a typical WebLogic SIP Server system. The number of threads in each queue should, match the number of deployed Managed Servers in both the engine and data tier clusters. By default, `weblogic.admin.HTTP` contains three threads and `weblogic.admin.RMI` contains two threads.

`weblogic.admin.RMI` and `weblogic.admin.RMI` are internal execute queues and are *not* displayed for configuration in the Administration Console. To add threads to the default queues, you must manually edit the `config.xml` file for your domain to specify the queue configuration. Listing 2-1 highlights the configuration entries required for managing 10 servers (10 threads in each queue). Note that

Listing 2-1 Increasing the Thread Count in Administration Server Execute Queues

```
<?xml version="1.0" encoding="UTF-8"?>
<Domain ConfigurationVersion="8.1.5.0" Name="my_domain">
    <Cluster MulticastAddress="237.0.0.1" Name="BEA_ENGINE_TIER_CLUST"/>
    <Cluster MulticastAddress="237.0.0.2" Name="BEA_DATA_TIER_CLUST"/>
    <Server ListenAddress="admin_server_address" ListenPort="7001"
        Name="my_admin_server" NativeIOEnabled="true"
        ReliableDeliveryPolicy="RMDefaultPolicy" ServerVersion="8.1.5.0">
        <SSL Enabled="false" HostnameVerificationIgnored="false"
            IdentityAndTrustLocations="KeyStores" Name="my_admin_server"/>
        <ExecuteQueue Name="weblogic.kernel.Default"/>
```

```
        <ExecuteQueue Name="sip.tracing.domain" QueueLength="1024"
            ThreadCount="1" ThreadsMaximum="1" ThreadsMinimum="1"/>
        <ExecuteQueue Name="weblogic.admin.RMI" ThreadCount="10"/>
        <ExecuteQueue Name="weblogic.admin.HTTP" ThreadCount="10"/>
    </Server>
```

Common Configuration Tasks

General administration and maintenance of WebLogic SIP Server requires that you manage both WebLogic Server configuration properties and WebLogic SIP Server container properties. These common configuration tasks are summarized in Table 2-1.

Table 2-1 Common WebLogic SIP Server Configuration Tasks

Task	Description
"Configuring Engine Tier Container Properties" on page 4-1	• Configuring SIP Container Properties using the Administration Console • Using WLST to perform batch configuration
"Configuring Data Tier Partitions and Replicas" on page 3-1	• Assigning WebLogic SIP Server instances to the data tier partitions • Replicating call state using multiple data tier instances
"Managing WebLogic SIP Server Network Resources" on page 7-1	• Configuring WebLogic Server network channels to handling SIP and HTTP traffic • Setting up multi-homed server hardware • Configuring load balancers for use with WebLogic SIP Server
Configuring Digest Authentication in *Configuring Security for WebLogic SIP Server*	• Configuring the LDAP Digest Authentication Provider • Configuring a trusted host list
"Logging SIP Requests and Responses" on page 10-1	• Configuring logging Servlets to record SIP requests and responses. • Defining log criteria for filtering logged messages • Maintaining WebLogic SIP Server log files

CHAPTER 3

Configuring Data Tier Partitions and Replicas

The following sections describe how to configure WebLogic SIP Server instances that make up the data tier cluster of a deployment:

Overview of Data Tier Configuration

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

The data tier cluster is arranged in one or more *partitions*. A partition consists of one or more data tier server instances that manage the same portion of the concurrent call state data. In a single-server WebLogic SIP Server installation, or in a two-server installation where one server resides in the engine tier and one resides in the data tier, all call state data is maintained in a single

partition. Multiple partitions are required when the size of the concurrent call state exceeds the maximum size that can be managed by a single server instance. When more than one partition is used, the concurrent call state is split among the partitions, and each partition manages an separate portion of the data. For example, with a two-partition data tier, one partition manages the call state for half of the concurrent calls (for example, calls A through M) while the second partition manages the remaining calls (N through Z).

In most cases, the maximum call state size that can be managed by an individual server corresponds to the Java Virtual Machine limit of approximately 1.6GB per server. See "Capacity Planning for WebLogic SIP Server Deployments" on page 6-1 for more information.

Additional servers can be added to the data tier to manage copies of the call state data. When multiple servers are part of the same partition, each server manages a copy of the same portion of the call data, referred to as a *replica* of the call state. If any server in a partition fails or cannot be contacted due to a network failure, another replica in the partition supplies the call state data to the engine tier.

datatier.xml Configuration File

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

The sections that follow show examples of the `datatier.xml` contents for common data tier configurations. See also Data Tier Configuration Reference (datatier.xml) in the *Configuration Reference Manual* for full information about the XML Schema and elements.

Configuration Requirements and Restrictions

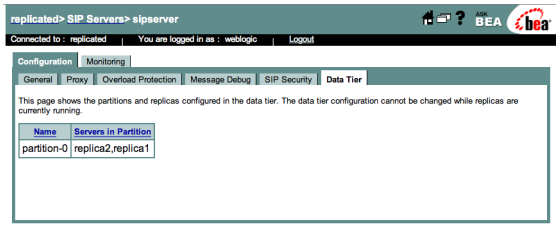
All servers that participate in the data tier should be members of the same WebLogic Server cluster. The cluster configuration enables each server to monitor the status of other servers. Using a cluster also enables you to easily target the `sipserver` application to all servers for deployment.

For high reliability, you can configure up to 3 replicas within a partition.

The data tier configuration cannot be changed dynamically. You must restart servers in the data tier in order to change data tier membership or reconfigure partitions or replicas. You can view

the current data tier configuration using the Configuration->Data Tier page of the WebLogic SIP Server Administration Console, as shown in Figure 3-1.

Figure 3-1 Administration Console Display of Data Tier Configuration (Read-Only)



Best Practices for Configuring and Managing Data Tier Servers

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

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

Data tier servers can have one of three different statuses:

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

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

WebLogic SIP Server automatically divides the call state evenly over all configured partitions.

Example Data Tier Configurations and Configuration Files

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

Data Tier with One Partition

A single-partition, single-server data tier represents the simplest data tier configuration. Listing 3-1 shows a data tier configuration for a single-server deployment.

Listing 3-1 Data Tier Configuration for Small Deployment

```
<st:data-tier
xmlns:st="http://bea.com/wcp/sip/management/internal/webapp">
 <st:partition>
   <st:name>Partition0</st:name>
   <st:server-name>DataNode0-0</st:server-name>
</st:partition>
</st:data-tier>
```

To add a replica to an existing partition, simply define a second `server-name` entry in the same partition. For example, the `datatier.xml` configuration file shown in Listing 3-2 recreates the two-replica configuration shown in Figure 6-3, "Small Deployment with High Availability," on page 6-8.

Listing 3-2 Data Tier Configuration for Small Deployment with Replication

```
<st:data-tier
xmlns:st="http://bea.com/wcp/sip/management/internal/webapp">
 <st:partition>
   <st:name>Partition0</st:name>
   <st:server-name>DataNode0-0</st:server-name>
```

```
   <st:server-name>DataNode0-1</st:server-name>
</st:partition>
</st:data-tier>
```

Data Tier with Two Partitions

Multiple partitions can be easily created by defining multiple `partition` entries in `datatier.xml`, as shown in Listing 3-3.

Listing 3-3 Two-Partition Data Tier Configuration

```
<st:data-tier
xmlns:st="http://bea.com/wcp/sip/management/internal/webapp">
<st:partition>
   <st:name>Partition0</st:name>
   <st:server-name>DataNode0-0</st:server-name>
</st:partition>
<st:partition>
   <st:name>Partition1</st:name>
   <st:server-name>DataNode0-1</st:server-name>
</st:partition>
</st:data-tier>
```

Data Tier with Two Partitions and Two Replicas

Replicas of the call state can be added by defining multiple data tier servers in each partition. Figure 6-4, "Medium-Sized Deployment," on page 6-9 shows a system having two partitions with two servers (replicas) in each partition. Listing 3-4 shows the `datatier.xml` configuration file used to define this data tier.

Listing 3-4 Data Tier Configuration for Small Deployment

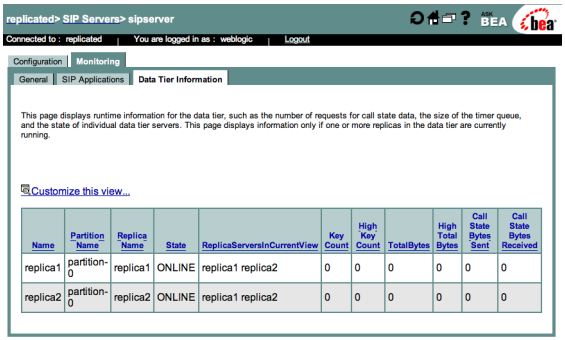
```
<st:data-tier
xmlns:st="http://bea.com/wcp/sip/management/internal/webapp">
 <st:partition>
   <st:name>Partition0</st:name>
   <st:server-name>DataNode0-0</st:server-name>
   <st:server-name>DataNode0-1</st:server-name>
</st:partition>
<st:partition>
   <st:name>Partition1</st:name>
   <st:server-name>DataNode1-0</st:server-name>
   <st:server-name>DataNode1-1</st:server-name>
</st:partition>
</st:data-tier>
```

Monitoring and Troubleshooting Data Tier Servers

A runtime MBean, `com.bea.wcp.sip.management.runtime.ReplicaRuntimeMBean`, provides valuable information about the current state and configuration of the data tier. See the WebLogic SIP Server JavaDocs for a description of the attributes provided in this MBean.

Many of these attributes can be viewed using the SIP Servers Monitoring->Data Tier Information tab in the Administration Console, as shown in "Data Tier Monitoring in the Administration Console" on page 3-7.

Figure 3-2 Data Tier Monitoring in the Administration Console



Listing 3-5 shows a simple WLST session that queries the current attributes of a single Managed Server instance in a data tier partition. Table 3-1, "ReplicaRuntimeMBean Method and Attribute Summary," on page 3-9 describes the MBean services in more detail.

Listing 3-5 Displaying ReplicaRuntimeMBean Attributes

```
connect('weblogic','weblogic','t3://datahost1:7001')
custom()
cd
('mydomain:Location=dataserver0-0,Name=dataserver0-0,ServerRuntime=dataser
ver0-0,Type=ReplicaRuntime')
ls()
-rw-   BytesReceived                          0
-rw-   BytesSent                              0
-rw-   CachingDisabled                        true
-rw-   CurrentViewId                          0
-rw-   DataItemCount                          0
-rw-   DataItemsToRecover                     0
-rw-   HighKeyCount                           0
-rw-   HighTotalBytes                         0
```

```
-rw-   KeyCount                                    0
-rw-   MBeanInfo                                  weblogic.management.tools.In
fo@194d9d5
-rw-   Name                                       myserver1
-rw-   ObjectName
mydomain:Location=dataserver0-0,
Name=dataserver0-0,ServerRuntime=dataserver0-0,Type=ReplicaRuntime
-rw-   Parent                                     mydomain:Location=dataserver0-0,
Name=myserver1,Type=ServerRuntime
-rw-   PartitionId                                0
-rw-   PartitionName                              partition-0
-rw-   Registered                                 false
-rw-   ReplicaId                                  0
-rw-   ReplicaServersInCurrentView
java.lang.String[dataserver0-0]
-rw-   ReplicasInCurrentView                      [I@169454d
-rw-   State                                      ONLINE
-rw-   TimerQueueSize                             0
-rw-   TotalBytes                                 0
-rw-   Type                                       ReplicaRuntime

-rwx   dumpState                                  void :
-rwx   preDeregister                              void :
```

Table 3-1 ReplicaRuntimeMBean Method and Attribute Summary

Method/Attribute	Description
`dumpState()`	Records the entire state of the selected data tier server instance to the WebLogic SIP Server log file. You may want to use the `dumpState()` method to provide additional diagnostic information to a Technical Support representative in the event of a problem.
`BytesReceived`	The total number of bytes received by this data tier server. Bytes are received as servers in the engine tier provide call state data to be stored.
`BytesSent`	The total number of bytes sent from this data tier server. Bytes are sent to engine tier servers when requested to provide the stored call state.
`CurrentViewId`	The current view ID. Each time the layout of the data tier changes, the view ID is incremented. For example, as multiple servers in a data tier cluster are started for the first time, the view ID is incremented when each server begins participating in the data tier. Similarly, the view is incremented if a server is removed from the data tier, either intentionally or due to a failure.
`DataItemCount`	The total number of stored call state keys for which this server has data. This attribute may be lower than the `KeyCount` attribute if the server is currently recovering data.
`DataItemsToRecover`	The total number of call state keys that must still be recovered from other replicas in the partition. A data tier server may recover keys when it has been taken offline for maintenance and is then restarted to join the partition.
`HighKeyCount`	The highest total number of call state keys that have been managed by this server since the server was started.
`HighTotalBytes`	The highest total number of bytes occupied by call state data that this server has managed since the server was started.

Table 3-1 ReplicaRuntimeMBean Method and Attribute Summary

Method/Attribute	Description
`KeyCount`	The number of call data keys that are stored on the replica.
`PartitionId`	The numerical partition ID (from 0 to 7) of this server's partition.
`PartitionName`	The name of this server's partition.
`ReplicaId`	The numerical replica ID (from 0 to 2) of this server's replica.
`ReplicaName`	The name of this server's replica.
`ReplicaServersInCurrentView`	The names of other WebLogic SIP Server instances that are participating in the partition.
`State`	The current state of the replica. Data tier servers can have one of three different statuses: • `ONLINE`—indicates that the server is available for managing call state transactions. • `OFFLINE`—indicates that the server is shut down or unavailable. • `ONLINE_LOCK_AUTHORITY_ONLY`—indicates that the server was rebooted and is currently being updated (from other replicas) with the current call state data. A recovering server cannot yet process call state transactions, because it does not maintain a full copy of the call state managed by the partition.

Table 3-1 ReplicaRuntimeMBean Method and Attribute Summary

Method/Attribute	Description
`TimerQueueSize`	The current number of timers queued on the data tier server. This generally corresponds to the KeyCount value, but may be less if new call states are being added but their associated timers have not yet been queued. **Note:** Engine tier servers periodically check with data tier instances to determine if timers associated with a call have expired. In order for SIP timers to function properly, all engine tier servers must actively synchronize their system clocks to a common time source. BEA recommends using a Network Time Protocol (NTP) client or daemon on each engine tier instance and synchronizing to a selected NTP server.
`TotalBytes`	The total number of bytes consumed by the call state managed in this server.

CHAPTER 4

Configuring Engine Tier Container Properties

The following sections describe how to configure SIP Container features in the engine tier of a WebLogic SIP Server deployment:

Overview of SIP Container Configuration

As described in "WebLogic SIP Server Configuration Overview" on page 2-2, WebLogic SIP Server engine and data tier features are implemented using the `sipserver` J2EE application, and SIP Container configuration is managed by the `config` Web Application contained in `sipserver`, which contains the `sipserver.xml` file.

You can configure SIP Container properties either by using a JMX utility such as the Administration Console or WebLogic Scripting Tool (WLST), or by programming a custom JMX application. "Using the Administration Console to Configure Container Properties" on page 4-2 describes how to configure container properties using the Administration Console graphical user interface.

"Configuring Container Properties Using WLST (JMX)" on page 4-4 describes how to directly access JMX MBeans to modify the container configuration. All examples use WLST to illustrate JMX access to the configuration MBeans.

Using the Administration Console to Configure Container Properties

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

1. Use your browser to access the URL http://*address*:7001/console where *address* is the Administration Server's listen address and 7001 is the listen port.
2. Expand the SIP Servers node in the left pane.
3. Select the sipserver entry to display configuration and monitoring tabs in the right pane of the console.

   Note: In most cases your configuration will have only a single sipserver container beneath the SIP Servers node. Additional containers may be available when performing a production upgrade, as described in "Upgrading Software and Converged Applications" on page B-1.

   The right pane of the console provides two levels of tabbed pages that are used for configuring and monitoring WebLogic SIP Server. Table 4-1 summarizes the available

pages and provides links to additional information about configuring SIP container properties.

Table 4-1 WebLogic SIP Server Configuration and Monitoring Pages

Page		Function
Configuration->	General	Configure SIP timer values, session timeout duration, and default WebLogic SIP Server behavior (proxy or user agent).
	Proxy	Configure proxy routing URIs and proxy policies.
	Overload Protection	Configure the conditions for enabling and disabling automatic overload controls.
	Message Debug	Enable or disable SIP message logging on a development system.
	SIP Security	Identify trusted hosts for which authentication is not performed.
	Data Tier	View the current configuration of data tier servers.
Monitoring->	General	View runtime information about messages and sessions processed in engine tier servers.
	SIP Applications	View runtime session information for deployed SIP applications.
	Data Tier Information	View runtime information about the current status and the work performed by servers in the data tier.

Locking and Persisting the Configuration

In order to modify information on any of the WebLogic SIP Server configuration pages, you must first obtain a lock on the configuration by clicking the Edit Configuration button. Locking a configuration prevents other Administrators from modifying the configuration at the same time. If you click Edit Configuration while another user has obtained a lock, you are unable to make configuration changes until the lock has been released.

If you obtain a lock on the configuration, you can change SIP Servlet container attribute values on multiple configuration pages as needed. You then have several options depending on whether you want to keep or discard the changes you have made. The available options are displayed as a series of buttons at the bottom of each configuration page:

- Save—Saves your current changes to a temporary configuration file, `sipserver.xml.saved`, in the `config` subdirectory of the `sipserver` application.

- Rollback—Discards your current changes, deleting any temporary configuration files that were written with previous Save operations.
- Activate—Persists all current changes to the `sipserver.xml` file (renaming the temporary `sipserver.xml.saved` files to `sipserver.xml`), and activates the changes.

Note that WebLogic SIP Server automatically saves the original boot configuration in the file `sipserver.xml.booted` in the `sipserver/config` subdirectory. You can use this file to revert to the booted configuration if necessary to discard all configuration changes made since the server was started.

Configuring Container Properties Using WLST (JMX)

Notes: The WebLogic Scripting Tool (WLST) is a utility that you can use to observe or modify JMX MBeans available on a WebLogic Server or WebLogic SIP Server instance. WLST is not distributed with WebLogic SIP Server, but can be downloaded from BEA's dev2dev site at https://submit-codesamples.projects.dev2dev.bea.com/servlets/Scarab?id=CS26. Documentation for WLST is included with the product download, and also at http://e-docs.bea.com/wls/docs90/config_scripting/index.html.

Before using WLST to configure a WebLogic SIP Server domain, you must add to your classpath all JAR files in the `APP-INF/lib` directory of the `sipserver` application. By default these files are located in *`DOMAIN_DIR`*`/sipserver/APP-INF/lib` where *`DOMAIN_DIR`* is the root of your WebLogic SIP Server domain. The libraries are automatically added to your classpath using the `setAdminClientEnv` script, described in "Invoking WLST" on page 4-9.

The `APP-INF/lib` classes are required *in addition to* the WLST JAR files described in the WLST documentation.

JMX configuration of the SIP Servlet container is managed by the `configManagerRuntimeMBean`. `ConfigManagerRuntimeMBean` manages tasks such as:

- Governing access to the active SIP Servlet container configuration
- Writing the current container configuration to the `sipserver.xml` configuration file
- Activating the `config` Web Application to apply changes to the running SIP Servlet container

Although any JMX application can access the SIP container's configuration MBeans, all changes to those MBeans must be coordinated through `ConfigManagerRuntimeMBean`.

ConfigManagerRuntimeMBean Usage and Reference

Table 4-2 describes the methods provided by `ConfigManagerRuntimeMBean`.

Table 4-2 ConfigManagerRuntimeMBean Method Summary

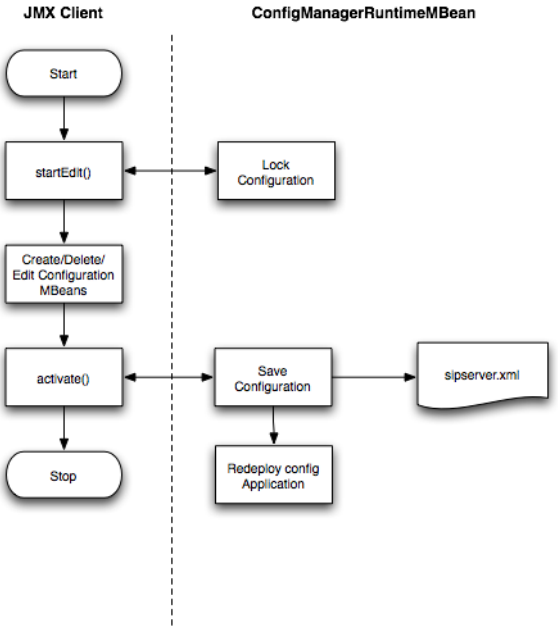
Method	Description
`activate()`	Writes the current configuration MBean attributes (the current SIP Servlet container configuration) to the `sipserver.xml` configuration file and applies changes to the running the `config` application.
`save()`	Writes the current configuration MBean attributes (the current SIP Servlet container configuration) to the `sipserver.xml` configuration file.
`startEdit()`	Locks changes to the active SIP Servlet container configuration. Other JMX application cannot alter the configuration until you explicitly call `stopEdit()`, or until your edit session is terminated. If you attempt to call `startEdit()` when another user has obtained the lock, you receive an error message that states the user who owns the lock.
`stopEdit()`	Releases the lock obtained for modifying SIP container properties and rolls back any pending MBean changes.

A typical configuration session involves the following tasks (also summarized in Figure 4-1):

1. Access the `ConfigManagerRuntimeMBean` for the WebLogic SIP Server instance that you want to configure and call `startEdit()` to obtain a lock on the active configuration.
2. Modify existing SIP Servlet container configuration MBean attributes (or create or delete configuration MBeans) to modify the active configuration. See "Configuration MBeans for the SIP Servlet Container" on page 4-6 for a summary of the configuration MBeans.
3. Call `ConfigManagerRuntimeMBean.save()` to persist all changes to a temporary configuration file named `sipserver.xml.saved`, or
4. Call `ConfigManagerRuntimeMBean.activate()` to persist changes to the `sipserver.xml.saved` file, rename `sipserver.xml.saved` to `sipserver.xml` (copying over the existing file), and apply changes to the running `config` application.

Note: When you boot the Administration Server for a WebLogic SIP Server domain, the server parses the current container configuration in `sipserver.xml` and creates a copy of the initial configuration in a file named `sipserver.xml.booted`. You can use this copy to revert to the booted configuration, as described in "Reverting to the Original Boot Configuration" on page 4-13.

Figure 4-1 Typical ConfigManagerRuntimeMBean Workflow



Configuration MBeans for the SIP Servlet Container

`ConfigManagerRuntimeMBean` manages access to and persists the configuration MBean attributes described in Table 4-3. Although you can modify other configuration MBeans, such as

WebLogic Server MBeans that manage resources such as network channels and other server properties, those MBeans are not managed by `ConfigManagerRuntimeMBean`.

Table 4-3 SIP Container Configuration MBeans

MBean Type	MBean Attributes	Description
`ClusterToLoadBalancerMap`	`ClusterName`, `LoadBalancerSipURI`	Manages the mapping of multiple clusters to internal virtual IP addresses during a software upgrade. This attribute is not used during normal operations. See also cluster-loadbalancer-map in the *Configuration Reference Manual*.
`OverloadProtection`	`RegulationPolicy`, `ThresholdValue`, `ReleaseValue`	Manages overload settings for throttling incoming SIP requests. See also overload in the *Configuration Reference Manual*.
`Proxy`	`ProxyURIs`, `RoutingPolicy`	Manages the URIs routing policies for proxy servers. See also proxy—Setting Up an Outbound Proxy Server in the *Configuration Reference Manual*.

Table 4-3 SIP Container Configuration MBeans

MBean Type	MBean Attributes	Description
SipSecurity	TrustedAuthenticationHosts	Defines trusted hosts for which authentication is not performed. See also sip-security in the *Configuration Reference Manual*.
SipServer	DefaultBehavior, EnableLocalDispatch, MaxApplicationSessionLifeTime, OverloadProtectionMBean, ProxyMBean, T1TimeoutInterval, T2TimeoutInterval, T4TimeoutInterval, TimerBTimeoutInterval, TimerFTimeoutInterval SipServer also has several helper methods: createProxy(), destroyProxy(), createOverloadProtection(), destroyOverloadProtection(), createClusterToLoadBalancerMap(), destroyClusterToLoadBalancerMap()	Configuration MBean that represents the entire sipserver.xml configuration file. You can use this MBean to obtain and manage each of the individual MBeans described in this table, or to set SIP timer or SIP Session timeout values. See also "Creating and Deleting MBeans" on page 4-11, default-behavior, enable-local-dispatch, max-application-session-lifetime, t1-timeout-interval, t2-timeout-interval, t4-timeout-interval, timerB-timeout-interval, and timerF-timeout-interval in the *Configuration Reference Manual*.

Locating the WebLogic SIP Server MBeans

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

mydomain:DomainConfig=*mydomain*,Location=*myserver*,Name=*myserver*,Type=*mbeantype*

where:

- *mydomain* is the name of the WebLogic SIP Server domain
- *myserver* is the name of the WebLogic SIP Server instance
- *mbeantype* corresponds to an MBean type listed in Table 4-3.

Runtime MBeans, such as `ConfigManagerRuntime`, are accessed using the path:

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

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

```
custom()
cd('mydomain:DomainConfig=mydomain,Location=myserver,Name=myserver,Type=Proxy')
ls()
```

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" on page 4-11.

If no entries are present in `sipserver.xml`, only the `SipServer` and `ConfigManagerRuntime` MBean types are available for browsing.

WLST Configuration Examples

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

Invoking WLST

To use WLST with WebLogic SIP Server, you must ensure that all WebLogic SIP Server JAR files are included in your classpath along with the required WLST JAR files. Follow these steps:

1. Set your WebLogic SIP Server client administration environment using a script installed with your domain:

```
cd c:\bea\wlss220\server\bin
.\setAdminClientEnv.cmd
```

2. Add the required WLST JAR files to your class path:

```
cd c:\wlst
set CLASSPATH=%CLASSPATH%;c:\wlst\jython.jar;c:\wlst\wlst.jar
```

3. Start WLST:

```
java weblogic.WLST
```

4. Connect to the Administration Server for your WebLogic SIP Server domain:

```
connect('system','weblogic','t3://myadminserver:7001')
```

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. Listing 4-1 shows a WLST script that contains the common commands needed to access `ConfigManagerRuntimeMBean`. The example script modifies the proxy `RoutingPolicy` attribute, which is set to `supplemental` by default in new WebLogic SIP Server domains. You can use this listing as a basic template, modifying commands to access and modify the configuration MBeans as necessary.

Listing 4-1 Template WLST Script for Accessing ConfigManagerRuntimeMBean

```
# Connect to the Administration Server
connect('weblogic','weblogic','t3://localhost:7001')
# Navigate to ConfigManagerRuntimeMBean and start an edit session.
custom()
cd('mydomain:Location=myserver,Name=sipserver,ServerRuntime=myserver,Type=
ConfigManagerRuntime')
cmo.startEdit()
# --MODIFY THIS SECTION AS NECESSARY--
# Edit SIP Servlet container configuration MBeans
```

```
cd('mydomain:DomainConfig=mydomain,Location=myserver,Name=myserver,SipServ
er=myserver,Type=Proxy')

set('RoutingPolicy','domain')

# Navigate to ConfigManagerRuntimeMBean and persist the configuration

# to sipserver.xml

cd('mydomain:Location=myserver,Name=sipserver,ServerRuntime=myserver,Type=
ConfigManagerRuntime')

cmo.activate()
```

Creating and Deleting MBeans

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

Listing 4-2 shows an example of how to use the helper commands to create and delete configuration MBeans that configuration elements in `sipserver.xml`. See also `Listing 4-3, "SIP Container Configuration MBeans," on page 4-7` for a listing of other helper methods in `SipServer`, or refer to the WebLogic SIP Server JavaDocs.

Listing 4-2 WLST Commands for Creating and Deleting MBeans

```
connect()

custom()

cd('mydomain:Location=myserver,Name=sipserver,ServerRuntime=myserver,Type=
ConfigManagerRuntime')

cmo.startEdit()

cd('mydomain:DomainConfig=mydomain,Location=myserver,Name=sipserver,Server
Runtime=myserver,Type=SipServer')

cmo.destroyOverloadProtection()

cmo.createProxy()
```

```
cd('mydomain:Location=myserver,Name=sipserver,ServerRuntime=myserver,Type=
ConfigManagerRuntime')

cmo.save()
```

Working with URI Values

Configuration MBeans such as `Proxy` require URI objects passed as attribute values. BEA provides a helper class, `com.bea.wcp.sip.util.URIHelper`, to help you easily generate URI objects from an array of Strings. Listing 4-3 modifies the sample shown in Listing 4-2, "WLST Commands for Creating and Deleting MBeans," on page 4-11 to add a new URI attribute to the `LoadBalancer` MBean. See also the WebLogic SIP Server JavaDocs for a full reference to the `URIHelper` class.

Listing 4-3 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=
ConfigManagerRuntime')

cmo.startEdit()

cd('mydomain:DomainConfig=mydomain,Location=myserver,Name=sipserver,Type=S
ipServer')

cmo.createProxy()

cd('mydomain:DomainConfig=mydomain,Location=myserver,Name=sipserver,SipSer
ver=sipserver,Type=Proxy')

stringarg =
jarray.array([java.lang.String("sip://siplb.bea.com:5060")],java.lang.Stri
ng)

uriarg = stringToSipURIs(stringarg)

set('ProxyURIs',uriarg)
```

```
cd('mydomain:Location=myserver,Name=sipserver,ServerRuntime=myserver,Type=
ConfigManagerRuntime')

cmo.save()
```

Reverting to the Original Boot Configuration

When you boot the Administration Server for a WebLogic SIP Server domain, the server creates parses the current container configuration in `sipserver.xml`, and generates a copy of the initial configuration in a file named `sipserver.xml.booted`. This backup copy of the initial configuration is preserved until you next boot the server; modifying the configuration using JMX does not affect the backup copy.

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

Configuring NTP for Accurate SIP Timers

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

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

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

BEA recommends using a Network Time Protocol (NTP) client or daemon on each WebLogic SIP Server instance and synchronizing to a common NTP server.

WARNING: You must accurately synchronize server system clocks to a common time source (to within one or two milliseconds) in order for the SIP protocol stack to function properly. Because the initial T1 timer value of 500 milliseconds controls the retransmission interval for INVITE request and responses, and also sets the initial

values of other timers, even small differences in system clock settings can cause improper SIP protocol behavior. For example, an engine tier server with a system clock 250 milliseconds faster than other servers will process more expired timers than other engine tier servers, will cause retransmits to begin in half the allotted time, and may force messages to timeout prematurely.

CHAPTER 5

Configuring Diameter Sh Client Nodes and Relay Agents

The following sections describe how to configure individual servers to act as Diameter nodes or relays in a WebLogic SIP Server domain:

Overview of Diameter Protocol Configuration

A typical WebLogic SIP Server domain deploys support for the Diameter base protocol and IMS Sh interface provider on all engine tier servers, which each act as Diameter Sh client nodes. SIP Servlets deployed on the engines can use the profile service API to initiate requests for user profile data, or to subscribe to and receive notification of profile data changes. The Sh interface is also used to communicate between multiple IMS Application Servers. Using the Profile Service API (Diameter Sh Interface) in *Programming Applications with WebLogic SIP Server* describes the API in more detail.

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

Note: In order to support multiple HSSs, the 3GPP defines the Dh interface to look up the correct HSS. WebLogic SIP Server 2.2 does not provide a Dh interface application, and can be configured only with a single HSS.

Note that relay agent servers do not function as either engine or 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).

WebLogic SIP Server also provides a simple HSS simulator that you can use for testing Sh client applications. You can configure a WebLogic SIP Server instance to function as an HSS simulator by deploying the appropriate application.

Steps for Configuring Diameter Client Nodes and Relay Agents

To configure Diameter support in a WebLogic SIP Server domain, follow these steps:

1. Install the WebLogic SIP Server Diameter Domain. Install the Diameter domain, which contains a sample configuration and template Web Applications configured for different Diameter node types.
2. Create Diameter network channels. Create the network channels necessary to support Diameter over TCP or TLS transports on engine tier servers and relays.
3. Configure the Diameter Sh client nodes (engine tier servers). Configure the Diameter Sh client application on engine tier servers with the host name, peers, and routes to the relay agents or HSS.
4. Configure Diameter relay agents (optional). Configure the Diameter relay application on one or more server instances to act as relay agents to an HSS.
5. Configure an HSS Simulator (optional). Configure a the Diameter relay application on a WebLogic SIP Server instance to function as an HSS simulator for testing or example purposes.

The sections that follow describe each step in detail. See also the "Example Domain Configuration" on page 5-14.

Installing the Diameter Domain

The Configuration Wizard includes a Diameter domain template that creates a domain having four WebLogic SIP Server instances:

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

You can use the installed Diameter domain as the basis for creating your own domain. Or, you can use the customized Diameter Web Applications as templates for configuring existing WebLogic SIP 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. Follow these steps to install the domain:

1. Change to the *WLSS_HOME*\common directory, where WLSS220 is the directory in which you install WebLogic SIP Server 2.2 (for example, c:\bea\wlss220\common).
2. Execute the config.cmd or config.sh script to launch the Configuration Wizard.
3. Select Create a new WebLogic Configuration and click Next.
4. Select the Diameter WebLogic SIP Server domain and click Next.
5. Select Express and click Next.
6. Select the preconfigured server, "myserver" to function as the Administration Server and click Next.
7. Enter a username and password for the Administrator of the new domain, and click Next.
8. Select the startup mode and JDKs to use with the new domain, and click Next.
9. Click Create to create the new domain using the default domain name and domain location (*BEA_HOME*/user_projects/domains/diameter).
10. Click Done.

Table 5-1 describes the server configuration installed with the Diameter domain.

Table 5-1 Key Configuration Elements of the Diameter Domain

Server Name	Network Channel Configuration	Deployed Applications	Notes
`myserver`	n/a	n/a	The Administration Server uses no SIP or Diameter network channels and deploys no `sipserver` or Diameter application.
`hssclient`	diameterchannel (UDP/TCP over port 3868) sipchannel (UDP/TCP over port 5060)	`./diameter_hssclient` `./sipserver`	The `hssclient` server shows the sample configuration of an engine tier server that functions as a Diameter Sh client node. The server contains network channels supporting both SIP and Diameter traffic. The `diameter_hssclient` Web Application configures the server as an Sh client node, and the `sipserver` implementation application provides SIP container services.
`relay`	diameterchannel (TCP over port 3869)	`./diameter_relay`	The `relay` server shows the sample configuration of a server that 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_relay` Web Application configures the Diameter relay application to connect the `hssclient` and `hss` servers. The `sipserver` EAR is not deployed to the relay as it provides no SIP container functionality.
`hss`	diameterchannel (TCP over port 3870)	`./diameter_hss`	The `hss` server deploys only the HSS simulator application code, and is configured with a Diameter network channel.

Creating Network Channels for the Diameter Protocol

WebLogic SIP Server's Diameter implementation supports the Diameter protocol over two different transport protocols: TCP and TLS. To enable incoming Diameter connections on a server, you must configure a dedicated network channel using the protocol type "diameter" or "diameters" for TCP and TCP/TLS transport, respectively. The provider may automatically upgrade Diameter connections to use TLS as is described in the Diameter specification (RFC 3558).

Servers that use a TCP/TLS channel for Diameter (diameters channels) must also enable two-way SSL.

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

1. Access the Administration Console for the WebLogic SIP Server domain.
2. In the left pane, select the name of the server to configure.
3. In the right pane, select Protocols->Channels to display the configured channels.
4. Click Configure a new Network Channel in the right pane.
5. Fill in the new channel fields as follows:
   - **Name:** Enter an administrative name for this channel, such as "Diameter TCP/TLS Channel."
   - **Protocol:** Select either diameter to support the TCP transport, or diameters to support both TCP and TLS transports.

     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).
   - **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.
6. Click Create to create the new channel.
7. Display the advanced configuration items for the newly-created channel by clicking the Show link next to Advanced Options.

8. Change the **Idle Connection Timeout** value from the default (65 seconds) to a larger value that will ensure the Diameter connection remains consistently available.

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

9. Click Apply.

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:

1. Configure basic SSL support, if you have not already done so. See Configuring SSL in the WebLogic Server 8.1 Documentation.
2. Access the Administration Console for the WebLogic SIP Server domain.
3. In the left pane, select the name of the server to configure.
4. Select the Configuration-->Keystores & SSL tab.
5. Click the Show link under Advanced Options.
6. Go to the Server attributes section of the window.
7. Change the Two Way Client Cert Behavior attribute from the default, Client Certs Not Requested (which defines one-way SSL) to one of the remaining options:
   - **Client Certs Requested But Not Enforced**—Requires a client to present a certificate. If a certificate is not presented, the SSL connection continues.
   - **Client Certs Requested And Enforced**—Requires a client to present a certificate. If a certificate is not presented or if the certificate is not trusted, the SSL connection is terminated.
8. Click Apply.
9. Reboot the server.

Configuring Diameter Sh Client Nodes

All engine tier servers that act as Diameter client nodes share the same configuration for the Diameter Sh client application. (Servers that act as Diameter relay agents configure deploy a different Diameter Web Application.) The Sh client application is deployed as a standalone Web Application, and is configured using a `diameter.xml` configuration file. To enable Sh client support for engine server nodes, you copy the `diameter_hssclient` Web Application from the Diameter domain you installed using the configuration wizard, and then modify the configuration files to suit your environment. Follow these steps:

1. Copy the `diameter_hssclient` directory contents from the Diameter domain you installed into your own domain. For example:

   ```
   cp -r ~/bea/user_projects/domains/diameter/diameter_hssclient
   ~/bea/user_projects/domains/replicated
   ```

2. Open the `diameter.xml` configuration file with a text editor:

   ```
   emacs
   ~/bea/user_projects/domains/replicated/diameter_hssclient/WEB-INF/confi
   g/diameter.xml
   ```

3. Edit the configuration file to change the following items to match those for your domain:

   – The host names of any relay agents configured in the domain, supplied as arguments to the Sh application and 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.

   The template `diameter.xml` file contains sample values for these items. Listing 5-1 below also shows a sample configuration file for an engine tier cluster that accesses an HSS using two relays, with highlighted comments for certain entries. See Using the IMS Sh Interface (Diameter) in *Programming Applications with WebLogic SIP Server* for more information about elements in the `diameter.xml` file.

Listing 5-1 diameter.xml Configuration for Sh Client Nodes Using Relay Agents

```
<?xml version="1.0" encoding="utf-8"?>
<configuration xmlns:xsi= "http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="http://www.bea.com/ns/wlcp/diameter.xsd">
```

```
    <!-- Omit the host and realm elements to dynamically assign the hostname
         and domain name of individual engine tier servers. -->
  <node>
    <applications>
      <application>
        <auth-application-id>16777217</auth-application-id>
        <vendor-id>10415</vendor-id>
        <!-- Specify the BEA Sh application ID and Diameter vendor ID. -->
        <class-name>com.bea.wcp.diameter.sh.WlssShApplication</class-name>
        <!-- Identify the BEA Sh application provider class name. -->
        <param>
          <name>dest.host</name>
          <value>HSS_hostname</value>
        </param>
        <!-- Include a dest.host param definition only if servers will
             communicate directly to an HSS (static routing), without using
             a relay agent. Omit the dest.host param completely when routing
             through relay agents. -->
        <param>
          <name>dest.realm</name>
          <value>relayorhss.com</value>
        </param>
        <!-- Specify the realm name of relay agent servers or the HSS,
             depending on whether or not the domain uses relay agents. -->
      </application>
    </applications>
    <peers>
```

```
    <peer>
      <host>relay1</host>
      <address>relay1host</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>3821</port>
    </peer>
    <peer>
      <host>relay2</host>
      <address>relay2host</address>
      <port>3821</port>
    </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. -->
</peers>
  <routes>
    <default-route>
      <action>relay</action>
      <server>relay1</server>
    </default-route>
    <!-- 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. -->
        <route>
          <action>relay</action>
          <server>relay2</server>
        </route>
        <!-- Include additional route entries for each relay agent in the
            domain. -->
      </routes>
    </node>
    </configuration>
```

4. Save your changes to the `diameter.xml` file and exit the text editor.
5. Start the Administration Server for your domain, and deploy the `diameter_hssclient` application directory to the engine tier cluster, as in:

```
java weblogic.Deployer -adminurl t3://localhost:7001 -user weblogic \
   -password weblogic -targets BEA_ENGINE_TIER_CLUST -deploy \
   -source ~/bea/user_projects/domains/replicated/diameter_hssclient
```

Configuring Diameter Relay Agents (Optional)

Each server instance that acts as a Diameter relay agent must be configured independently (have a dedicated `diameter.xml` configuration file packaged in a copy of the Diameter Web Application). Relay agents are not required in a Diameter configuration, but BEA 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: In addition to deploying a custom `diameter.xml` configuration to each relay agent, you must ensure that relay servers *do not* also act as WebLogic SIP Server engine tier servers or data tier servers. This means that the servers should not be configured with “sip” or “sips” network channels, and should not deploy the `sipserver` Enterprise Application.

Relay agent nodes route Sh messages between client nodes and the HSS, but they do not modify the messages except as defined in the Diameter Sh specification. Relays always route responses

from the HSS back the client node that initiated the message, or the message the response is dropped if that node is unavailable.

For each relay agent node that you want to configure, follow these steps:

1. Copy the `diameter_relay` directory contents from the Diameter domain you installed into a new directory in your own domain:

   ```
   cp -r ~/bea/user_projects/domains/diameter/diameter_relay
   ~/bea/user_projects/domains/replicated/diameter_relay1
   ```

   Note that you will need to create and configure a separate "`diameter_relay`" application directory for each relay in your system.

2. Open the `diameter_relay` application's `diameter.xml` configuration file with a text editor. For example:

   ```
   emacs
   ~/bea/user_projects/domains/replicated/diameter_relay1/WEB-INF/config/d
   iameter.xml
   ```

3. Edit the configuration file to change the following items to match those for your domain:
   - The host and realm name of the relay agent server.
   - The host and port names of all servers acting as Diameter Sh client nodes, defined as Diameter peer nodes.
   - A default route to the HSS configured in the network.

   The template `diameter.xml` file contains sample values for these items. Listing 5-2 below also shows a sample configuration file for a single relay node in the example two-relay system described in "Example Domain Configuration" on page 5-14.

Listing 5-2 diameter.xml Configuration for a Diameter Relay Agent Server

```
<?xml version="1.0" encoding="utf-8"?>
<configuration xmlns:xsi= "http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="http://www.bea.com/ns/wlcp/diameter.xsd">
  <host>relay1</host>
  <realm>bea.com</realm>
  <!-- Specify the host and realm name for this relay agent. -->
<node>
```

```
<applications>
  <application>
    <auth-application-id>-1</auth-application-id>
    <class-name>com.bea.wcp.diameter.relay.RelayApplication</class-name>
    <!-- Identify the BEA relay application class name. -->
  </application>
</applications>
<peers>
  <!-- Define peer connection information for each Sh client, or use the
       allow-dynamic-peers functionality in combination with TLS
       transport to allow peers to be recognized automatically. -->
  <peer>
    <host>engine1</host>
    <address>engine1host</address>
    <port>3821</port>
  </peer>
  <peer>
    <host>engine2</host>
    <address>engine2host</address>
    <port>3821</port>
  </peer>
  <peer>
    <host>engine3</host>
    <address>engine3host</address>
    <port>3821</port>
  </peer>
  <peer>
```

```
      <host>relay2</host>
      <address>relay2host</address>
      <port>3821</port>
    </peer>
    <peer>
      <host>hss</host>
      <address>hsshost</address>
      <port>3821</port>
    </peer>
    <!-- Include peer entries for each engine tier server (each Diameter
         client node), additional relay agents, and for the HSS itself. -->
  </peers>
  <routes>
    <default-route>
      <action>relay</action>
      <server>hss</server>
    </default-route>
    <!-- Enter a default route for this agent to relay messages
         to the HSS. -->
  </routes>
</node>
</configuration>
```

4. Save your changes to the `diameter.xml` file and exit the text editor.
5. Start the Administration Server for your domain, and deploy the correct "`diameter_relay`" application directory to the corresponding WebLogic SIP Server instance. Use either the Administration Console or the `weblogic.Deployer` utility, as in:

```
java weblogic.Deployer -adminurl t3://localhost:7001 -user weblogic \
   -password weblogic -deploy -targets Relay1 \
```

```
-source ~/bea/user_projects/domains/replicated/diameter_relay1
```

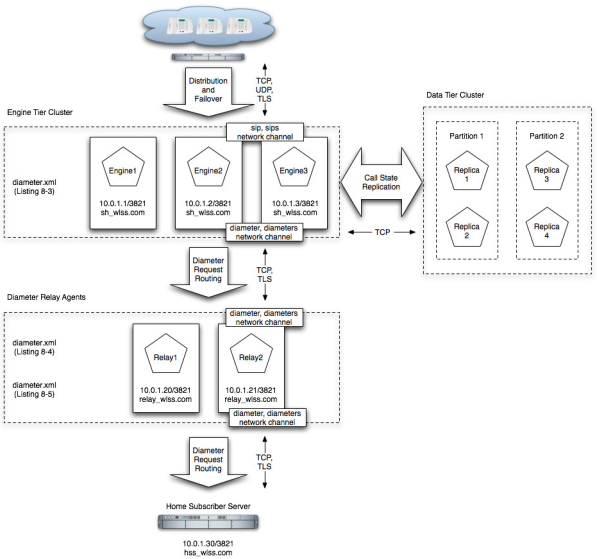
Example Domain Configuration

This section describes a sample WebLogic SIP Server configuration that provides basic Diameter Sh protocol capabilities. The layout of the sample domain includes the following:

- Three engine tier servers which host SIP applications and also deploy the Diameter Sh application for accessing user profiles.
- Four data tier servers arranged into two partitions with two replicas each.
- Two servers that act as Diameter relay agents and forward diameter requests to an HSS.

Figure 5-1 shows the individual servers in the sample configuration.

Figure 5-1 Sample Diameter Domain



The listings that follow show the contents of `diameter.xml` files used in to configure the sample domain.

Listing 5-3 diameter.xml Configuration for Sample Engine Tier Cluster

```
<?xml version="1.0" encoding="utf-8"?>
<configuration xmlns:xsi= "http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="http://www.bea.com/ns/wlcp/diameter.xsd">
<node>
  <applications>
    <application>
      <auth-application-id>16777217</auth-application-id>
      <vendor-id>10415</vendor-id>
      <class-name>com.bea.wcp.diameter.sh.WlssShApplication</class-name>
      <param>
        <name>dest.realm</name>
        <value>relay_wlss.com</value>
      </param>
    </application>
  </applications>
  <peers>
    <peer>
      <host>Relay1</host>
      <address>10.0.1.20</address>
      <port>3821</port>
    </peer>
    <peer>
      <host>Relay2</host>
      <address>10.0.1.21</address>
      <port>3821</port>
    </peer>
```

```
  </peers>
  <routes>
    <default-route>
      <action>relay</action>
      <server>Relay1</server>
    </default-route>
    <route>
      <action>relay</action>
      <server>Relay2</server>
    </route>
  </routes>
</node>
</configuration>
```

Listing 5-4 diameter.xml Configuration for Sample Relay Agent Server 1

```
<?xml version="1.0" encoding="utf-8"?>
<configuration xmlns:xsi= "http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="http://www.bea.com/ns/wlcp/diameter.xsd">
<node>
  <host>Relay1</host>
  <realm>bea.com</realm>
  <applications>
    <application>
      <auth-application-id>-1</auth-application-id>
      <class-name>com.bea.wcp.diameter.relay.RelayApplication</class-name>
    </application>
  </applications>
```

```
<peers>
  <peer>
    <host>Engine1</host>
    <address>10.0.1.1</address>
    <port>3821</port>
  </peer>
  <peer>
    <host>Engine2</host>
    <address>10.0.1.2</address>
    <port>3821</port>
  </peer>
  <peer>
    <host>Engine3</host>
    <address>10.0.1.3</address>
    <port>3821</port>
  </peer>
  <peer>
    <host>Relay2</host>
    <address>10.0.1.21</address>
    <port>3821</port>
  </peer>
  <peer>
    <host>HSS</host>
    <address>10.0.1.30</address>
    <port>3821</port>
  </peer>
</peers>
```

```
  <routes>
    <default-route>
      <action>relay</action>
      <server>HSS</server>
    </default-route>
  </routes>
</node>
</configuration>
```

Listing 5-5 diameter.xml Configuration for Sample Relay Agent Server 2

```
<?xml version="1.0" encoding="utf-8"?>
<configuration xmlns:xsi= "http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="http://www.bea.com/ns/wlcp/diameter.xsd">
<node>
  <host>Relay2</host>
  <realm>bea.com</realm>
  <applications>
    <application>
      <auth-application-id>-1</auth-application-id>
      <class-name>com.bea.wcp.diameter.relay.RelayApplication</class-name>
    </application>
  </applications>
  <peers>
    <peer>
      <host>Engine1</host>
      <address>10.0.1.1</address>
      <port>3821</port>
```

```
    </peer>
    <peer>
      <host>Engine2</host>
      <address>10.0.1.2</address>
      <port>3821</port>
    </peer>
    <peer>
      <host>Engine3</host>
      <address>10.0.1.3</address>
      <port>3821</port>
    </peer>
    <peer>
      <host>Relay1</host>
      <address>10.0.1.20</address>
      <port>3821</port>
    </peer>
    <peer>
      <host>HSS</host>
      <address>10.0.1.30</address>
      <port>3821</port>
    </peer>
  </peers>
  <routes>
    <default-route>
      <action>relay</action>
      <server>HSS</server>
    </default-route>
```

```
    </routes>
</node>
</configuration>
```

Configuring an HSS Simulator

The Diameter domain template installs a `diameter_hss` Web Application that you can deploy to a server instance in order to simulate an HSS in your domain. This is provided for testing or development purposes only, and is not meant as a substitute for a production HSS.

To set up a WebLogic SIP Server instance with the HSS simulator application:

1. Copy the `diameter_hss` directory contents from the Diameter domain you installed into a new directory in your own domain:

   ```
   cp -r ~/bea/user_projects/domains/diameter/diameter_hss
   ~/bea/user_projects/domains/replicated/
   ```

2. Open the `diameter_hss` application's `diameter.xml` configuration file with a text editor. For example:

   ```
   emacs
   ~/bea/user_projects/domains/replicated/diameter_hss/WEB-INF/config/diam
   eter.xml
   ```

3. Edit the configuration file to change the following items to match those for your domain:
   - The host and realm name of the HSS.
   - The host and port names of all servers acting as Diameter relay nodes (or as Diameter Sh client nodes if your system uses no relays), defined as Diameter peer nodes.
   - A default route to relay to the HSS server itself.

   The template `diameter.xml` file contains sample values for a domain having a single relay. Listing 5-6 below shows the contents of this file.

Listing 5-6 diameter.xml Configuration for an HSS Simulator

```
<?xml version="1.0" encoding="utf-8"?>
<configuration xmlns:xsi= "http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="http://www.bea.com/ns/wlcp/diameter.xsd">
```

```
<node>
  <host>hss</host>
  <realm>hss.com</realm>
  <applications>
    <application>
      <auth-application-id>16777217</auth-application-id>
      <vendor-id>10415</vendor-id>
      <class-name>com.bea.wcp.diameter.sh.HssSimulator</class-name>
    </application>
  </applications>
  <peers>
    <peer>
      <host>relay</host>
      <address>localhost</address>
      <port>3869</port>
    </peer>
  </peers>
 <routes>
   <default-route>
     <action>relay</action>
     <server>relay</server>
   </default-route>
 </routes>
</node>
</configuration>
```

CHAPTER 6

Capacity Planning for WebLogic SIP Server Deployments

The following sections describe how to configure WebLogic SIP Server domains to support the SIP traffic and features required in your organization:

Introduction to Capacity Planning

BEA WebLogic SIP Server runs on a wide variety of hardware, and provides a highly scalable architecture that can be deployed on multiple machines. Capacity planning is the process of determining the hardware configuration that is required to meet your organization's performance and reliability goals. This document provides capacity planning suggestions for WebLogic SIP Server with a focus on server hardware requirements.

Notes: Capacity planning is not an exact science. Every application is different, particularly converged HTTP and SIP applications. This document is intended as a general guide for

planning server deployments; it uses conservative estimates for basic hardware components and deployed applications. You should err on the side of caution when planning for applications or hardware that differ from the basic recommendations described in this document.

Any and all recommendations generated by this guide should be verified by actual benchmarks before placing a system into production.

Your BEA sales or professional services representative may have more detailed capacity planning information than is available in this document.

WebLogic SIP Server is implemented using a flexible architecture in which you can easily add servers as needed to increase throughput for SIP traffic, or to provide high reliability for defending against hardware failures. The primary goal of capacity planning for WebLogic SIP server is to determine the size and configuration of the WebLogic SIP Server engine tier, which hosts application logic, and the data tier, which stores the call state for SIP messages. Both tiers may be extended to improve throughput or to increase the availability of services on your network.

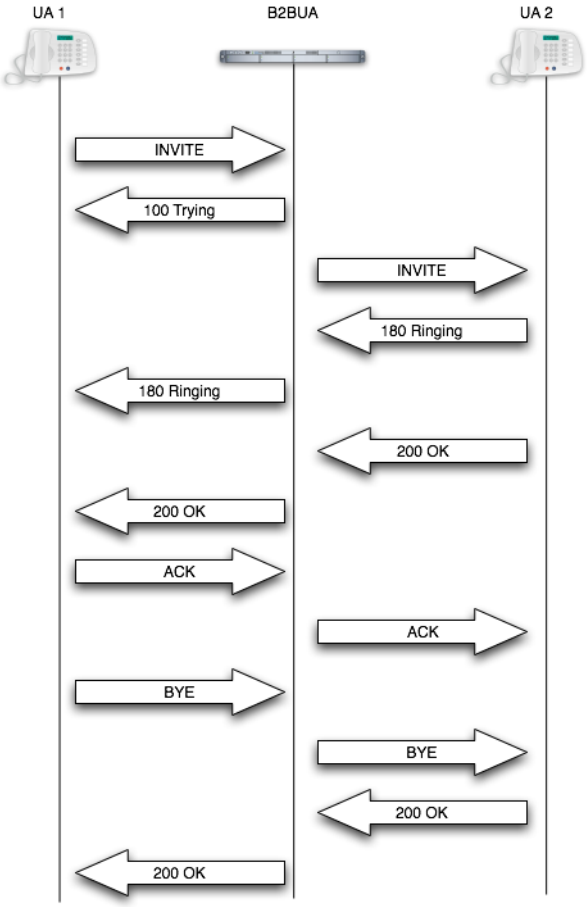
See "Overview of the WebLogic SIP Server Architecture" on page 1-1 for more information about the basic architecture of a WebLogic SIP Server deployment.

Determining Performance Goals

Accurate capacity planning begins with a clear understanding of the throughput and reliability requirements for your deployed applications. The following questions will help drive the capacity planning process for WebLogic SIP Server:

- **What is the required call volume?** The call volume, expressed in calls per hour, is a primary numerical input for determining the hardware requirements of a WebLogic SIP Server deployment.
- **How many SIP messages do my applications generate?** Basic throughput estimates for WebLogic SIP Server are based on a Back-to-Back User Agent (B2BUA) Servlet processing and originating 13 SIP messages as shown in the call flow in Figure 6-1. The number of messages per call, combined with the call volume, determines the total number of SIP messages that the system must process in a given period of time. This value drives hardware requirements for both the engine and data tiers.

Figure 6-1 Call Flow for Capacity Planning B2BUA



If you deploy a B2BUA that generates more SIP messages for each call, each WebLogic SIP Server instance will support fewer calls per second. If you deploy an application that generates fewer SIP messages per call, or if your system provides mainly proxy services, then each server can support additional call volume.

- **What is the average call duration?** This guide uses an estimated average call duration of 6 minutes. When combined with the call volume estimate, the call duration helps you determine the total number of concurrent calls your deployment must manage at a given

time. Longer call durations require additional RAM and possibly additional server hardware in the data tier to handle the increase in concurrent call state.

- **What is the size of the call state?** This guide uses a conservative estimate of 30K per call to manage the call state. If your application stores a larger amount of data for each call, additional RAM may be required in the data tier for maintaining state information.
- **What level of redundancy is required for keeping call state available?** WebLogic SIP Server enables you to replicate call state data on multiple machines to provide high availability in the event of a hardware failure. If a host in the data tier fails, the call state data associated with the failed server can be immediately retrieved from another SIP Server instance in the data tier. You can choose to maintain up to 3 backup copies of the call state data as necessary to provide high availability for your deployment. Each backup copy requires additional RAM and generally additional host hardware in the data tier to manage the replicated data between servers.

Note: The deployment scenarios described in this document use estimates for call duration, call state size, and number of SIP messages per call for a B2BUA application. BEA has derived these estimates from working with organizations that are deploying WebLogic SIP Server for production use. The derived numbers are generally conservative and provide workable hardware estimates for many production environments. However, if your system or application differs significantly from the estimated numbers, you must perform additional profiling to determine the exact hardware configuration required.

Note: WebLogic SIP 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 "Tuning JVM Garbage Collection for Production Deployments" on page F-1 for suggestions about maximizing JVM performance in a production domain.

Basic Hardware Configuration and Throughput Values

The capacity planning scenarios described in this document use one or more basic machine configurations consisting of:

- Dual Xeon 3.6Ghz Processors
- 4 GB of RAM
- Gigabit Ethernet (dual NICs required for redundant connection to data tier).

Each machine should host either a single WebLogic SIP Server engine tier server instance (which uses both processors) or two WebLogic SIP Server data tier server instances (with one processor per instance).

Because the engine tier relies on data tier servers in order to retrieve call state data, BEA recommends using dual Network Interface Cards (NICs) on engine and data tier machines to provide redundant network connections. Gigabit Ethernet is required for the high throughput achieved in most production deployments. See "Throughput Values for WebLogic SIP Server Instances" on page 6-5 for more detailed throughput calculations.

The JVM for each engine tier server should use the minimum amount of heap space required for your deployed applications. This document assumes roughly 700 megabytes per engine tier server instance. Data tier servers should use the maximum possible heap space for the Java Virtual Machine. This document use a conservative value of 1.6 gigabytes per JVM.

If your organization uses substantially different machine specifications, you will need to profile the hardware to determine the exact throughput capabilities for each machine.

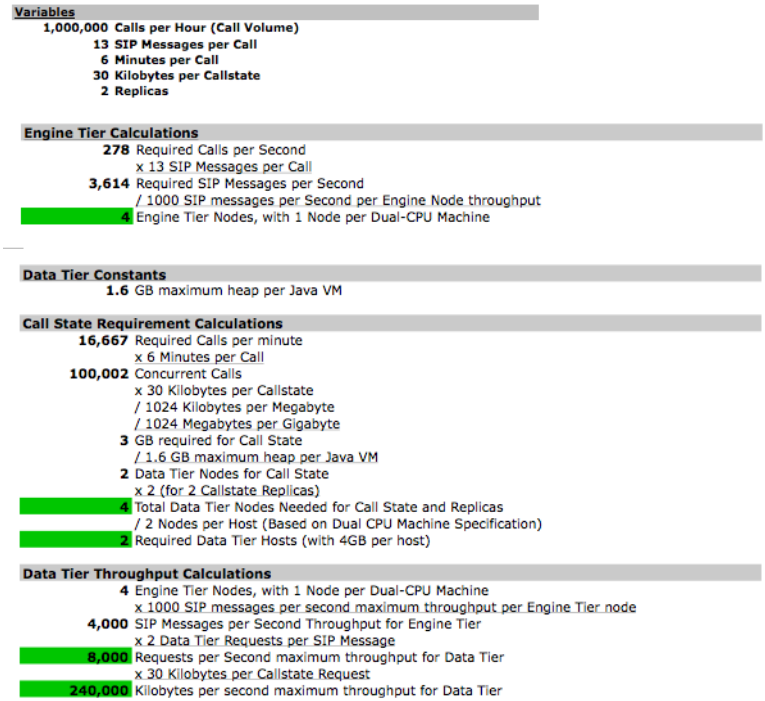
Throughput Values for WebLogic SIP Server Instances

With the processing power provided by the basic hardware configuration described above, each WebLogic SIP Server instance in the engine tier cluster can process and originate approximately 1,000 SIP messages per second, or 76 calls per second per second (assuming 13 SIP messages per call for a single B2BUA Servlet as shown in Figure 6-1). This basic throughput value is used to drive the hardware requirements for the WebLogic SIP Server engine tier.

The size of the WebLogic SIP Server data tier is driven by the expected total number of simultaneous calls managed by the deployment. Given the total number of concurrent calls and the average size of the call state, you can determine the maximum amount of RAM required to store concurrent calls state for your system. This value is multiplied according to the number of redundant call state replicas you wish to deploy for high availability. The total RAM requirement is then divided by the maximum heap size per JVM to determine the number of data tier nodes, and ultimately the number of host machines required in the WebLogic SIP Server data tier.

Figure 6-2, "Using Throughput Values to Determine Engine and Data Tier Requirements," on page 6-6 shows the sample calculations used to determine the medium-sized deployment described in "Medium Deployment" on page 6-8.

Figure 6-2 Using Throughput Values to Determine Engine and Data Tier Requirements



Sample Deployment Scenarios

Given the basic hardware configuration and expected throughput values described in Sample Deployment Scenarios, you can quickly estimate the total number of clustered WebLogic SIP Server instances that are required in the engine and data tiers, as well as the total number of hosts and RAM required in each tier. The following sections describe common deployment sizes used in production environments:

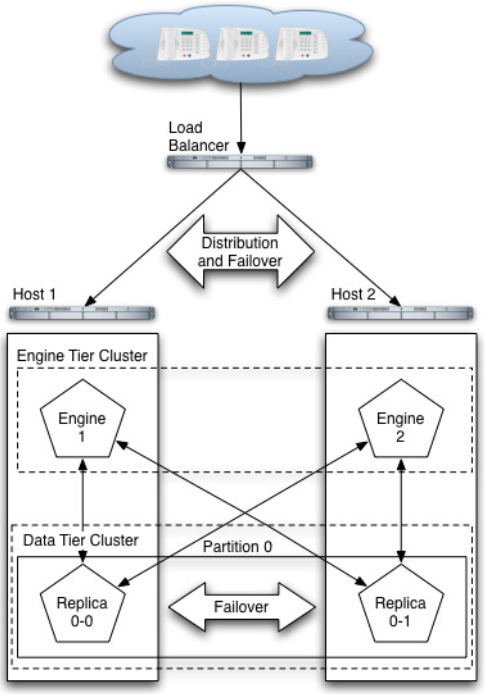
- Small Deployment describes a simple system that supports 500,000 calls per hour.
- Medium Deployment describes an average call system that utilizes multiple clustered servers to support 1 million calls per hour, with data replication to ensure high availability.
- Large Deployment describes a high performance, highly-available system that supports 10 million calls per hour with two replicas of the call state.

Small Deployment

A typical small deployment of WebLogic SIP Server consists of two dual-CPU machines each running two WebLogic SIP Server instances. A system of this size can process over 500,000 calls per hour, given the hardware and throughput values described in "Basic Hardware Configuration and Throughput Values" on page 6-4. The small deployment utilizes only a single partition in the data tier, but with two replicas to provide failover if one of the engines should fail. A load balancer distributes client connections and performs failover from one engine tier server to the other if an engine tier node fails or is brought down for maintenance. This configuration is shown in Figure 6-3.

Note: The exact configuration shown in Figure 6-3 should only be used in cases where limited hardware is available. Although limited failover is provided via two replicas in the data tier, the overall throughput of the system is greatly reduced should one of the machines fail.

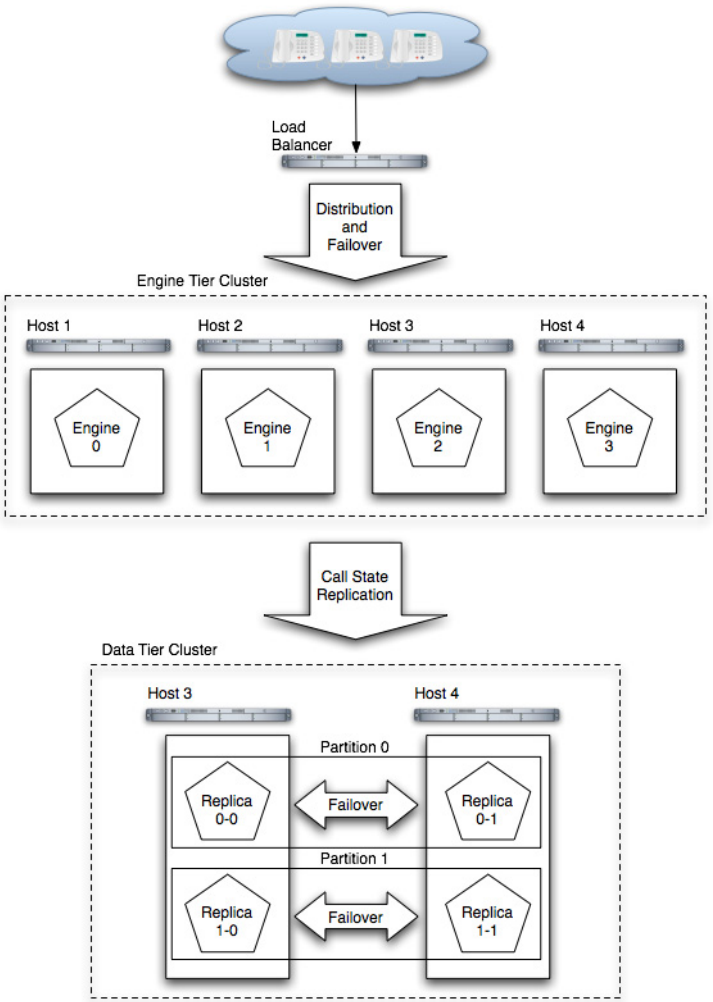
Figure 6-3 Small Deployment with High Availability



Medium Deployment

A typical medium deployment, shown in Figure 6-4, is configured to support a call rate of one million calls per hour. In the engine tier, four WebLogic SIP Server instances (deployed on four Dual-CPU machines) are required to support the call throughput. In the data tier, two partitions are required to manage the call state for the maximum number of expected concurrent calls. Two replicas in each partition provide replication and failover in the event of a data tier host failure.

Figure 6-4 Medium-Sized Deployment



Large Deployment

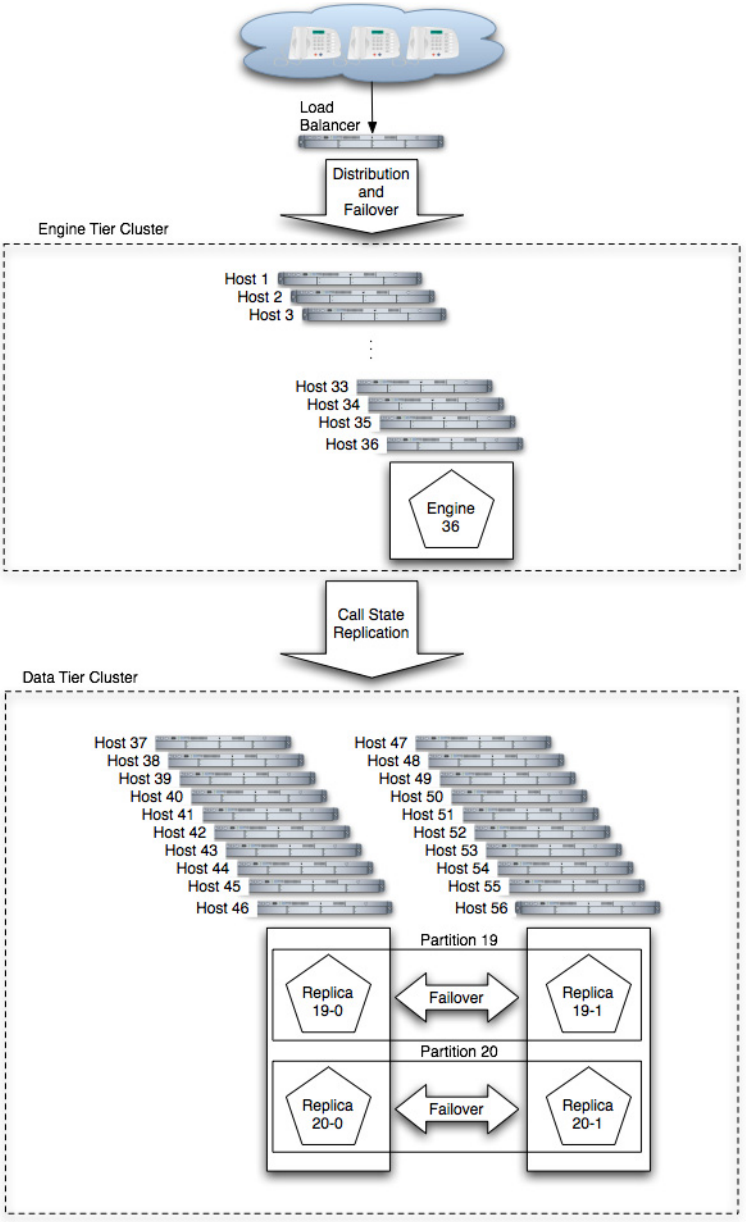
In the sample large-scale deployment, both the engine tier and data tier clusters have been expanded to support a call rate of 10 million calls per hour, as shown in Figure 6-5, “Large-Scale Deployment,” on page 6-11.

In the engine tier, 36 nodes are required given the maximum throughput per node value of 1,000 SIP messages per second.

In the data tier, 20 servers are required to manage the call state for the estimated number of concurrent calls. However, to provide redundancy in the event of a failure, two replicas in each partition store copies of the partition's call state, resulting in 40 server nodes in the data tier.

To maximize the reliability for such a large deployment, each group of servers in the data tier should be located in different physical locations, and/or on separate, dedicated networks.

Figure 6-5 Large-Scale Deployment



CHAPTER 7

Managing WebLogic SIP Server Network Resources

The following sections describe how to configure network resources for use with WebLogic SIP Server:

Overview of Network Configuration

The default HTTP network configuration for each WebLogic SIP Server instance is determined from the Listen Address and Listen Port setting for each server. However, WebLogic SIP 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 WebLogic SIP Server instance. A network channel is a configurable 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 (load balancer address) to embed in SIP headers when the channel is used for an outbound connection.

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

When you configure a new network channel for the SIP protocol, WebLogic SIP Server automatically creates the necessary both the UDP and TCP transport protocols on the configured port. You cannot create a SIP channel that uses only UDP transport or only TCP transport. When you configure a network channel for the SIPS protocol, the server uses the TLS transport protocol for the connection.

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

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

Configuring Load Balancer Addresses

If your system uses one or more load balancers to distribute connections to the engine tier, you must configure SIP network channels to include a load balancer address as the *external listen address*. When a SIP channel has an external listen address that differs from the channel's primary listen address, WebLogic SIP Server embeds the host and port number of the external address in SIP headers such as `Response`. In this way, subsequent communication for the call is directed to the public load balancer address, rather than the local engine tier server address (which may not be accessible to external clients).

If a network channel does not have a configured external listen address, the primary listen address is embedded into SIP headers.

Multiple Load Balancers and Multihomed Load Balancers

If your system uses two load balancers, you must define two channels on each engine tier server (one for each network connection to each load balancer) and assign the external listen address to the corresponding load balancer. When a particular network interface on the engine tier server is selected for outbound traffic, the network channel associated with that NIC's address is examined to determine the external listen address to embed in SIP headers.

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

Configuring Network Channels for SIP or SIPS

When you create a new domain using the Configuration Wizard, WebLogic SIP 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" on page 7-4 for to create a new channel resource to support additional protocols or additional network interfaces.

Reconfiguring an Existing Channel

Note: You cannot change the protocol supported by an existing channel. To reconfigure an existing listen address/port combination to use a different network protocol, you must delete the existing channel and create a new channel using the instructions in "Creating a New SIP or SIPS Channel" on page 7-4.

1. Access the Administration Console for the WebLogic SIP Server domain.
2. In the left pane, select the name of the server to configure.
3. In the right pane, select Protocols->Channels to display the configured channels.
4. To delete an existing channel, click the trash can icon next the channel name.
5. To reconfigure an existing channel:
   a. Select the channel's name from the channel list (for example, the default `sipchannel`).

b. Edit the Listen Address or Listen Port fields to correspond to the address of a NIC or logical address on the associated engine tier machine.

c. Edit the External Listen Address or External Listen Port fields to match the public address of a load balancer in the system.

d. Edit the advanced channel attributes as necessary (see "Creating a New SIP or SIPS Channel" on page 7-4 for details.)

e. Click Apply to apply your changes.

Creating a New SIP or SIPS Channel

To create add a new SIP or SIPS channel to the configuration of a WebLogic SIP Server instance:

1. Access the Administration Console for the WebLogic SIP Server domain.
2. In the left pane, select the name of the server to configure.
3. In the right pane, select Protocols->Channels to display the configured channels.
4. Click Configure a new Network Channel in the right pane.
5. Fill in the new channel fields as follows:
   - **Name:** Enter an administrative name for this channel, such as "SIPS-Channel-eth0."
   - **Protocol:** Select either SIP to support UDP and TCP transport, or SIPS to support TLS transport. Note that a SIP channel cannot support only UDP or only TCP transport on the configured port.
   - **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. 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.
6. Click Create to create the new channel.
7. Edit the **External Listen Address** and **External Listen Port** fields to match the public address of a load balancer associated with this channel. When the server selects an interface or logical address to use for outbound network traffic, WebLogic SIP Server examines the channel that was configured with the same primary **Listen Address**; if the **External Listen**

Address of this channel differs, the external address is embedded into SIP message headers for further call traffic. See "Configuring Load Balancer Addresses" on page 7-2.

8. Optionally click Show to display and edit advanced channel properties, such as connection timeout values. Keep in mind the following restrictions and suggestions for advanced channel properties:
   - **Outbound Enabled**—This attribute cannot be unchecked, because all SIP and SIPS channels can originate network connections.
   - **HTTP Enabled for This Protocol**—This attribute cannot be selected for SIP and SIPS channels, because WebLogic SIP Server does not support HTTP transport SIP protocols.
   - **Maximum Message Size**—This attribute specifies the maximum TCP message size that the server allows on a connection from this channel. WebLogic SIP Server shuts off any connection where the messages size exceeds the configured value. The default size of 10,000,000 bytes is large. If you are concerned about preventing Denial Of Service (DOS) attacks against the server, reduce this attribute to a value that is compatible with your deployed services.
   - **Tcp Connect Timeout Millis**—This attribute specifies the amount of time WebLogic SIP Server waits before it declares a destination address (for an outbound connection) as unreachable. The attribute is applicable only to SIP channels; WebLogic SIP Server ignores this attribute value for SIPS channels.
9. Click Apply.

Configuring SIP Channels for Multi-Homed 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 WebLogic SIP Server. WebLogic SIP Server uses the listen address and listen port values for each channel when embedding routing information into SIP message system headers.

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

Configuring TCP and TLS Channels for Diameter Support

WebLogic SIP Server's Diameter implementation supports the Diameter protocol over the TCP or TLS transport protocols. To enable incoming Diameter connections on a server, you configure a dedicated network channel using the protocol type "diameter" for TCP transport, or "diameters"

for both TCP and TLS transport. The Diameter implementation application may automatically upgrade Diameter connections to use TLS as described in the Diameter specification (RFC 3558).

See "Configuring Diameter Sh Client Nodes and Relay Agents" on page 5-1 for more information about configuring network channels for Diameter protocol support.

Configuring Engine Servers to Listen on Any IP Interface (0.0.0.0)

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

Note: 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.

Configuring Unique Listen Address Attributes for Data Tier Replicas

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

1. Access the Administration Console for the WebLogic SIP Server domain.
2. In the left pane, select the name of the server to configure.
3. Select the Configuration->General tab.
4. Enter a unique DNS name or IP address in the Listen Address field.
5. Click Apply.

CHAPTER 8

Production Network Architectures and WebLogic SIP Server Configuration

The following sections describe common network architectures used in production deployments, and explain how WebLogic SIP Server is configured to run in those architectures:

Overview

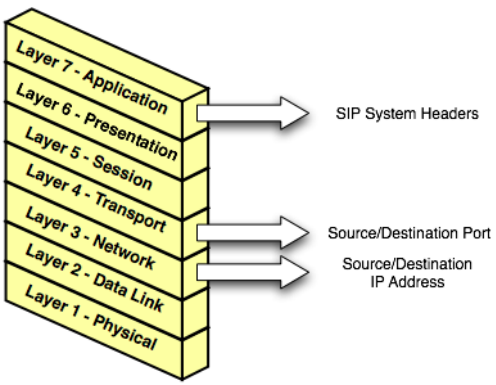
Most production installations of WebLogic SIP Server are contain one or more of the following characteristics:

- Multiple engine tier servers arranged in a cluster.
- Multiple network channels per engine tier server instance, in support of multiple SIP transport protocols or multiple Network Interface Cards (NICs) on multihomed hardware.
- One or more load balancers, or a multihomed load balancer, performing server failover and possibly Network Address Translation (NAT) for source or destination network packets.

A combination of these network elements can make it difficult to understand how elements interact with one another, and how a particular combination of elements or configuration options affects the contents of a SIP message or transport protocol datagram.

The sections that follow attempt to describe common WebLogic SIP Server network architectures and explain how servers are configured in each architecture. The sections also explain how information in SIP messages and transport datagrams is affected by each configuration. Figure 8-1 shows the typical Open Systems Interconnect (OSI) model layers that can be affected by different network configurations.

Figure 8-1 OSI Layers Affected by WebLogic SIP Server Network Configuration



Layer 3 (Network) and Layer 4 (Transport) contain the source or destination IP address and port numbers for both outgoing and incoming transport datagrams. Layer 7 (Application) may also be affected because the SIP protocol specifies that certain SIP headers include addressing information for contacting the sender of a SIP message.

Single-NIC Configurations with TCP and UDP Channels

In a simple network configuration for a server having a single NIC, one or more network channels may be created to support SIP messages over UDP and TCP, or SIPS over TLS. It is helpful to understand how this simple configuration affects information in the OSI model, so that you can understand how more complex configurations involving multihomed hardware and load balancers affect the same information.

Figure 8-2 Single-NIC Network Channel Configuration

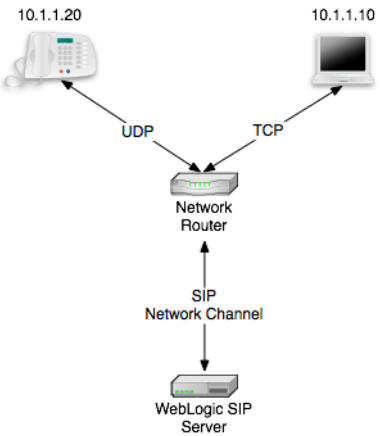


Figure 8-2 shows a single engine tier server instance with a single NIC. The server is configured with one network channel supporting SIP over UDP and TCP. (SIP channels always support both UDP and TCP transports; you cannot support only one of the two.) Figure 8-2 also shows two clients communicating with the server, one over UDP and one over TCP.

For the TCP transport, the outgoing datagram (delivered from WebLogic SIP Server to the UA) contains the following information:

- Layer 3 includes the source IP address specified by the network channel (10.1.1.10 in the example above).
- Layer 4 includes the source port number allocated by the underlying operating system.

Incoming TCP datagrams (delivered from the UA to WebLogic SIP Server) contain the following information:

- Layer 3 includes the destination IP address specified by the network channel (10.1.1.10).
- Layer 4 contains the destination port number specified by the network channel (5060).

For outgoing UDP datagrams, the OSI layer information contains the same information as with TCP transport. For incoming UDP datagrams, the OSI layer information is the same as TCP except in the case of incoming datagram Layer 4 information. For incoming UDP datagrams, Layer 4 contains either:

- The destination port number specified by the network channel (5060), or
- The ephemeral port number previously allocated by WebLogic SIP Server.

By default WebLogic SIP Server allocates ports from the ephemeral port number range of the underlying operating system for outgoing UDP datagrams. WebLogic SIP Server allows external connections to use an ephemeral point as the destination port number, in addition to the port number configured in the network channel. In other words, WebLogic SIP Server automatically listens on all ephemeral ports that the server allocates. You can optionally disable WebLogic SIP Server's use of ephemeral port numbers by specifying the following option when starting the server:

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

You can determine WebLogic SIP Server's use of a particular ephemeral port by examining the server log file:

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

Static Port Configuration for Outbound UDP Packets

WebLogic SIP 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.

WARNING: BEA does not recommend using the `SourcePorts` attribute in most configurations because it degrades performance. Configure `SourcePorts` only in cases where you must specify the exact ports that WebLogic SIP Server uses to originate UDP packets.

To configure `SourcePorts`, use a JMX client such as WLST or directly modify a network channel configuration in `config.xml` to include the attribute. `SourcePorts` defines an array of port numbers or port number ranges. Do not include spaces in the `SourcePorts` element - use only port numbers, hyphens ("-") to designate ranges of ports, and commas (",") to separate ranges or individual ports. See Listing 8-1 for an example configuration.

Listing 8-1 Static Port Configuration for Outgoing UDP Packets

```
<NetworkAccessPoint HttpEnabledForThisProtocol="false"
   ListenPort="5060" Name="sipchannel" OutboundEnabled="true"
   Protocol="sip" SourcePorts="6100-6150,6200-6250,6300"/>
```

Multihomed Server Configurations Overview

Engine tier servers in a production deployment frequently utilize multihomed server hardware, having two or more NICs. Multihomed hardware is typically used for one of the following purposes:

- To provide redundant network connections within the same subnet. Having multiple NICs ensures that one or more network connections are available to communicate with data tier servers or the Administration Server, even if a single NIC fails.
- To support SIP communication across two or more different subnets. For example WebLogic SIP Server may be configured to proxy SIP requests from UAs in one subnet to UAs in a second subnet, when the UAs cannot directly communicate across subnets.

The configuration requirements and OSI layer information differ depending on the use of multihomed hardware in your system. When multiple NICs are used to provide redundant connections within a subnet, servers are generally configured to listen on all available addresses (IP_ANY) as described in "Multihomed Servers Listening On All Addresses (IP_ANY)" on page 8-5.

When using multiple NICs to support different subnets, you must configure multiple network on the server for each different NIC as described in "Multihomed Servers Listening on Multiple Subnets" on page 8-6.

Multihomed Servers Listening On All Addresses (IP_ANY)

The simplest multihome configuration enables a WebLogic SIP Server instance to listen on all available NICs (physical NICs as well as logical NICs), sometimes described as IP_ANY. To accomplish this, you simply configure a single network channel and specify a channel listen address of 0.0.0.0.

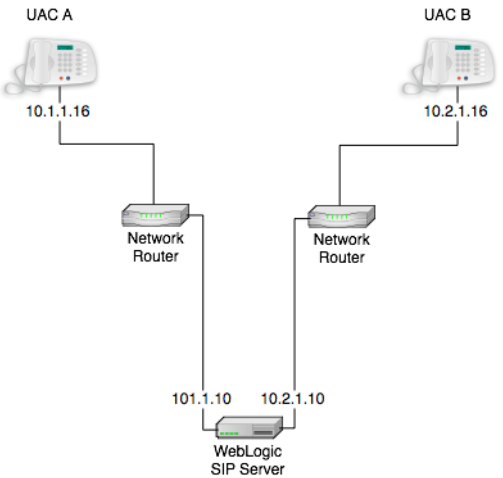
Note that you must configure the 0.0.0.0 address directly on the server's network channel. If you specify no IP address in the channel, the channel inherits the listen address configured for the

server instance itself. See “Configuring Engine Servers to Listen on Any IP Interface (0.0.0.0)” on page 7-6.

Multihomed Servers Listening on Multiple Subnets

Multiple NICs can also be used in engine tier servers to listen on multiple subnets. The most common configuration uses WebLogic SIP Server to proxy SIP traffic from one subnet to another where no direct access between subnets is permitted. Figure 8-3 shows this configuration.

Figure 8-3 Multihomed Configuration for Proxying between Subnets



To configure the WebLogic SIP Server instance in Figure 8-3 you must define a separate network channel for each NIC used on the server machine. Listing 8-2 shows the `config.xml` entries that define channels for the sample configuration.

Listing 8-2 Sample Network Channel Configuration for NICs on Multiple Subnets

```
<NetworkAccessPoint ListenAddress=”10.1.1.10” ListenPort="5060"
Name="sipchannelA" Protocol="sip"/>
```

```
<NetworkAccessPoint ListenAddress=”10.2.1.10” ListenPort="5060"
Name="sipchannelB" Protocol="sip"/>
```

Understanding the Route Resolver

When WebLogic SIP Server is configured to listen on multiple subsets, a feature called the *route resolver* is responsible for the following activities:

- Populating OSI Layer 7 information (SIP system headers such as Via, Contact, and so forth) with the correct address.
- Populating OSI Layer 3 information with the correct source IP address.

For example, in the configuration shown in Figure 8-3, WebLogic SIP Server must add the correct subnet address to SIP system headers and transport datagrams in order for each UA to continue processing SIP transactions. If the wrong subnet is used, replies cannot be delivered because neither UA can directly access the other UA’s subnet.

The route resolver works by determining which NIC the operating system will use to send a datagram to a given destination, and then examining the network channel that is associated with that NIC. It them uses the address configured in the selected network channel to populate SIP headers and Layer 3 address information.

For example, in the configuration shown in Figure 8-3, an INVITE message sent from WebLogic SIP Server to UAC B would have a destination address of 10.2.1.16. The operating system would transmit this message using NIC B, which is configured for the corresponding subnet. The route resolver associates NIC B with the configured `sipchannelB` and embeds the channel’s IP address (10.2.1.10) in the VIA header of the SIP message. UAC B then uses the VIA header to direct subsequent messages to the server using the correct IP address. A similar process is used for UAC A, to ensure that messages are delivered only on the correct subnet.

IP Aliasing with Multihomed Hardware

IP aliasing assigns multiple logical IP addresses to a single NIC, and is configured in the underlying server operating system. If you configure IP aliasing and all logical IP addresses are within the same subnet, you can simply configure WebLogic SIP Server to listen on all addresses as described in “Multihomed Servers Listening On All Addresses (IP_ANY)” on page 8-5.

If you configure IP aliasing to create multiple logical IP addresses on different subnets, you must configure a separate network channel for each logical IP address. In this configuration, WebLogic SIP Server treats all logical addresses as separate physical interfaces (NICs) and uses the route resolver to populate OSI Layer 4 and Layer 7 information based on the configured channel.

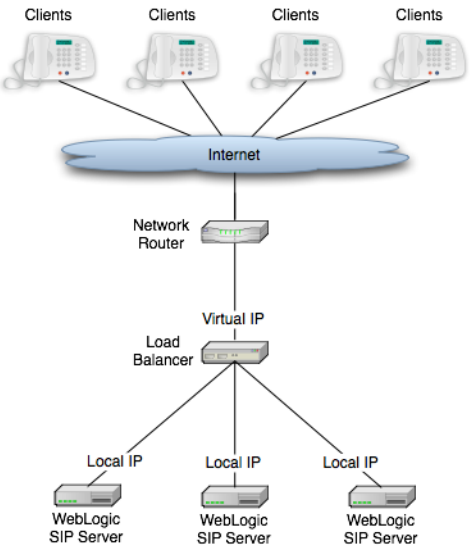
Load Balancer Configurations

In addition to providing failover capabilities and distributing the client load across multiple servers, a load balancer is also an important tool for configuring the network information transmitted between clients and servers. The sections that follow describe common load balancer configurations used with WebLogic SIP Server.

Single Load Balancer Configuration

The most common load balancer configuration utilizes a single load balancer that gates access to a cluster of engine tier servers, as shown in Figure 8-4.

Figure 8-4 SIngle Load Balancer Configuration



To configure WebLogic SIP Server for use with a single load balancer as in Figure 8-4, configure one or more network channels for each server, and configure the public address of each channel with the Virtual IP address of the load balancer. In this configuration, WebLogic SIP Server embeds the load balancer IP address in SIP message system headers to ensure that clients can reach the cluster for subsequent replies. "Managing WebLogic SIP Server Network Resources" on page 7-1 presents detailed steps for configuring network channels with load balancer addresses.

Note: Although some load balancing switches can automatically re-route all SIP messages in a given call to the same engine tier server, this functionality is not required with WebLogic SIP Server installations. See "Alternate Configurations" on page 1-5.

Multiple Load Balancers and Multihomed Load Balancers

Multiple load balancers (or a multihomed load balancer) can be configured to provide several virtual IP addresses for a single WebLogic SIP Server cluster. To configure WebLogic SIP Server for use with a multihomed load balancer, you create a dedicated network channel for each load balancer or local server NIC, and set the channel's public address to the virtual IP address of the appropriate load balancer. In this configuration, the route resolver associates a configured channel with the NIC used for originating SIP messages. The public address of the selected channel is then used for populating SIP system messages. See "Understanding the Route Resolver" on page 8-7.

Network Address Translation Options

In the most common case, a load balancer is configured using destination NAT to provide a public IP address that clients use for communicating with one or more internal (private) WebLogic SIP Server addresses. Load balancers may also be configured using source NAT, which modifies the Layer 3 address information originating from a private address to match the virtual IP address of the load balancer itself.

With the default route resolver behavior, a WebLogic SIP Server engine originates UDP packets having a source IP address that matches the address of a local NIC (the private address). This can be problematic for applications that try to respond directly to the Layer 3 address embedded in the transport packet, because the local server address may not be publicly accessible. If your applications exhibit this problem, BEA recommends that you configure the load balancer to perform source NAT to change the transport Layer 3 address to a publicly-accessible virtual IP address.

IP Masquerading Alternative to Source NAT

WARNING: Using the WebLogic SIP Server IP masquerading functionality can lead to network instability, because it requires duplicate IP addresses on multiple servers. Production deployments must use a load balancer configured for source NAT, rather than IP masquerading, to ensure reliable network performance.

If you choose not to enable source NAT on your load balancer, WebLogic SIP Server provides limited IP masquerading functionality. To use this functionality, configure a logical address on

each engine tier server using the public IP address of the load balancer for the cluster. (Duplicate the same logical IP address on each engine tier server machine). When a local server interface matches the IP address of a configured load balancer (defined in the public address of a network channel), WebLogic SIP Server uses that interface to originate SIP UDP messages, and the Layer 3 address contains a public address.

You can disable WebLogic SIP Server's IP masquerading functionality by using the startup option:

```
-Dwlss.udp.lb.masquerade=false
```

CHAPTER 9

Example WebLogic SIP Server Network Configuration

The following sections describe a sample network configuration for WebLogic SIP Server using a non-SIP-aware load balancer:

Overview

WebLogic SIP Server is compatible with load balancers that are not SIP-aware, meaning that they do not consider existing SIP dialogues when routing requests to servers. This document demonstrates load balancer and WebLogic SIP Server configuration, as well as SIP and Network Address Translation (NAT) interactions in various configurations.

For more information about implementation-dependent issues surrounding NAT see the IETF document, *NAT Behavioral Requirements for Unicast UDP*.

Example Network Topology

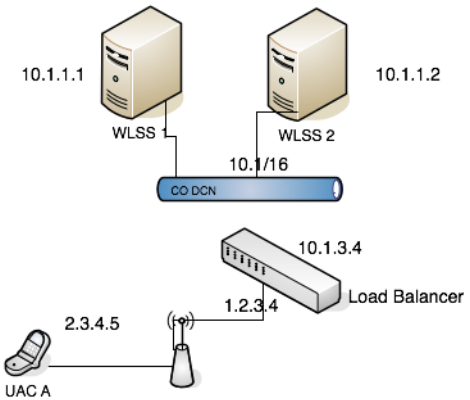
Figure 9-1 shows the sample network topology described in this section. A WebLogic SIP Server cluster, consisting of engines WLSS 1 and WLSS 2, is configured on private IP network 10.1/16

(an internal 16-bit subnet). The cluster's public IP address is 1.2.3.4, which is the virtual IP address configured on the load balancer.

The User Agent, UAC A, with IP address 2.3.4.5 never sees the internal IP addresses configured for the WebLogic SIP Server cluster. Instead, it sends requests to, and receives responses from 1.2.3.4.

The sections that follow discuss configuring the WebLogic SIP Server cluster and load balancer for this example system.

Figure 9-1 Example Network Topology



WebLogic SIP Server Configuration

The WebLogic SIP Server cluster configuration specifies the public address as 1.2.3.4, and the public port as 5060 (see "Configuring Load Balancer Addresses" on page 7-2) for each engine. The default route on both WebLogic SIP Server engines points to the load balancer's 10.1/16 network interface: 10.1.3.4. The WebLogic SIP Server (servers WLSS 1 and WLSS 2) routing table is shown in Listing 9-1.

Listing 9-1 WebLogic SIP Server Routing Table

```
$ /sbin/route
```

```
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.1.0.0          *             255.255.0.0     U     0      0        0 eth0
default         10.1.3.4         0.0.0.0         UG    0      0
```

Load Balancer Configuration

The load balancer is configured with a virtual IP address of 1.2.3.4, and two real servers, WLSS 1 and WLSS 2, having addresses 10.1.1.1 and 10.1.1.2, respectively. The load balancer also has an internal IP address of 10.1.3.4 configured on the 10.1/16 network. The UAC address, 2.3.4.5, is reachable from the load balancer by static route configuration on the load balancer. The load balancer routing table is shown in Listing 9-2.

Listing 9-2 Load balancer Routing Table

```
$ /sbin/route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.1.0.0          *             255.255.0.0     U     0      0        0 eth1
1.2.0.0           *              255.255.0.0     U     0      0
```

Because the SIP protocol specification (RFC 3261) dictates the destination IP address and UDP port numbers that user agents must use when sending requests or responses, the NAT configuration of the load balancer must be done in a way that does not violate RFC 3261 requirements. Three setup options can be used to accomplish this configuration:

- NAT-based configuration
- maddr-Based Configuration
- rport-Based Configuration

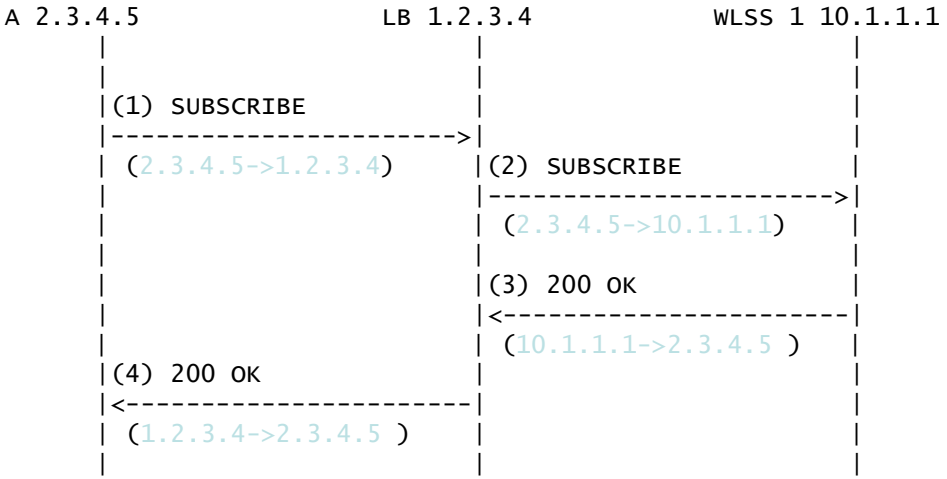
The sections that follow describe each approach.

NAT-based configuration

The default UDP NAT behavior for load balancers is to perform destination IP address translation in the public->private network direction, and source IP address translation in the private->public network direction. This means setting up destination address translation in the UAC->WebLogic SIP Server (2.3.4.5->1.2.3.4) direction without source address translation, and source address translation in the WebLogic SIP Server->UAC (10.1/16->2.3.4.5) direction without destination address translation.

Figure 9-2 illustrates the UDP packet flow for a SUBSCRIBE/200OK transaction.

Figure 9-2 SUBSCRIBE Sequence



Note that the source and destination IP addresses of the UDP packets are shown in blue. In the UAC->WebLogic SIP Server direction, the load balancer translates the destination IP address but not the source IP address. In the WebLogic SIP Server->UAC direction, the load balancer translates the source IP address but not the destination IP address.

The complete message trace (including IP and UDP headers, as well as the SIP payload) for the sequence from Figure 9-2 is shown in Listing 9-3 below.

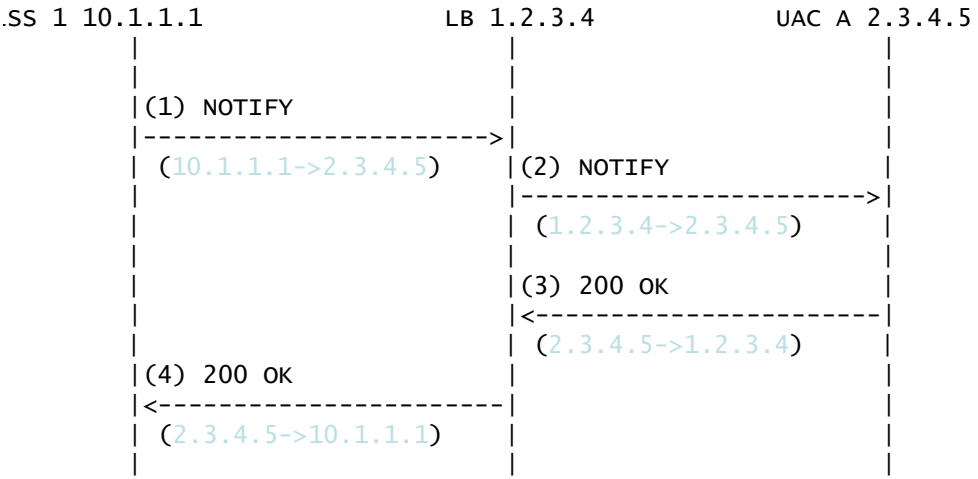
Listing 9-3 Complete SUBSCRIBE Message Trace

```
No.     Time        Source                Destination           Protocol Info
      1 1.425250    2.3.4.5           1.2.3.4          SIP      Request:
SUBSCRIBE sip:subscribe@1.2.3.4:5060


Internet Protocol, Src: 2.3.4.5 (2.3.4.5), Dst: 1.2.3.4 (1.2.3.4)
User Datagram Protocol, Src Port: 9999 (9999), Dst Port: sip (5060)
Session Initiation Protocol
    Request-Line: SUBSCRIBE sip:subscribe@1.2.3.4:5060 SIP/2.0
    Message Header
        Via: SIP/2.0/UDP 2.3.4.5:9999;branch=1
        From: sipp <sip:sipp@2.3.4.5>;tag=1
        To: sut <sip:subscribe@1.2.3.4:5060>
        Call-ID: 1-25923@2.3.4.5
        Cseq: 1 SUBSCRIBE
        Contact: sip:sipp@2.3.4.5:9999
        Max-Forwards: 70
        Event: ua-profile
        Expires: 10
        Content-Length: 0


No.     Time        Source                Destination           Protocol Info
      2 2.426250    2.3.4.5           10.1.1.1          SIP      Request:
SUBSCRIBE sip:subscribe@1.2.3.4:5060


Internet Protocol, Src: 2.3.4.5 (2.3.4.5), Dst: 10.1.1.1 (10.1.1.1)
User Datagram Protocol, Src Port: 9999 (9999), Dst Port: sip (5060)
```

```
Session Initiation Protocol
    Request-Line: SUBSCRIBE sip:subscribe@1.2.3.4:5060 SIP/2.0
    Message Header
        Via: SIP/2.0/UDP 2.3.4.5:9999;branch=1
        From: sipp <sip:sipp@2.3.4.5>;tag=1
        To: sut <sip:subscribe@1.2.3.4:5060>
        Call-ID: 1-25923@2.3.4.5
        Cseq: 1 SUBSCRIBE
        Contact: sip:sipp@2.3.4.5:9999
        Max-Forwards: 70
        Event: ua-profile
        Expires: 10
        Content-Length: 0

No.     Time        Source                Destination           Protocol Info
     3 3.430903   10.1.1.1            2.3.4.5         SIP     Status: 200 OK

Internet Protocol, Src: 10.1.1.1 (10.1.1.1), Dst: 2.3.4.5 (2.3.4.5)
User Datagram Protocol, Src Port: 42316 (42316), Dst Port: 9999 (9999)
Session Initiation Protocol
    Status-Line: SIP/2.0 200 OK
    Message Header
        To: sut <sip:subscribe@1.2.3.4:5060>;tag=82722c03
        Content-Length: 0
        Contact:
<sip:app-12eomtm5h5f77@1.2.3.4:5060;transport=udp;wlsscid=1ae4479ac6ff71>
        CSeq: 1 SUBSCRIBE
        Call-ID: 1-25923@2.3.4.5
```

If WebLogic SIP Server subsequently sends a NOTIFY request to the UAC, the sequence shown in Figure 9-3 takes place:

Figure 9-3 NOTIFY Sequence



As in the previous sequence, the IP address translation takes place in the WebLogic SIP Server->UAC direction for the source IP address, and UAC->WebLogic SIP Server direction for the destination IP address.

Note that this setup does not require the load balancer to maintain session state information or to be SIP-aware. The complete message trace from Figure 9-3 is shown in Listing 9-4 below.

Listing 9-4 Complete NOTIFY Message Trace

```
No.     Time        Source                Destination           Protocol Info
     1 5.430952    10.1.1.1               2.3.4.5          SIP     Request:
NOTIFY sip:sipp@2.3.4.5:9999


Internet Protocol, Src: 10.1.1.1 (10.1.1.1), Dst: 2.3.4.5 (2.3.4.5)
User Datagram Protocol, Src Port: 42316 (42316), Dst Port: 9999 (9999)
Session Initiation Protocol
```

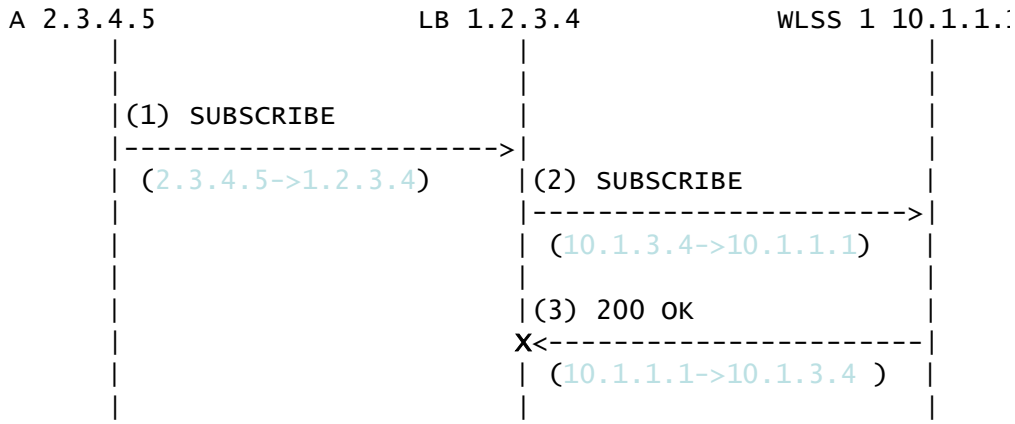
```
    Request-Line: NOTIFY sip:sipp@2.3.4.5:9999 SIP/2.0
    Message Header
        To: sipp <sip:sipp@2.3.4.5>;tag=1
        Content-Length: 0
        Contact:
<sip:app-12eomtm5h5f77@1.2.3.4:5060;transport=udp;wlsscid=1ae4479ac6ff71>
        CSeq: 1 NOTIFY
        Call-ID: 1-25923@2.3.4.5
        Via: SIP/2.0/UDP
1.2.3.4:5060;wlsscid=1ae4479ac6ff71;branch=z9hG4bKc5e4c3b4c22be517133ab749
adeece4e
        From: sut <sip:subscribe@1.2.3.4:5060>;tag=82722c03
        Max-Forwards: 70

No.     Time        Source                Destination           Protocol Info
      2 6.430952    1.2.3.4          2.3.4.5           SIP      Request:
NOTIFY sip:sipp@2.3.4.5:9999

Internet Protocol, Src: 1.2.3.4 (1.2.3.4), Dst: 2.3.4.5 (2.3.4.5)
User Datagram Protocol, Src Port: 2222 (2222), Dst Port: 9999 (9999)
Session Initiation Protocol
    Request-Line: NOTIFY sip:sipp@2.3.4.5:9999 SIP/2.0
    Message Header
        To: sipp <sip:sipp@2.3.4.5>;tag=1
        Content-Length: 0
        Contact:
<sip:app-12eomtm5h5f77@1.2.3.4:5060;transport=udp;wlsscid=1ae4479ac6ff71>
        CSeq: 1 NOTIFY
        Call-ID: 1-25923@2.3.4.5
```

```
        Via: SIP/2.0/UDP
1.2.3.4:5060;wlsscid=1ae4479ac6ff71;branch=z9hG4bKc5e4c3b4c22be517133ab749
adeece4e
        From: sut <sip:subscribe@1.2.3.4:5060>;tag=82722c03
        Max-Forwards: 70


No.     Time        Source                Destination           Protocol Info
      3 7.431367    2.3.4.5          1.2.3.4          SIP      Status: 200 OK


Internet Protocol, Src: 2.3.4.5 (2.3.4.5), Dst: 1.2.3.4 (1.2.3.4)
User Datagram Protocol, Src Port: 9999 (9999), Dst Port: sip (5060)
Session Initiation Protocol
    Status-Line: SIP/2.0 200 OK
    Message Header
        Via: SIP/2.0/UDP
1.2.3.4:5060;wlsscid=1ae4479ac6ff71;branch=z9hG4bKc5e4c3b4c22be517133ab749
adeece4e
        From: sut <sip:subscribe@1.2.3.4:5060>;tag=82722c03
        To: sipp <sip:sipp@2.3.4.5>;tag=1;tag=1
        Call-ID: 1-25923@2.3.4.5
        CSeq: 1 NOTIFY
        Contact: <sip:2.3.4.5:9999;transport=UDP>
```

WARNING: If NAT is performed on both the source (SNAT) and destination IP addresses, the configuration does not work because the load balancer usually relies on a specific destination port number value to be sent in responses to requests. That port number value is dictated by RFC 3261, and must come from the Via header, which presents a conflict with load balancer's NAT requirements. RFC 3261 requires that responses to SIP requests be sent to the IP address used to send the request (unless maddr is present in the Via, as described in “maddr-Based

Configuration" on page 9-12). Consequently, in Figure 9-4 below, Step 3, WebLogic SIP Server sends a 200 OK response to the load balancer internal IP address (10.1.3.4) and port 5060. That response is then dropped.

Figure 9-4 Source and Destination NAT



The complete message trace from Figure 9-4 is show in Listing 9-5 below.

Listing 9-5 Complete Failing SUBSCRIBE Message Trace

```
No.     Time        Source                Destination          Protocol Info
      1 1.425250    2.3.4.5           1.2.3.4          SIP      Request:
SUBSCRIBE sip:subscribe@1.2.3.4:5060


Internet Protocol, Src: 2.3.4.5 (2.3.4.5), Dst: 1.2.3.4 (1.2.3.4)
User Datagram Protocol, Src Port: 9999 (9999), Dst Port: sip (5060)
Session Initiation Protocol
    Request-Line: SUBSCRIBE sip:subscribe@1.2.3.4:5060 SIP/2.0
    Message Header
        Via: SIP/2.0/UDP 2.3.4.5:9999;branch=1
```

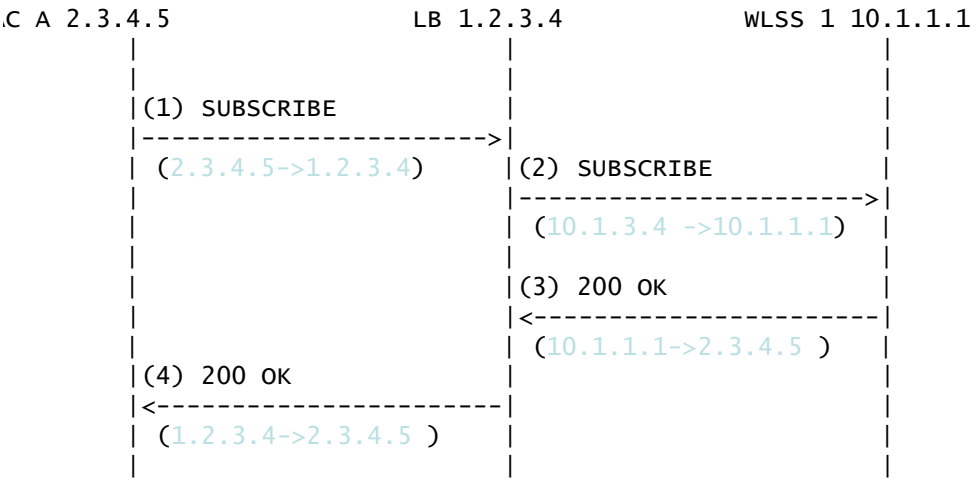
```
        From: sipp <sip:sipp@2.3.4.5>;tag=1
        To: sut <sip:subscribe@1.2.3.4:5060>
        Call-ID: 1-25923@2.3.4.5
        Cseq: 1 SUBSCRIBE
        Contact: sip:sipp@2.3.4.5:9999
        Max-Forwards: 70
        Event: ua-profile
        Expires: 10
        Content-Length: 0

No.     Time       Source                Destination           Protocol Info
      2 2.426250    10.1.3.4           10.1.1.1          SIP     Request:
SUBSCRIBE sip:subscribe@1.2.3.4:5060

Internet Protocol, Src: 10.1.3.4 (10.1.3.4), Dst: 10.1.1.1 (10.1.1.1)
User Datagram Protocol, Src Port: 2222 (2222), Dst Port: sip (5060)
Session Initiation Protocol
    Request-Line: SUBSCRIBE sip:subscribe@1.2.3.4:5060 SIP/2.0
    Message Header
        Via: SIP/2.0/UDP 2.3.4.5:9999;branch=1
        From: sipp <sip:sipp@2.3.4.5>;tag=1
        To: sut <sip:subscribe@1.2.3.4:5060>
        Call-ID: 1-25923@2.3.4.5
        Cseq: 1 SUBSCRIBE
        Contact: sip:sipp@2.3.4.5:9999
        Max-Forwards: 70
        Event: ua-profile
        Expires: 10
```

```
        Content-Length: 0

No.     Time        Source                Destination           Protocol Info
     3 3.430903    10.1.1.1            10.1.3.4         SIP     Status: 200 OK

Internet Protocol, Src: 10.1.1.1 (10.1.1.1), Dst: 10.1.3.4 (10.1.3.4)
User Datagram Protocol, Src Port: 42316 (42316), Dst Port: 9999 (9999)
Session Initiation Protocol
```

maddr-Based Configuration

When the maddr parameter is present in the Via header, the response is sent to the IP address specified in the maddr rather than to the received IP address (even when SNAT is enabled). In the example below, the UAC specifies a maddr set to 2.3.4.5 in the Via header. Consequently the response from the SIP server makes it to the UAC.

Figure 9-5 maddr Sequence



The complete message trace from Figure 9-5 is shown in Listing 9-6 below.

Listing 9-6 Complete maddr Message Trace

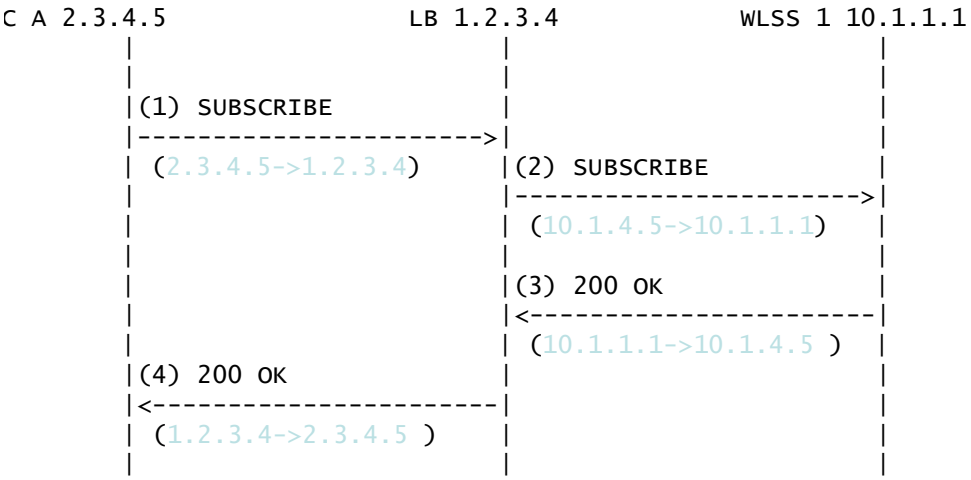
```
No.     Time        Source                Destination           Protocol Info
      1 1.425250    2.3.4.5           1.2.3.4          SIP      Request:
SUBSCRIBE sip:subscribe@1.2.3.4:5060

Internet Protocol, Src: 2.3.4.5 (2.3.4.5), Dst: 1.2.3.4 (1.2.3.4)
User Datagram Protocol, Src Port: 9999 (9999), Dst Port: sip (5060)
Session Initiation Protocol
    Request-Line: SUBSCRIBE sip:subscribe@1.2.3.4:5060 SIP/2.0
    Message Header
        Via: SIP/2.0/UDP 2.3.4.5:9999;maddr=2.3.4.5;branch=1
        From: sipp <sip:sipp@2.3.4.5>;tag=1
        To: sut <sip:subscribe@1.2.3.4:5060>
        Call-ID: 1-25923@2.3.4.5
        Cseq: 1 SUBSCRIBE
        Contact: sip:sipp@2.3.4.5:9999
        Max-Forwards: 70
        Event: ua-profile
        Expires: 10
        Content-Length: 0

No.     Time        Source                Destination           Protocol Info
      2 2.426250    10.1.3.4          10.1.1.1          SIP     Request:
SUBSCRIBE sip:subscribe@1.2.3.4:5060

Internet Protocol, Src: 10.1.3.4 (10.1.3.4), Dst: 10.1.1.1 (10.1.1.1)
User Datagram Protocol, Src Port: 2222 (2222), Dst Port: sip (5060)
```

```
Session Initiation Protocol
    Request-Line: SUBSCRIBE sip:subscribe@1.2.3.4:5060 SIP/2.0
    Message Header
        Via: SIP/2.0/UDP 2.3.4.5:9999;maddr=2.3.4.5;branch=1
        From: sipp <sip:sipp@2.3.4.5>;tag=1
        To: sut <sip:subscribe@1.2.3.4:5060>
        Call-ID: 1-25923@2.3.4.5
        Cseq: 1 SUBSCRIBE
        Contact: sip:sipp@2.3.4.5:9999
        Max-Forwards: 70
        Event: ua-profile
        Expires: 10
        Content-Length: 0

No.     Time        Source                Destination           Protocol Info
     3 3.430903   10.1.1.1            2.3.4.5         SIP     Status: 200 OK

Internet Protocol, Src: 10.1.1.1 (10.1.1.1), Dst: 2.3.4.5 (2.3.4.5)
User Datagram Protocol, Src Port: 42316 (42316), Dst Port: 9999 (9999)
Session Initiation Protocol
    Status-Line: SIP/2.0 200 OK
    Message Header
        To: sut <sip:subscribe@1.2.3.4:5060>;tag=82722c03
        Content-Length: 0
        Contact:
<sip:app-12eomtm5h5f77@1.2.3.4:5060;transport=udp;wlsscid=1ae4479ac6ff71>
```

rport-Based Configuration

RFC 3581 improves SIP and NAT interactions by allowing the client to request that the server send responses to a UDP port number from the request rather than from the Via. In order for both SUBSCRIBE and NOTIFY to work correctly, both the UAC as well as WebLogic SIP Server must support RFC 3581. Figure 9-6 illustrates the SUBSCRIBE flow.

Figure 9-6 rport SUBSCRIBE Sequence



The complete message trace from Figure 9-6 is shown in Listing 9-7 below.

Listing 9-7 Complete Message Trace for rport SUBSCRIBE

```
No.     Time        Source                Destination          Protocol Info
      1 1.425250    2.3.4.5           1.2.3.4          SIP      Request:
SUBSCRIBE sip:subscribe@1.2.3.4:5060


Internet Protocol, Src: 2.3.4.5 (2.3.4.5), Dst: 1.2.3.4 (1.2.3.4)
User Datagram Protocol, Src Port: 9999 (9999), Dst Port: sip (5060)
Session Initiation Protocol
```

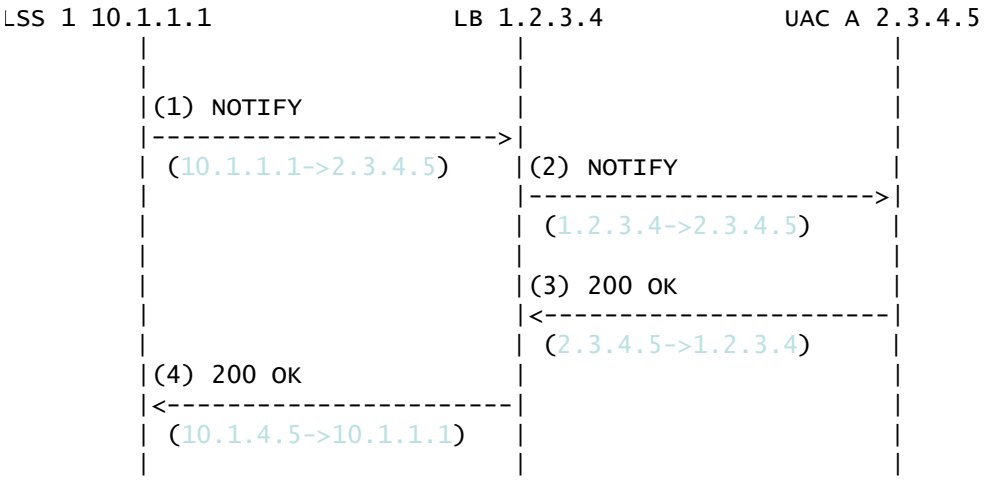
```
    Request-Line: SUBSCRIBE sip:subscribe@1.2.3.4:5060 SIP/2.0
    Message Header
        Via: SIP/2.0/UDP 2.3.4.5:9999;rport;branch=1
        From: sipp <sip:sipp@2.3.4.5>;tag=1
        To: sut <sip:subscribe@1.2.3.4:5060>
        Call-ID: 1-25923@2.3.4.5
        Cseq: 1 SUBSCRIBE
        Contact: sip:sipp@2.3.4.5:9999
        Max-Forwards: 70
        Event: ua-profile
        Expires: 10
        Content-Length: 0

No.     Time        Source                Destination           Protocol Info
      2 2.426250    10.1.3.4              10.1.1.1              SIP     Request:
SUBSCRIBE sip:subscribe@1.2.3.4:5060

Internet Protocol, Src: 10.1.3.4 (10.1.3.4), Dst: 10.1.1.1 (10.1.1.1)
User Datagram Protocol, Src Port: 2222 (2222), Dst Port: sip (5060)
Session Initiation Protocol
    Request-Line: SUBSCRIBE sip:subscribe@1.2.3.4:5060 SIP/2.0
    Message Header
        Via: SIP/2.0/UDP 2.3.4.5:9999;rport;branch=1
        From: sipp <sip:sipp@2.3.4.5>;tag=1
        To: sut <sip:subscribe@1.2.3.4:5060>
        Call-ID: 1-25923@2.3.4.5
        Cseq: 1 SUBSCRIBE
        Contact: sip:sipp@2.3.4.5:9999
```

```
        Max-Forwards: 70
        Event: ua-profile
        Expires: 10
        Content-Length: 0

No.     Time        Source                Destination           Protocol Info
     3 3.430903   10.1.1.1           10.1.3.4        SIP     Status: 200 OK

Internet Protocol, Src: 10.1.1.1 (10.1.1.1), Dst: 10.1.3.4 (10.1.3.4)
User Datagram Protocol, Src Port: 42316 (42316), Dst Port: 2222 (2222)
Session Initiation Protocol
    Status-Line: SIP/2.0 200 OK
    Message Header
        To: sut <sip:subscribe@1.2.3.4:5060>;tag=82722c03
        Content-Length: 0
        Contact:
<sip:app-12eomtm5h5f77@1.2.3.4:5060;transport=udp;wlsscid=1ae4479ac6ff71>
        CSeq: 1 SUBSCRIBE
        Call-ID: 1-25923@2.3.4.5
```

Figure 9-7 illustrates the NOTIFY flow.

Note that while source address NAT is enabled for both directions (UAS->WebLogic SIP Server and WebLogic SIP Server->UA), the load balancer can correctly identify the destination address in Step 3 by relying on receiving responses on the same port number as the one used to send requests. This implies that the load balancer maintains state.

Figure 9-7 rport NOTIFY Sequence



The complete message trace from Figure 9-7 is shown in Listing 9-8 below.

Listing 9-8 Complete Message Trace for rport NOTIFY

```
No.     Time        Source                Destination           Protocol Info
     1 5.430952    10.1.1.1              2.3.4.5          SIP     Request:
NOTIFY sip:sipp@2.3.4.5:9999


Internet Protocol, Src: 10.1.1.1 (10.1.1.1), Dst: 2.3.4.5 (2.3.4.5)
User Datagram Protocol, Src Port: 42316 (42316), Dst Port: 9999 (9999)
Session Initiation Protocol
    Request-Line: NOTIFY sip:sipp@2.3.4.5:9999 SIP/2.0
    Message Header
        To: sipp <sip:sipp@2.3.4.5>;tag=1
        Content-Length: 0
        Contact:
<sip:app-12eomtm5h5f77@1.2.3.4:5060;transport=udp;wlsscid=1ae4479ac6ff71>
```

```
        CSeq: 1 NOTIFY

        Call-ID: 1-25923@2.3.4.5

        Via: SIP/2.0/UDP
1.2.3.4:5060;wlsscid=1ae4479ac6ff71;branch=z9hG4bKc5e4c3b4c22be517133ab749
adeece4e;rport

        From: sut <sip:subscribe@1.2.3.4:5060>;tag=82722c03

        Max-Forwards: 70


No.     Time        Source                Destination           Protocol Info

      2 6.430952    1.2.3.4          2.3.4.5           SIP      Request:
NOTIFY sip:sipp@2.3.4.5:9999


Internet Protocol, Src: 1.2.3.4 (1.2.3.4), Dst: 2.3.4.5 (2.3.4.5)

User Datagram Protocol, Src Port: 2222 (2222), Dst Port: 9999 (9999)

Session Initiation Protocol

    Request-Line: NOTIFY sip:sipp@2.3.4.5:9999 SIP/2.0

    Message Header

        To: sipp <sip:sipp@2.3.4.5>;tag=1

        Content-Length: 0

        Contact:
<sip:app-12eomtm5h5f77@1.2.3.4:5060;transport=udp;wlsscid=1ae4479ac6ff71>

        CSeq: 1 NOTIFY

        Call-ID: 1-25923@2.3.4.5

        Via: SIP/2.0/UDP
1.2.3.4:5060;wlsscid=1ae4479ac6ff71;branch=z9hG4bKc5e4c3b4c22be517133ab749
adeece4e;rport

        From: sut <sip:subscribe@1.2.3.4:5060>;tag=82722c03

        Max-Forwards: 70
```

```
No.     Time        Source                Destination           Protocol Info
      3 7.431367    2.3.4.5          1.2.3.4         SIP      Status: 200 OK

Internet Protocol, Src: 2.3.4.5 (2.3.4.5), Dst: 1.2.3.4 (1.2.3.4)
User Datagram Protocol, Src Port: 9999 (9999), Dst Port: (2222)
Session Initiation Protocol
    Status-Line: SIP/2.0 200 OK
    Message Header
        Via: SIP/2.0/UDP
1.2.3.4:5060;wlsscid=1ae4479ac6ff71;branch=z9hG4bKc5e4c3b4c22be517133ab749
adeece4e;rport
        From: sut <sip:subscribe@1.2.3.4:5060>;tag=82722c03
        To: sipp <sip:sipp@2.3.4.5>;tag=1;tag=1
        Call-ID: 1-25923@2.3.4.5
        CSeq: 1 NOTIFY
        Contact: <sip:2.3.4.5:9999;transport=UDP
```

CHAPTER 10

Logging SIP Requests and Responses

The following sections describe how to configure and manage logging for SIP requests and responses:

Overview of SIP Logging

WebLogic SIP 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 WebLogic SIP Server, or in the log files for individual Managed Server instances. Because SIP messages share the same log files as WebLogic SIP 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 that is available in the template `sipserver-tracing.war` application. 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.

WebLogic SIP Server includes a template Web Application, `sipserver-tracing.war`, that defines several SIP logging Servlets. You can use the Servlets that are predefined in this application, or you can copy the Servlet implementation class into your own applications and define logging Servlets as needed. See "Using the Template Logging Servlet" on page 10-2. 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" on page 10-7.

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" on page 10-16 for an example of using the factory in a Servlet's `init()` method.

Using the Template Logging Servlet

The template logging application, `sipserver-tracing.war`, contains a logging Servlet implementation that you can customize to perform logging in a WebLogic SIP Server domain. You can either use `sipserver-tracing.war` as a standalone application that you configure and deploy along with other applications on your system, or you can incorporate the Servlet implementation class from `sipserver-tracing.war` directly into other applications to provide tracing functionality. The following sections describe each approach.

Deploying the Template Logging Application

Notes: The default SIP Logging Application is not deployed to new domains by default. Follow the instructions below to deploy the application.

If you want to create and deploy logging Servlets in your own applications (instead of using the template Web Application described below), you must package the `sipserver-tracing.jar` library from the template in your Web Application. The library is not deployed by default with the `sipserver` implementation application.

The default SIP Logging Servlets are included in a Web Application, *WL_HOME*`/telco/lib/sipserver-tracing.war`. To deploy this application:

1. Create a new directory from which to deploy the logging application. For example:

   ```
   cd c:\bea\user_projects\domains\mydomain
   mkdir sipserver-tracing
   ```

2. Change to the newly-created application directory:

   ```
   cd sipserver-tracing
   ```

3. Extract the logging application into the new application directory:

   ```
   jar xvf c:\bea\wlss220\telco\lib\sipserver-tracing.war
   ```

4. The logging Servlets are activated by deploying the `sipserver-tracing` application. Deploy the new application using either the Administration Console or the `weblogic.Deployer` utility. For example:

   ```
   java weblogic.Deployer -adminurl t3://localhost:7001 -user weblogic
   -password weblogic -deploy -nostage -source
   c:\bea\user_projects\domains\mydomain\sipserver-tracing
   ```

5. Deploying the unmodified application enables the template logging Servlets with default pattern matching configuration. See "Defining Logging Servlets in sip.xml" on page 10-4 and "Specifying the Criteria for Logging Messages" on page 10-7 for information about customizing the template application to perform logging for your system.

Using the Logging Servlet Implementation in Other Applications

Follow these steps to add logging capabilities to an existing application:

1. Create a temporary directory into which you will extract the template logging application:

```
mkdir c:\tracing-tmp
```

2. Change to the newly-created application directory:

   ```
   cd c:\tracing-tmp
   ```

3. Extract the template logging application into the new application directory:

   ```
   jar xvf c:\bea\wlss220\telco\lib\sipserver-tracing.war
   ```

4. Copy the `sipserver-tracing.jar` library from the temporary directory into the `WEB-INF/lib` directory of your own application. For example:

   ```
   cp WEB-INF\lib\sipserver-tracing.jar
   c:\bea\user_projects\mydomain\myapplication\WEB-INF\lib
   ```

5. See "Defining Logging Servlets in sip.xml" on page 10-4 to define a new logging Servlet in your existing application. Then read "Specifying the Criteria for Logging Messages" on page 10-7 to customizing the logging performed by the Servlet.

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 `sipserver-tracing` template application defines two logging servlets in its `sip.xml` deployment descriptor, `msgTraceLogger` and `invTraceLogger`. The definition for `msgTraceLogger` is shown in Listing 10-1.

Listing 10-1 Template Logging Servlets

```
<servlet>
    <servlet-name>msgTraceLogger</servlet-name>
    <servlet-class>com.bea.wcp.sip.engine.tracing.listener.TraceMessageLis
tenerImpl</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>
```

You can either maintain all of your logging Servlets within the template application, or you can add logging Servlets to your own SIP applications by defining Servlets that use the `com.bea.wcp.sip.engine.tracing.listener.TraceMessageListenerImpl` implementation class. See "Using the Template Logging Servlet" on page 10-2.

Configuring the Logging Level and Destination

Logging attributes such as the level of logging detail and the destination log file for SIP messages are passed as initialization parameters to the logging Servlet. Table 10-1 lists the parameters and parameter values that you can specify as `init-param` entries. Listing 10-1, "Template Logging

Servlets," on page 10-4 shows the sample `init-param` entries for a Servlet that logs full SIP message information to the domain log file.

Table 10-1 Logging Level and Destination Parameters

param-name Entry	Possible param-value Entries	Description
`domain`	`true`, `false`	The `domain` parameter determines if whether or not matching SIP messages are logged to the domain log file. If set to true, SIP Messages are logged to the domain log file as well as the local server log file. The default location of the domain log file is in a file named `wl-domain.log` in the domain directory. If set to false, WebLogic SIP Server logs SIP messages only to the Managed Server's local log file.
`level`	`terse`, `basic`, `full`	The `level` parameter determines the amount of information logged for each matching SIP message: • `terse`—Logs only `domain` setting, logging Servlet name, logging `level`, and whether or not the message is an incoming message. • `basic`—Logs the `terse` items plus the SIP message status, reason phrase, the type of response or request, the SIP method, the **From** header, and the **To** header. • `full`—Logs the `basic` items plus all SIP message headers plus the timestamp, protocol, request URI, request type, response type, content type, and raw content.

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 WebLogic SIP Server uses for selecting SIP requests and responses to place in the log file.

Note: By default WebLogic SIP Server logs both requests and responses. If you do not want to log responses, you must define a `response-pattern.xml` file with empty matching criteria.

A typical pattern definition defines a condition for matching a particular value in a SIP message header. For example, the sample `response-pattern.xml` used by the `msgTraceLogger` Servlet matches all MESSAGE requests. The contents of this descriptor are shown in

Listing 10-2 Sample response-pattern.xml for msgTraceLogger Servlet

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE pattern
   PUBLIC "Registration//Organization//Type Label//Definition Language"
   "trace-pattern.dtd">
<pattern>
  <equal>
    <var>response.method</var>
    <value>MESSAGE</value>
  </equal>
</pattern>
```

Additional operators and conditions for matching SIP messages are described in "trace-pattern.dtd Reference" on page 10-12. Most conditions, such as the `equal` condition shown in Listing 10-2, require a variable (`var` element) that identifies the portion of the SIP message to evaluate. Table 10-2 lists some common variables and sample values. For additional variable names and examples, see Chapter 11: Mapping Requests to Servlets in the *SIP Servlet API 1.0* specification; WebLogic SIP Server enables mapping of both request and response variables to logging Servlets.

Table 10-2 Pattern-matching Variables and Sample Values

Variable	Sample Values
request.method, response.method	MESSAGE, INVITE, ACK, BYE, CANCEL
request.uri.user, response.uri.user	guest, admin, joe
request.to.host, response.to.host	server.mydomain.com

Both `request-pattern.xml` and `response-pattern.xml` use the same Document Type Definition (DTD). See "trace-pattern.dtd Reference" on page 10-12 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" on page 10-5.

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" on page 10-7). 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. Listing 10-3 shows the full `sip.xml` entry for the sample logging Servlet, `invTraceLogger`. The final two `init-param` elements specify that the Servlet log only `INVITE` request methods and `OPTIONS` response methods.

Listing 10-3 Logging Criteria Specified as init-param Elements

```
<servlet>
      <servlet-name>invTraceLogger</servlet-name>
      <servlet-class>com.bea.wcp.sip.engine.tracing.listener.TraceMessageL
istenerImpl</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 WebLogic SIP 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, WebLogic SIP 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, WebLogic SIP 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, WebLogic SIP 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.

Listing 10-4 shows a sample `message-debug` configuration that logs String content for three additional Content-Type values, in addition to text/* and application/sdp content.

Listing 10-4 Logging String Content for Additional Content Types

```
<message-debug>
  <level>full</level>
  <string-rep>
    <content-type>application/msml+xml</content-type>
    <content-type>application/media_control+xml</content-type>
    <content-type>application/media_control</content-type>
  </string-rep>
</message-debug>
```

Managing Logging Performance

The SIP message logging implementation uses the threads in two execute queues, `sip.tracing.local` and `sip.tracing.domain`, to write log messages to the server and domain log files, respectively. By default each queue is configured with only a single thread. If the volume of log messages exceeds the capacity of either of these queues, log messages are

dropped and a notification of drop messages is written to the file. Normal logging continues when the volume of logged messages can be handled by the available threads.

If the number of dropped message notifications is unacceptable, follow these instructions to increase the number of threads available in the queue:

1. Access the Administration Console for the WebLogic SIP Server domain.
2. Expand the Servers node in the left pane of the Administration Console.
3. Right-click the name of the server that contains the execute queue you want to configure, and select View Execute Queues. (If you want to configure the queue used for writing to the domain log file, right-click any available server.)
4. In the right pane of the console, click either `sip.tracing.local` or `sip.tracing.domain` to configure the queue.
5. Edit the Thread Count value to change the number of threads allocated to the pool, or change any other Execute Queue properties to improve performance as needed.
6. Click Apply to apply your changes.
7. Reboot the WebLogic SIP Server instance to realize the change.

Enabling Log Rotation and Viewing Log Files

The WebLogic SIP Server logging infrastructure enables you to automatically write to a new log file when the existing log file reaches a specified size. You can also view log contents using the Administration Console or configure additional server-level events that are written to the log. See Server Log in the WebLogic Server 8.1 documentation for more information about basic log management.

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.

Listing 10-5 trace-pattern.dtd

```
<!--
```

```
The different types of conditions supported.
-->

<!ENTITY % condition "and | or | not |
                     equal | contains | exists | subdomain-of">

<!--
A pattern is a condition: a predicate over the set of SIP requests.
-->

<!ELEMENT pattern (%condition;)>

<!--
An "and" condition is true if and only if all its constituent conditions
are true.
-->

<!ELEMENT and (%condition;)+>

<!--
An "or" condition is true if at least one of its constituent conditions
is true.
-->

<!ELEMENT or (%condition;)+>

<!--
```

```
    Negates the value of the contained condition.
    -->

    <!ELEMENT not (%condition;)>

    <!--
    True if the value of the variable equals the specified literal value.
    -->

    <!ELEMENT equal (var, value)>

    <!--
    True if the value of the variable contains the specified literal value.
    -->

    <!ELEMENT contains (var, value)>

    <!--
    True if the specified variable exists.
    -->

    <!ELEMENT exists (var)>

    <!--
    -->

    <!ELEMENT subdomain-of (var, value)>
```

```
<!--
Specifies a variable. Example:
  <var>request.uri.user</var>
-->

<!ELEMENT var (#PCDATA)>

<!--
Specifies a literal string value that is used to specify rules.
-->

<!ELEMENT value (#PCDATA)>

<!--
Specifies whether the "equal" test is case sensitive or not.
-->

<!ATTLIST equal ignore-case (true|false) "false">

<!--
Specifies whether the "contains" test is case sensitive or not.
-->

<!ATTLIST contains ignore-case (true|false) "false">

<!--
```

```
The ID mechanism is to allow tools to easily make tool-specific
references to the elements of the deployment descriptor. This allows
tools that produce additional deployment information (i.e information
beyond the standard deployment descriptor information) to store the
non-standard information in a separate file, and easily refer from
these tools-specific files to the information in the standard sip-app
deployment descriptor.
-->

<!ATTLIST pattern id ID #IMPLIED>
<!ATTLIST and id ID #IMPLIED>
<!ATTLIST or id ID #IMPLIED>
<!ATTLIST not id ID #IMPLIED>
<!ATTLIST equal id ID #IMPLIED>
<!ATTLIST contains id ID #IMPLIED>
<!ATTLIST exists id ID #IMPLIED>
<!ATTLIST subdomain-of id ID #IMPLIED>
<!ATTLIST var id ID #IMPLIED>
<!ATTLIST value id ID #IMPLIED>
```

Adding Tracing Functionality to SIP Servlet Code

Tracing functionality can be added to your own Servlets or to Java code by using the `TraceMessageListenerFactory`. `TraceMessageListenerFactory` enables clients to reuse the default trace message listener implementation behaviors by creating an instance and then delegating to it. The factory implementation instance can be found in the servlet context for SIP Servlets by looking up the value of the `TraceMessageListenerFactory.TRACE_MESSAGE_LISTENER_FACTORY` attribute.

Note: Instances created by the factory are not registered with WebLogic SIP Server to receive callbacks upon SIP message arrival and departure.

To implement tracing in a Servlet, you use the factory class to create a delegate in the Servlet's `init()` method as shown in Listing 10-6.

Listing 10-6 Using the TraceMessageListenerFactory

```
public final class TraceMessageListenerImpl extends SipServlet implements
MessageListener {

  private MessageListener delegate;


  public void init() throws ServletException {

    ServletContext sc = (ServletContext) getServletContext();

    TraceMessageListenerFactory factory = (TraceMessageListenerFactory)
sc.getAttribute(TraceMessageListenerFactory.TRACE_MESSAGE_LISTENER_FACTORY);

    delegate = factory.createTraceMessageListener(getServletConfig());

  }

  public final void onRequest(SipServletRequest req, boolean incoming) {

    delegate.onRequest(req,incoming);

  }

  public final void onResponse(SipServletResponse resp, boolean incoming) {

    delegate.onResponse(resp,incoming);

  }

}
```

Order of Startup for Listeners and Logging Servlets

If you deploy both listeners and logging servlets, the listener classes are loaded first, followed by the Servlets. Logging Servlets are deployed in order according to the load order specified in their Web Application deployment descriptor.

CHAPTER 11

Avoiding and Recovering From Server Failures

A variety of events can lead to the failure of a server instance. Often one failure condition leads to another. Loss of power, hardware malfunction, operating system crashes, network partitions, or unexpected application behavior may each contribute to the failure of a server instance.

WebLogic SIP Server uses a highly clustered architecture as the basis for minimizing the impact of failure events. However, even in a clustered environment it is important to prepare for a sound recovery process in the event that an engine tier server, data tier server, or Diameter relay node were to suddenly fail.

The following sections provide information and procedures for recovering failed server instances:

Failure Prevention and Recovery Features

WebLogic SIP Server provides several features that facilitate recovery from and protection against server failure.

Overload Protection

WebLogic SIP Server detects increases in system load that could affect the performance and stability of deployed SIP Servlets, and automatically throttles message processing at predefined load thresholds.

Using overload protection helps you avoid failures that could result from unanticipated levels of application traffic or resource utilization.

WebLogic SIP Server attempts to avoid failure when certain conditions occur:

- The rate at which SIP sessions are created reaches a configured value, or
- The size of the SIP timer and SIP request-processing execute queues reaches a configured length.

See overload in the *Configuration Reference* for more information.

Redundancy and Failover for Clustered Services

You can increase the reliability and availability of your applications by using multiple engine tier servers in a dedicated cluster, as well as multiple data tier servers (replicas) in a dedicated data tier cluster. Because engine tier clusters maintain no stateful information about applications, the failure of an engine tier server does not result in any data loss or dropped calls. Multiple replicas in a data tier partition store redundant copies of call state information, and automatically failover to one another should a replica fail.

See "Overview of the WebLogic SIP Server Architecture" on page 1-1 for more information.

Automatic Restart for Failed Server Instances

Using Node Manager, server self-health monitoring enables you to automatically reboot servers that have failed. This improves the overall reliability of a domain, and requires no direct intervention from an administrator.

For more information, see Configuring, Starting, and Stopping Node Manager in the WebLogic Server 8.1 documentation.

Managed Server Independence Mode

Managed Servers maintain a local copy of the domain configuration. When a Managed Server starts, it contacts its Administration Server to retrieve any changes to the domain configuration that were made since the Managed Server was last shut down. If a Managed Server cannot

connect to the Administration Server during startup, it can use its locally-cached configuration information—this is the configuration that was current at the time of the Managed Server's most recent shutdown. A Managed Server that starts up without contacting its Administration Server to check for configuration updates is running in *Managed Server Independence (MSI)* mode. By default, MSI mode is enabled. See Replicating a Domain's Configuration Files for Managed Server Independence in the WebLogic Server 8.1 documentation.

Directory and File Backups for Failure Recovery

Recovery from the failure of a server instance requires access to the domain's configuration and security data. This section describes file backups that WebLogic SIP Server performs automatically, as well as manual backup procedures that an administrator should perform periodically.

Backing up config.xml

By default, an Administration Server stores a domain's configuration data in a file called *domain_name*/config.xml, where *domain_name* is the root directory of the domain.

Back up config.xml to a secure location in case a failure of the Administration Server renders the original copy unavailable. BEA recommends storing each new version of a config.xml file to a source control repository. If an Administration Server fails, you can copy the most recent backup version to a different machine and restart the Administration Server on that machine.

Automated config.xml Archiving

By default, the Administration Server archives up to 5 previous versions of config.xml in the *domain-name*/configArchive directory.

When you save a change to a domain's configuration, the Administration Server saves the previous configuration in *domain-name*\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.

To configure how the number of config.xml file versions that the server maintains:

1. In the left pane of the Administration Console, click on the name of the domain.
2. In the right pane, click the Configuration->General tab.
3. In the Advanced Options bar, click Show.
4. In the Archive Configuration Count box, enter the number of versions to save.
5. Click Apply.

Automatic Backup of config.xml at Server Startup

In addition to the files in *domain-name*\configArchive, the Administration Server creates two other files that back up the domain's configuration at key points during the startup process:

- *domain-name**config-file*.xml.original—The configuration file just before the Administration Server parses it and adds subsystem data.
- *domain-name**config-file*.xml.booted—The configuration file just after the Administration Server successfully boots. If the config.xml becomes corrupted, you can boot the Administration Server with this file.

Backing Up the sipserver Application

As with the config.xml file, the sipserver implementation application contains configuration information used by all engine and data tier servers deployed within a domain. The sipserver application also generally includes the diameter application for engine tier servers that act as Diameter client nodes.

By default the sipserver application is stored in *domain_name*/sipserver. Backup the entire application directory, which includes the sipserver.xml, datatier.xml, and diameter.xml configuration files, as well as any additional patches you may have installed.

Backing Up the Diameter Application

If you configure one or more WebLogic SIP Server instances to function as Diameter relay agent nodes, the Diameter Web Application is generally deployed as a standalone application (outside of the sipserver implementation application). Backup each Diameter application used to configure a relay agent node. This generally involves a separate Diameter application directory for each relay.

Backing Up Server Start Scripts

In a WebLogic SIP Server deployment, the start scripts used to boot engine and data tier servers are generally customized to include domain-specific configuration information such as:

- JVM Garbage Collection parameters required to achieve throughput targets for SIP message processing (see "Tuning JVM Garbage Collection for Production Deployments" on page F-1). Different parameters (and therefore, different start scripts) are generally used to boot engine and data tier servers.
- Configuration parameters and startup information for the WebLogic SIP Server heartbeat mechanism (see "Improving Failover Performance for Physical Network Failures" on page E-1). If you use the heartbeat mechanism, engine tier server start scripts should include startup options to enable and configure the heartbeat mechanism. Data tier server start scripts should include startup options to enable heartbeats and start the `WlssEchoServer` process.

Backup each distinct start script used to boot engine tier, data tier, or diameter relay servers in your domain.

Backing Up Logging Servlet Applications

If you use WebLogic SIP Server logging Servlets (see "Logging SIP Requests and Responses" on page 10-1) 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 WebLogic SIP Server store their data in an LDAP server. Each WebLogic SIP Server contains an embedded LDAP server. The Administration Server contains the master LDAP server, which is replicated on all Managed Servers. If any of your security realms use these installed providers, you should maintain an up-to-date backup of the following directory tree:

domain_name`\`*adminServer*`\ldap`

where *domain_name* is the domain's root directory and *adminServer* is the directory in which the Administration Server stores runtime and security data.

Each WebLogic SIP 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 Configuring Backups for the Embedded LDAP Server in the WebLogic Server 8.1 Documentation.

Backing Up SerializedSystemIni.dat and Security Certificates

All servers create a file named `SerializedSystemIni.dat` and place it in the server's root directory. This file contains encrypted security data that must be present to boot the server. You must back up this file.

If you configured a server to use SSL, also back up the security certificates and keys. The location of these files is user-configurable.

Backing Up Additional Operating System Configuration Files

Certain files maintained at the operating system level are also critical in helping you recover from system failures. Consider backing up the following information as necessary for your system:

- Load Balancer configuration scripts. For example, any automated scripts used to configure load balancer pools and virtual IP addresses for the engine tier cluster, as well as NAT configuration settings.
- NTP client configuration scripts used to synchronize the system clocks of engine and data tier servers.
- Host configuration files for each WebLogic SIP Server machine (host names, virtual and real IP addresses for multihomed machines, IP routing table information).

Restarting a Failed Administration Server

If no Managed Servers in the domain are running when you restart a failed Administration Server, no special steps are required. Start the Administration Server as you normally would.

If the Administration Server shuts down while Managed Servers continue to run, you do not need to restart the Managed Servers that are already running in order to recover management of the domain. The procedure for recovering management of an active domain depends upon whether you can restart the Administration Server on the same machine it was running on when the domain was started.

Restarting an Administration Server on the Same Machine

If you restart the WebLogic Administration Server while Managed Servers continue to run, by default the Administration Server can discover the presence of the running Managed Servers.

Note: Make sure that the startup command or startup script does not include `-Dweblogic.management.discover=false`, which disables an Administration Server from discovering its running Managed Servers.

The root directory for the domain contains a file, `running-managed-servers.xml`, which contains a list of the Managed Servers in the domain and describes whether they are running or not. When the Administration Server restarts, it checks this file to determine which Managed Servers were under its control before it stopped running.

When a Managed Server is gracefully or forcefully shut down, its status in `running-managed-servers.xml` is updated to “not-running”. When an Administration Server restarts, it does not try to discover Managed Servers with the “not-running” status. A Managed Server that stops running because of a system crash, or that was stopped by killing the JVM or the command prompt (shell) in which it was running, will still have the status “running’ in `running-managed-servers.xml`. The Administration Server will attempt to discover them, and will throw an exception when it determines that the Managed Server is no longer running.

Restarting the Administration Server does not cause Managed Servers to update the configuration of static attributes. *Static attributes* are those that a server refers to only during its startup process. Servers instances must be restarted to take account of changes to static configuration attributes. Discovery of the Managed Servers only enables the Administration Server to monitor the Managed Servers or make runtime changes in attributes that can be configured while a server is running (dynamic attributes).

Restarting an Administration Server on Another Machine

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

1. Install the WebLogic SIP Server software on the new administration machine (if this has not already been done).
2. Make your application files available to the new Administration Server by copying them from backups or by using a shared disk. Your application files should be available in the same relative location on the new file system as on the file system of the original Administration Server.
3. Make your configuration and security data available to the new administration machine by copying them from backups or by using a shared disk. For more information, refer to "Backing Up Security Data" on page 11-5.
4. Restart the Administration Server on the new machine.

   Make sure that the startup command or startup script does not include `-Dweblogic.management.discover=false`, which disables an Administration Server from discovering its running Managed Servers.

When the Administration Server starts, it communicates with the Managed Servers and informs them that the Administration Server is now running on a different IP address.

Restarting Failed Managed Servers

If the Administration Server is reachable by Managed Server that failed, you can:

- Restart it manually or automatically using Node Manager—You must configure Node Manager and the Managed Server to support this behavior. For details, see Configuring, Starting, and Stopping Node Manager in the WebLogic Server 8.1 documentation.
- Start it manually with a command or script.

If a Managed Server cannot connect to the Administration Server during startup, it can retrieve its configuration by reading locally-cached configuration data. A Managed Server that starts in this way is running in Managed Server Independence (MSI) mode. For a description of MSI mode, and the files that a Managed Server must access to start up in MSI mode, see Replicating a Domain's Configuration Files for Managed Server Independence in the WebLogic Server 8.1 documentation.

To start up a Managed Server in MSI mode:

1. Ensure that the following files are available in the Managed Server's root directory:
   - `msi-config.xml`.
   - `SerializedSystemIni.dat`
   - `boot.properties`

   If these files are not in the Managed Server's root directory:

   a. Copy the `config.xml` and `SerializedSystemIni.dat` file from the Administration Server's root directory (or from a backup) to the Managed Server's root directory.

   b. Rename the configuration file to `msi-config.xml`. When you start the server, it will use the copied configuration files.

   Note: Alternatively, use the `-Dweblogic.RootDirectory=`*`path`* startup option to specify a root directory that already contains these files.

2. Start the Managed Server at the command line or using a script.

   The Managed Server will run in MSI mode until it is contacted by its Administration Server. For information about restarting the Administration Server in this scenario, see "Restarting a Failed Administration Server" on page 11-7.

CHAPTER 12

Configuring SNMP

The following sections describe how to configure and manage SNMP services with WebLogic SIP Server 2.2:

Overview of WebLogic SIP Server SNMP

WebLogic SIP Server includes a dedicated SNMP MIB to monitor activity on engine tier and data tier server instances. The WebLogic SIP Server MIB is available on both Managed Servers and the Administration Server of a domain. However, WebLogic SIP Server engine and data tier traps are generated only by the Managed Server instances that make up each tier. If your Administration Server does not deploy the `sipserver` implementation EAR, it will generate only WebLogic Server 8.1 SNMP traps (for example, when a server in a cluster fails). Administrators should monitor both WebLogic Server 8.1 and WebLogic SIP Server traps to evaluate the behavior of the entire domain.

Note: WebLogic SIP Server MIB objects are read-only. You cannot modify a WebLogic SIP Server configuration using SNMP.

Browsing the MIB

You can use either of the following methods to browse the contents of the WebLogic SIP Server MIB:

- Use a MIB browser. WebLogic SIP Server does not provide a MIB browser, but most vendors of SNMP utilities do. The MIB is located in a file named *WLSS_HOME*/telco/lib/BEA-WLSS-MIB.asn1.
- Use a Web browser to view the WebLogic SIP Server SNMP MIB Reference on the BEA e-docs Web site.

  Because the MIB Reference uses Javascript and DHTML to provide browsing capabilities that are similar to a MIB browser, you must use one of the following Web browsers:

  – Firefox
  – Internet Explorer, version 5 or higher
  – Mozilla
  – Netscape Navigator, version 6 or higher
  – Opera 7 or higher

Configuring SNMP

To enable SNMP monitoring for the entire WebLogic SIP Server domain, follow these steps:

1. Login to the Administration Console for the WebLogic SIP Server domain.
2. In the left pane, select the Services->SNMP node.
3. Select the Enabled check box to enable SNMP.

   Note: WebLogic SIP Server instances ignore the SNMP port number specified on this page. See "SNMP Port Binding for WebLogic SIP Server" on page 12-2.
4. Click Apply to apply your changes.

SNMP Port Binding for WebLogic SIP Server

If you run multiple Managed Server instances on the same machine, each server instance would normally attempt to bind to the same configured SNMP port (for example, port 161). WebLogic SIP Server instances automatically manage SNMP port conflicts by automatically attempting to

bind to port 1610, and incrementing the port number as needed if the current port is unavailable. This helps to avoid a SNMP startup failure when multiple WebLogic SIP Server instances are deployed on the same server hardware.

You can also manually override the starting port number that WebLogic SIP Server attempts to bind to by supplying the `-DWLSS.SNMPPort=`*`port_number`* startup argument.

WARNING: If you specify the `-DWLSS.SNMPPort` option, ensure that the starting port number and subsequent numbers are unused on your system. The default starting port of 1610 was selected because no services are commonly bound to the 1610 port range.

Understanding and Responding to SNMP Traps

The following sections describe the WebLogic SIP Server SNMP traps in more detail. Recovery procedures for responding to individual traps are also included where applicable.

Files for Troubleshooting

The following WebLogic SIP Server log and configuration files are frequently helpful for troubleshooting problems, and may be required by your technical support contact:

- `$`*`DOMAIN_DIR`*`/config.xml`
- `$`*`DOMAIN_DIR`*`/sipserver/config/sipserver.xml`
- `$`*`DOMAIN_DIR`*`/*.log (domain log file)`
- `$`*`DOMAIN_DIR`*`/`*`servername`*`/*.log (server and access logs)`
- `sip.xml (in the /WEB-INF subdirectory of the application)`
- `web.xml (in the /WEB-INF subdirectory of the application)`

General information that can help the technical support team includes:

- The specific versions of:
  - WebLogic SIP Server
  - Java SDK
  - Operating System
- Thread dumps for hung WebLogic SIP Server processes
- Network analyzer logs

Trap Descriptions

Table 12-1 lists the WebLogic SIP Server SNMP traps and indicates whether the trap is generated by servers in the engine tier or data tier. Each trap is described in the sections that follow.

Table 12-1 WebLogic SIP Server SNMP Traps

Server Node in which Trap is Generated	Trap Name
Engine Tier Servers	"licenseLimitExceeded" on page 12-5
	"overloadControlActivated, overloadControlDeactivated" on page 12-8
	"sipAppDeployed" on page 12-10
	"sipAppUndeployed" on page 12-11
	"sipAppFailedToDeploy" on page 12-11
Engine and Data Tier Servers, if servers are members of a cluster	"serverStopped" on page 12-9
Data Tier Servers	"connectionLostToPeer" on page 12-4
	"connectionReestablishedToPeer" on page 12-5
	"dataTierServerStopped" on page 12-5
	"replicaAddedToPartition" on page 12-9
	"replicaRemovedFromPartition" on page 12-9

connectionLostToPeer

Description

This trap is generated by an engine tier server instance when it loses its connection to a replica in the data tier. It may indicate a network connection problem between the engine and data tiers, or may be generated with additional traps if a data tier server fails.

Recovery Procedure

If this trap occurs in isolation from other traps indicating a server failure, it generally indicates a network failure. Verify or repair the network connection between the affected engine tier server and the data tier server.

If the trap is accompanied by additional traps indicating a data tier server failure (for example, `dataTierServerStopped`), follow the recovery procedures for the associated traps.

connectionReestablishedToPeer

Description

This trap is generated by an engine tier server instance when it successfully reconnects to a data tier server after a prior failure (after a `connectionLostToPeer` trap was generated). Repeated instances of this trap may indicate an intermittent network failure between the engine and data tiers.

Recovery Procedure

See "connectionLostToPeer" on page 12-4.

dataTierServerStopped

Description

WebLogic SIP Server data tier nodes generate this alarm when an unrecoverable error occurs in a WebLogic Server instance that is part of the data tier. Note that this trap may be generated by the server that is shutting down, by another replica in the same partition, or in some cases by both servers (network outages can sometimes trigger both servers to generate the same trap).

Recovery Procedure

See the Recovery Procedure for "serverStopped" on page 12-9.

licenseLimitExceeded

Description

WebLogic SIP Server engine tier nodes generate this trap when it detects a license violation. This trap can occur if server usage or access exceeds the limitations specified in the license file, or if the file is accidentally modified. Never modify, move, or delete the license file during normal operations.

License violations may cause one or more of the following behaviors if the license file has been modified or corrupted:

- If the license signature in the license file was changed, WebLogic SIP Server may detect the anomaly and shut down. Furthermore, WebLogic SIP Server will not start up if the license signature is modified or corrupted.
- If the IP address in the license file is incorrect, WebLogic SIP Server will not start up.
- If the license expiration is reached WebLogic SIP Server will not start up. If the server is already running, it will shut down automatically after the expiration is reached.

Note: Permanent licenses have no expiration. If you purchased a permanent license but your license expires, your server may be using an evaluation license instead of your purchased license.

If usage reaches the maximum values set in the license file (max-sessions, max-registers or max-users) WebLogic SIP Server continues to run but rejects requests that exceed the defined limits. The following behaviors may be observed when usage limits are reached:

1. If the number of sessions has reached max-sessions and there is a request to create a new session:
   - WebLogic SIP Server generates a `licenseLimitExceeded` exception and rejects the request.
   - The application must handle the exception and notify the source of the request.
   - If the application passes the exception to the SIP container as is, WebLogic SIP Server returns a “503 Out Of Licensed Resources” response.
2. If there is an attempt to register a user beyond the maximum number specified in `max-users`:
   - The user management component generates an `IllegalLicenseException`.
   - The application must handle the exception and notify the source of the request.
   - If the application passes the exception to the SIP container as is, WebLogic SIP Server returns a “503 Out Of Licensed Resources” response.
   - Removing an existing user enables a new user to be registered.
3. If the number of connected terminals has reached `max-registers`:
   - The registrar servlet returns “503 Service Unavailable” to the new REGISTER request.
   - Registered terminals can still be refreshing or removed (UNREGISTERed).

– UNREGISTERing a terminal enables a new REGISTER to succeed.

The license file is an XML document located at `$BEA_HOME/license.bea`. The following sample shows a portion of an evaluation license:

```
<license-group format="1.0" product="WebLogic SIP Server" release="2.2">
    <license
        component="SIP Servlet Engine"
        expiration="never"
        ip="any"
        licensee="BEA Evaluation Customer"
        msgspersec="100"
        serial="616351266349-1813874379535"
        type="SDK"
       signature="MC0CFQDeWBkXTSZ5bO1qy0D/AfukgzqhDwIURsL8bkpTwlypiTSBq+d1b
dyzbRM="
    />
</license-group>
</license-group>
```

Recovery Procedure

1. Check the license file to insure that it has not been accidentally removed, changed or corrupted.
2. Check the expiration date in the license, and confirm that an EVAL license was not accidentally installed over the permanent license.
3. Notify Tier 4 Support of the condition, and send them a copy of the license file.

Additional License FAQs

Question: What IP should be used for licensing purposes?

Question: Each box has multiple IP addresses. Which IP should be assigned to the license?

Answer: Use the IP address that is returned with get local host command.

Question: I've upgraded my hardware system or need to move WebLogic SIP Server to a new machine. How do I modify the license file to use a new IP address?

Answer: Contact BEA Support with the updated IP address. BEA will generate a new license for you. You can then replace the license file with the updated file immediately without stopping WebLogic SIP Server.

overloadControlActivated, overloadControlDeactivated

Description

Weblogic SIP Server engine tier nodes use a configurable throttling mechanism that helps you control the number of new SIP requests that are processed. After a configured overload condition is observed, WebLogic SIP Server destroys new SIP requests by responding with "503 Service Unavailable" to the caller. The server continues to destroy new requests until the overload condition is resolved according to a configured threshold control value. This alarm is generated when the throttling mechanism is activated. The throttling behavior should eventually return the server to a non-overloaded state, and further action may be unnecessary. See Overload in the *Configuration Reference Manual*.

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 FAQs

Question: How can I monitor load using the Administration Server? How can I tell when I'm near a threshold?

Answer: If you set the queue length as an incoming call overload control, you can monitor the length of the queue using the Administration Console. If you specify a session rate control, you cannot monitor the session rate using the Administration Console. (The Administration Console only displays the current number of SIP sessions, not the rate of new sessions generated.)

replicaAddedToPartition

Description

WebLogic SIP Server data tier nodes generate this alarm when a server instance is added to a partition in the data tier.

Recovery Procedure

This trap is generated during normal startup procedures when data tier servers are booted.

replicaRemovedFromPartition

Description

WebLogic SIP Server data tier nodes generate this alarm when a server is removed from the data tier, either as a result of a normal shutdown operation or because of a failure. There must be at least one replica remaining in a partition to generate this trap; if a partition has only a single replica and that replica fails, the trap cannot be generated. In addition, because engine tier nodes determine when a replica has failed, an engine tier node must be running in order for this trap to be generated.

Recovery Procedure

If this trap is generated as a result of a server instance failure, additional traps will be generated to indicate the exception. See the recovery procedures for traps generated in addition to `replicaRemovedFromPartition`.

serverStopped

Description

This trap indicates that the WebLogic Server instance is now down. This trap applies to both engine tier and data tier server instances, but only when the servers are members of a named WebLogic Server cluster. If this trap is received spontaneously and not as a result of a controlled shutdown, follow the steps below.

Recovery Procedure

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 WebLogic SIP Server immediately. See Restarting Failed Server Instances in the WebLogic Server 8.1 documentation.

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: WebLogic SIP 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 FAQs

Question: If the server shuts down, are all SNMP traps for the server lost?

Answer: The Administration Console generates SNMP messages for managed WebLogic Server instances only until the ServerShutDown message is received. Afterwards, no additional messages are generated.

sipAppDeployed

Description

WebLogic SIP 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

Description

WebLogic SIP Server engine tier nodes generate this alarm when a SIP application shuts down, or if a SIP application is undeployed. This generally occurs when WebLogic SIP 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

Description

WebLogic SIP Server engine tier nodes generate this trap when an application deploys successfully as a Web Application but fails to deploy as a SIP application.

Recovery Procedure

The typical failure is caused by an invalid `sip.xml` configuration file and should occur only during software installation or upgrade procedures. When it occurs, undeploy the application, validate the sip.xml file, and retry the deployment.

Note: This alarm should never occur during normal operations. If it does, contact Tier 4 support immediately.

APPENDIX A

Upgrading Deployed SIP Applications

Note: The upgrade procedures described in this section apply only to SIP protocol applications. Converged applications that use both SIP and HTTP protocols cannot be upgraded in this manner. See "Upgrading Software and Converged Applications" on page B-1 instead.

The following sections describe how to upgrade deployed SIP applications to a newer version of the same application without losing active calls:

Overview of SIP Application Upgrades

With WebLogic SIP Server 2.2, 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.

WebLogic SIP 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.

WebLogic SIP 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 simply undeploy the newer version. WebLogic SIP Server then automatically directs all new requests to the older application version.

Requirements and Restrictions for Upgrading Deployed Applications

To use the application upgrade functionality of WebLogic SIP 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, WebLogic SIP Server automatically treats this application as the "older" version. See "Assign a Version Identifier" on page A-3.
- Both the deployed application and the updated application must provide only SIP protocol functionality. You cannot upgrade converged HTTP/SIP applications using these procedures. See "Upgrading Software and Converged Applications" on page B-1 instead.
- 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 SIP Server provides `SipApplicationRuntimeMBean` 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), WebLogic SIP Server always uses the latest version of an application when only the base name is supplied.

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 `SipApplicationRuntimeMBean` for Accessing the Application Name and Version Identifier.

Assign a Version Identifier

WebLogic SIP 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 valid characters as identified in Table A-1.

Table A-1 Valid and Invalid Characters

Valid ASCII Characters	Invalid Version Constructs
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, and you must append the version identifier to the WAR

filename. EAR deployments must additionally append the version identifier to the application name in the `context-root` specified in `application.xml`.

Defining the Version in the Manifest

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

Listing A-1 Version Identifier in Manifest

```
Manifest-Version: 1.0

Created-By: 1.4.1_05-b01 (Sun Microsystems Inc.)

Weblogic-Application-Version: v2
```

WebLogic SIP Server strips the version string specified in the manifest from the application's deployment name, so that it can recognize when multiple versions of the same application are deployed.

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

Appending the Version to the Archive Name

In the name of the SIP application WAR or EAR file, you must append the same version identifier that you defined in the manifest. Continuing with the example shown in Listing A-1, if the application is named `mySIPApplication`, you would package the deployment as `mySIPApplicationv2.war` (or `mySIPApplicationv2.ear` for an Enterprise Application).

Appending the Version to the context-root (Enterprise Applications)

If you are deploying an EAR file, the `context-root` element in the `application.xml` deployment descriptor must match the full application name and version string used in the archive name. For example:

```
<context-root>mySIPApplicationv2</context-root>
```

This context root requirement exists because you cannot deploy two different applications having the same context root. If your Enterprise Application must obtain a simple application name (without a version string) programmatically, you can use `SipApplicationRuntimeMBean` to retrieve the application name without the appended version string. See "Accessing the Application Name and Version Identifier" on page A-7.

Deploy the Updated Application Version

To begin the upgrade process, simply deploy the updated application archive using either the Administration Console or `weblogic.Deployer` utility. WebLogic SIP Server examines the version identifier in the manifest file, and strips the string value from the end of the deployment name 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 WebLogic SIP 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.

WebLogic SIP 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" on page A-3.

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. Listing A-2 shows the sample WLST commands for viewing the active session count for the `findme` sample application on the default single-server domain.

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

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

Listing A-2 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.In

fo@5ae636

-rw-   Name                                        myserver_myserver_findme_fin

dme

-rw-   ObjectName                                  examples:Location=myserver,N

ame=myserver_myserver_findme_findme,ServerRuntime=myserver,Type=SipApplication
Ru

ntime

-rw-   Parent                                      examples:Location=myserver,N

ame=myserver,Type=ServerRuntime

-rw-   Registered                                  false

-rw-   SipSessionCount                             0
```

```
-rw-   Type                                          SipApplicationRuntime

-rwx   preDeregister                                 void :
```

Roll Back the Upgrade Process

If you deploy a new version of an application and discover a problem with it, simply undeploying that version reverts the update process. WebLogic SIP Server automatically routes all requests to the earlier version of the application.

Note: Reverting the upgrade process results in losing all calls that were initiated using the new version of the application.

Accessing the Application Name and Version Identifier

If you intend to use WebLogic SIP Server's production upgrade feature, applications that are composed of multiple SIP Servlets should not hard-code the application name. This is because the deployment name and `context-root` of the application must change from version to version in order to support side-by-side deployment. Instead of hard-coding the application name, your application can dynamically access the base application name (without version identifier), the deployment name, or just the version identifier by using helper methods in `SipApplicationRuntimeMBean`. Table A-2 describes the methods as they would apply to an application deployed as "myApplicationv2.war" with the version identifier "v2" defined in the manifest.

Table A-2 SipApplicationRuntimeMBean Methods for Returning Name and Version Information

Method Signature	Description
`getApplicationVersion()`	Returns only the version string assigned to the application (for example, "v2".

Table A-2 SipApplicationRuntimeMBean Methods for Returning Name and Version Information

Method Signature	Description
`getActiveVersionState()`	Returns an int value that describes the state of the selected application version: • 0 indicates that this version of the application is INACTIVE. This means that the application is either still in the process of being deployed and has not yet been activated, or that WebLogic SIP Server is currently retiring the application. • 2 indicates that this version of the application is ACTIVE. The application is currently processing all *new* requests for the application.
`getApplicationName()`	Returns the base application name without the version identifier (for example, “myApplication”).

APPENDIX B

Upgrading Software and Converged Applications

Notes: The sections that follow provide only general instructions for upgrading WebLogic SIP Server software and converged applications. Your service pack or new software may contain additional instructions and tools to help you upgrade the software.

If you want to upgrade a SIP protocol application, rather than a converged HTTP/SIP protocol application, these instructions are not required. See "Upgrading Deployed SIP Applications" on page A-1 instead.

The following sections describe how to upgrade production WebLogic SIP Server installations to a new release of the software, and how to upgrade converged applications on a production server installation:

Overview of System and Application Upgrades

Because a typical production WebLogic SIP Server installation uses multiple server instances in both the engine and data tiers, upgrading the WebLogic SIP Server software, or a SIP Servlet

deployed to the engine tier, requires that you follow very specific practices. These practices ensure that:

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

The sections that follow describe how to use a configured load balancer to perform a "live" upgrade of the WebLogic SIP Server software, or a deployed SIP application on a production installation. The procedure for either type of upgrade is similar, but each procedure is described in a separate section for clarity.

When upgrading the WebLogic SIP Server software (for example, in response to a Service Pack), or upgrading a SIP Servlet where the Servlet's session data is incompatible with the older version, a new engine tier cluster is created to host newly-upgraded engine tier instances or new versions of SIP Servlets. One-by-one, servers in the engine tier are shut-down, upgraded, and then restarted in the new target cluster. While servers are being upgraded, WebLogic SIP Server automatically forwards requests from one engine tier cluster to the other as necessary to ensure that data tier requests are always initiated by a compatible engine tier server. After all servers have been upgraded, the older cluster is removed and no longer used. After upgrading the engine tier cluster, servers in the data tier may also be upgraded, one-by-one. See "Upgrading a Deployed Production Application (Compatible Session Data)" on page B-12 for more information.

Requirements for Upgrading a Production System

To upgrade a production WebLogic SIP Server installation you require:

- Nostage-mode deployments for existing converged applications. (Nostage-mode deployment enables you to upgrade a deployed SIP Servlet without performing a redeployment operation, as described in "Upgrading a Deployed Production Application (Compatible Session Data)" on page B-12.)
- Cluster-targeted deployments for all converged applications. All deployed SIP Servlets must be targeted to the engine tier cluster, rather than to individual Managed Server instances within the cluster. Cluster-level targeting is required in order to perform an upgrade of the WebLogic SIP Server software without disrupting existing clients.
- A compatible load balancer product and administrator privileges for reconfiguring the load balancer virtual IP addresses and pools.

- Adequate disk space on the Administration Server machine and on each Managed Server machine for installing a copy of the new WebLogic SIP Server software (for server software upgrades only).
- Privileges for modifying configuration files on the WebLogic SIP Server Administration Server machine.
- Privileges for shutting down and starting up individual Managed Server instances.
- Three or more replicas in each partition of the data tier, in order to upgrade the WebLogic SIP Server software to a new version. With fewer than three replicas in each partition, it is not possible to safely upgrade a production data tier deployment as no backup replica would be available during the upgrade procedure.

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

Upgrading to a New Version of WebLogic SIP Server

Follow these steps to upgrade a production installation of WebLogic SIP Server to a newer version of the WebLogic SIP Server software. These instructions upgrade both the SIP Servlet container implementation and the data tier replication and failover implementation included in the `sipserver` Enterprise Application (EAR).

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

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

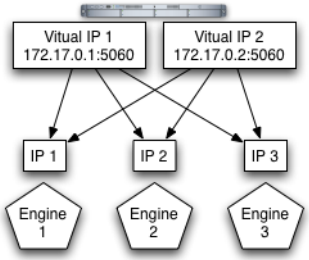
6. Upgrade Data Tier Servers—Shut down individual data tier servers, restarting them with the new data tier software implementation.

Each procedure is described in the sections that follow.

Configure the Load Balancer

Begin the software upgrade procedure by defining a new internal virtual IP address for the new engine tier cluster you will create in "Configure the New Engine Tier Cluster" on page B-4. The individual server IP addresses (the pool definition) for the new virtual IP address should be identical to the pool definition of your currently-active engine tier cluster. Figure B-1 shows a sample configuration for a cluster having three engine tier server instances; both virtual IP addresses define the same servers.

Figure B-1 Virtual IP Address Configuration for Parallel Clusters



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

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

Configure the New Engine Tier Cluster

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

1. On the Administration Server machine, install the new WebLogic SIP Server software into a new BEA home directory. The steps that follow refer to `c:\beanew` as the BEA home directory in which the new software was installed. `c:\bea` refers to the software implementation that is being upgraded.
2. On the Administration Server machine, copy the new `sipserver` application directory into a new directory from which it will be deployed. For example:

   ```
   cp -r c:\beanew\wlss220\samples\domains\telco\sipserver c:\deployments
   ```

3. Log in to the Administration Console for the active WebLogic SIP Server domain.
4. In the Administration Console, create a new, empty engine tier cluster for hosting the upgraded engine tier servers:
   a. In the left pane, select the Clusters node.
   b. Select Configure a New Cluster...
   c. Enter a name for the new cluster. For example, “NewEngineCluster.”
   d. Click Create to create the cluster.
5. Proceed to “Define the Cluster-to-Load Balancer Mapping” on page B-5.

Note: During the upgrade process you need to target the new `sipserver` Enterprise Application, as well as your own converged applications, to the newly-created cluster. However, you cannot target applications to an empty cluster. For this reason, targeting applications to the new cluster occurs only after you have added the first engine tier server to the new cluster in “Upgrade Engine Tier Servers and Target Applications to the New Cluster” on page B-7.

Define the Cluster-to-Load Balancer Mapping

In this procedure, you manually edit the active `sipserver.xml` configuration file to define the `cluster-loadbalancer-map` element. This XML element defines the internal, virtual IP address that is assigned to the older and newer engine tier clusters.

To define the cluster-to-load balancer mapping:

1. Move to the directory containing the `sipserver.xml` configuration file for your production domain. For example:

   ```
   cd c:\bea\user_projects\domains\mydomain\sipserver\config
   ```

2. Open the `sipserver.xml` file with a text editor:

```
notepad sipserver.xml
```

3. Add a `cluster-loadbalancer-map` element definition to the end of the configuration file, before the final `</sip-server>` line. The full definition must include a mapping for both the older and the newer engine tier cluster. A mapping consists of the internal virtual IP address of the cluster configured on the load balancer, as well as the cluster name defined in the WebLogic SIP Server domain. Listing B-1 shows an entry for the sample clusters described earlier.

Listing B-1 Sample cluster-loadbalancer-map Definition

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

4. Save your changes to `sipserver.xml` and exit your text editor.

Duplicate the SIP Servlet Container and Data Tier Configuration

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

```
cp c:\bea\user_projects\domains\mydomain\sipserver\config\sipserver.xml
c:\deployments\sipserver\config
```

```
cp c:\bea\user_projects\domains\mydomain\sipserver\config\datatier.xml
c:\deployments\sipserver\config
```

The `sipserver.xml` file in both the old and the new `sipserver` application should now be identical; only the implementation classes for each application are different. As engine tier servers are restarted in the new engine tier cluster in the next procedure, they will have the same SIP container configuration but will use the new container implementation.

Upgrade Engine Tier Servers and Target Applications to the New Cluster

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

1. Access the Administration Console for your production domain.
2. Select the first running engine tier server that you want to upgrade:
   a. Expand the Servers tab in the left pane.
   b. Select the name of the server you want to upgrade.
3. Select the Control->Start/Stop tab in the right pane.
4. Select Graceful shutdown of this server...
5. Select Yes to perform the shutdown.

   The server remains active while clients are still accessing the server, but no new connection requests are accepted. After all existing client connections have ended or timed out, the server shuts down. Other server instances in the engine tier process client requests during the shutdown procedure.
6. Select the Servers tab in the left pane and verify that the Managed Server has shut down.
7. Change the stopped server's cluster membership so that it is a member of the new engine tier cluster:
   a. Expand the Clusters tab in the left pane.
   b. Select the name of the active engine tier cluster.
   c. Select the Configuration->Servers tab in the right pane.
   d. Select the name of the stopped server in the Chosen column, and use the arrow to move it to the Available column.
   e. Click Apply.

f. Next, expand the Clusters tab and select the newly-created engine tier cluster ("NewEngineCluster").

g. Select the Configuration->Servers tab in the right pane.

h. Select the name of the stopped server in the Available column, and use the arrow to move it to the Chosen column.

i. Click Apply.

8. After adding the first engine tier server to the new cluster, you can now target the `sipserver` Enterprise Application and your own converged applications to the new cluster.

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

   a. In the left pane, select the Deployments->Applications node.

   b. Select Deploy a new Application...

   c. Using the links in the Location field, select the new `sipserver` application directory on the Administration Server machine (for example, `c:\deployments\sipserver`).

   d. Click Target Application.

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

   f. Click Continue.

   g. Select the option, I will make the application accessible from the following location.

   h. Click Deploy.

   i. Repeat this step to deploy all of your converged applications to the new cluster. Both the new and old engine tier clusters should be configured similarly, except that the new cluster hosts the new `sipserver` application while the existing cluster hosts the older `sipserver` application.

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

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

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

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

10. In the Administration Console, select the Servers node and verify that the Managed Server has started.

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

At this point, all running Managed Servers are using the new SIP Container implementation (the new `sipserver` application deployment) and are hosting your production SIP Servlets with the same SIP Servlet container settings as your old configuration. Data tier servers can now be upgraded using the instructions in "Upgrade Data Tier Servers" on page B-9.

Upgrade Data Tier Servers

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

The procedure for upgrading server instances in the data tier is similar to the procedure for upgrading servers in the engine tier, except that:

- No applications are targeted to the newly-created data tier cluster.
- No cluster-to-load balancer map is necessary for the parallel data tier cluster.

Apart from these differences, the process for upgrading a data tier cluster involves creating a new cluster for hosting upgraded server instances, targeting the new `sipserver` application to the new cluster, and restarting individual server instances in the new cluster. While upgrading individual data tier servers, care must be taken to ensure that each partition always contains an two active replicas in each partition to protect against hardware or software failures during the upgrade.

To upgrade data tier servers to a new WebLogic SIP Server implementation:

1. First perform all previous procedures to upgrade the engine tier servers in your domain. See "Upgrading to a New Version of WebLogic SIP Server" on page B-3.
2. Log in to the Administration Console for the active WebLogic SIP Server domain.
3. In the Administration Console, create a new, empty data tier cluster for hosting the upgraded data tier servers:
   a. In the left pane, select the Clusters node.

b. Select Configure a New Cluster...

c. Enter a name for the new cluster. For example, “NewDataCluster.”

d. Click Create to create the cluster.

4. Select a running data tier server that you want to upgrade:

   a. Expand the Servers tab in the left pane.

   b. Select the name of the server you want to upgrade.

WARNING: Do not shut down a data tier server instance in a partition unless two additional servers in the same partition are available, and both are in the `ONLINE` state. See “Monitoring and Troubleshooting Data Tier Servers” on page 3-6 for information about determining the state of data tier servers.

5. Select the Control->Start/Stop tab in the right pane.

6. Select Graceful shutdown of this server...

7. Select Yes to perform the shutdown.

   The server remains active while engine tier instances are still accessing the server, but no new connection requests are accepted. After all existing connections have ended, the server shuts down. Other replicas in the same data tier partition process engine tier requests for call state data during the shutdown procedure.

8. Select the Servers tab in the left pane and verify that the Managed Server has shut down.

9. Change the stopped server’s cluster membership so that it is a member of the new data tier cluster:

   a. Expand the Clusters tab in the left pane.

   b. Select the name of the active data tier cluster.

   c. Select the Configuration->Servers tab in the right pane.

   d. Select the name of the stopped server in the Chosen column, and use the arrow to move it to the Available column.

   e. Click Apply.

   f. Next, expand the Clusters tab and select the newly-created data tier cluster (“NewDataCluster”).

g. Select the Configuration->Servers tab in the right pane.

h. Select the name of the stopped server in the Available column, and use the arrow to move it to the Chosen column.

i. Click Apply.

10. After adding the first data tier server to the new cluster, target the new `sipserver` Enterprise Application to the cluster:

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

    a. In the left pane, select the Deployments->Applications node.

    b. Select Deploy a new Application...

    c. Using the links in the Location field, select the new `sipserver` application directory on the Administration Server machine (for example, `c:\deployments\sipserver`).

    d. Click Target Application.

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

    f. Click Continue.

    g. Select the option, I will make the application accessible from the following location.

    h. Click Deploy.

11. Restart the stopped managed server to bring it up in the new data tier cluster:

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

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

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

12. In the Administration Console, select the Servers node and verify that the Managed Server has started.

13. Repeat these steps to upgrade the remaining data tier servers.

14. After all data tier servers have been upgraded, delete the original data tier cluster:

    a. Select the Clusters node in the left pane.

b. Select the trash can icon next to the name of the older cluster in the cluster table.

c. Select Yes to delete the cluster definition.

15. Finally, delete the original engine tier cluster:

a. Select the Clusters node in the left pane.

b. Select the trash can icon next to the name of the older cluster in the cluster table.

c. Select Yes to delete the cluster definition.

At this point, all running Managed Servers are using the new WebLogic SIP Server data tier implementation (the new `sipserver` application deployment) and serving call state data to upgraded servers in the Engine Tier.

Upgrading a Deployed Production Application (Compatible Session Data)

This section describes how to upgrade a deployed SIP Servlet to a new version of the same SIP Servlet in a production environment. The instructions that follow assume that the session data used by the new Servlet version is compatible with the older Servlet version. If the session data is incompatible, see "Upgrading a Deployed Production Application (Incompatible Session Data)" on page B-13 instead.

WARNING: In order to upgrade a SIP Servlet using the procedure below, the session state information stored by the new version of the Servlet must be compatible with the older version of the Servlet. If the older and newer Servlets use incompatible session information, you must follow the instructions in "Upgrading a Deployed Production Application (Incompatible Session Data)" on page B-13 instead.

To upgrade an individual SIP Servlet to a new version:

1. On the Administration Server machine, make a backup copy of the source files used to deploy the older SIP Servlet version. You may need the older files if you decide to revert to the previously-deployed application. For example:

```
cd c:\deployments
mkdir myServlet_backup
cp -r myServlet\* myServlet_backup
```

2. Replace the current deployment source files with the updated version of the deployment files. For example, if the file list for the new version of the Servlet is the same as the old:

```
cp -r myNewServlet\* myServlet
```

If files have been deleted from the old servlet version, delete the original deployment files before copying over the new files:

```
rm -r myServlet\*
cp -r myNewServlet\* myServlet
```

At this point, the source files for the already-deployed SIP Servlet should represent the upgraded Servlet implementation. The currently-active SIP Servlet deployed to the engine tier uses the older version of the implementation.

3. To deploy the new source files and make the upgraded Servlet implementation active, use the Administration Console to gracefully shutdown a single Managed Server instance in the engine tier, and then restart the same server.
4. After the server has started and joined the cluster, repeat the previous step for an additional server in the engine tier. Repeat this process until each server in the engine tier has been restarted.

For nostage-mode deployments, the final two steps have the effect of deploying the SIP Servlet using the updated source files.

WARNING: It is important that the updated application files are deployed by gracefully shutting down and then restarting individual servers, rather than by simply redeploying the running application. Redeploying an application immediately unloads the application's implementation classes and replaces them with the newer classes. This makes the application unavailable during redeployment and may result in dropped client connections; ***never redeploy a running application in a production system***.

Upgrading a Deployed Production Application (Incompatible Session Data)

This section describes how to upgrade a deployed SIP Servlet to a new version when the session data used by the new Servlet is compatible with the older version. The upgrade procedure is similar to the procedure described in "Upgrading to a New Version of WebLogic SIP Server" on page B-3, except that the SIP Servlet container (`sipserver` application) is not upgraded.

The steps for performing this type of upgrade are divided into these high-level procedures:

1. Configure the Load Balancer—Define a new, internal Virtual IP address for the new engine tier cluster you will configure.

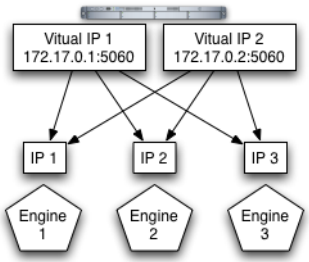
2. Configure the New Engine Tier Cluster—Create and configure a new, empty engine tier cluster that will host the new version of the SIP Servlet.
3. Define the Cluster-to-Load Balancer Mapping—Modify the SIP Servlet container configuration to indicate the virtual IP address of each engine tier cluster.
4. Migrate Engine Tier Servers and Target Applications to the New Cluster—Shut down individual engine tier server instances, restarting them in the new engine tier cluster.

Each procedure is described in the sections that follow.

Configure the Load Balancer

Begin the software upgrade procedure by defining a new internal virtual IP address for the new engine tier cluster you will create in "Configure the New Engine Tier Cluster" on page B-4. The individual server IP addresses (the pool definition) for the new virtual IP address should be identical to the pool definition of your currently-active engine tier cluster. Figure B-1 shows a sample configuration for a cluster having three engine tier server instances; both virtual IP addresses define the same servers.

Figure B-2 Virtual IP Address Configuration for Parallel Clusters



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

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

Configure the New Engine Tier Cluster

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

1. On the Administration Server machine, install the new WebLogic SIP Server software into a new BEA home directory. The steps that follow refer to `c:\beanew` as the BEA home directory in which the new software was installed. `c:\bea` refers to the software implementation that is being upgraded.
2. On the Administration Server machine, copy the new `sipserver` application directory into a new directory from which it will be deployed. For example:

   ```
   cp -r c:\beanew\wlss220\samples\domains\telco\sipserver c:\deployments
   ```

3. Log in to the Administration Console for the active WebLogic SIP Server domain.
4. In the Administration Console, create a new, empty engine tier cluster for hosting the upgraded engine tier servers:
   a. In the left pane, select the Clusters node.
   b. Select Configure a New Cluster...
   c. Enter a name for the new cluster. For example, “NewEngineCluster.”
   d. Click Create to create the cluster.
5. Proceed to “Define the Cluster-to-Load Balancer Mapping” on page B-5.

Note: During the upgrade process you need to target your new converged applications to the newly-created cluster. However, you cannot target applications to an empty cluster. For this reason, targeting applications to the new cluster occurs only after you have added the first engine tier server to the new cluster in “Migrate Engine Tier Servers and Target Applications to the New Cluster” on page B-16.

Define the Cluster-to-Load Balancer Mapping

In this procedure, you manually edit the active `sipserver.xml` configuration file to define the `cluster-loadbalancer-map` element. This XML element defines the internal, virtual IP address that is assigned to the older and newer engine tier clusters.

To define the cluster-to-load balancer mapping:

1. Move to the directory containing the `sipserver.xml` configuration file for your production domain. For example:

   ```
   cd c:\bea\user_projects\domains\mydomain\sipserver\config
   ```

2. Open the `sipserver.xml` file with a text editor:

   ```
   notepad sipserver.xml
   ```

3. Add a `cluster-loadbalancer-map` element definition to the end of the configuration file, before the final `</sip-server>` line. The full definition must include a mapping for both the older and the newer engine tier cluster. A mapping consists of the internal virtual IP address of the cluster configured on the load balancer, as well as the cluster name defined in the WebLogic SIP Server domain. Listing B-1 an entry for the sample clusters described earlier.

Listing B-2 Sample cluster-loadbalancer-map Definition

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

4. Save your changes to `sipserver.xml` and exit your text editor.

Migrate Engine Tier Servers and Target Applications to the New Cluster

To deploy the new version of the SIP Servlet, you gracefully shut each server down, change its cluster membership, and then restart it in the new cluster. Because you target the newer versions of your converged applications to the new cluster, restarting server instances in the new cluster deploys the latest application versions. Follow these steps:

1. Access the Administration Console for your production domain.

2. Select a running engine tier server that you want to upgrade:
   a. Expand the Servers tab in the left pane.
   b. Select the name of the server you want to upgrade.
3. Select the Control->Start/Stop tab in the right pane.
4. Select Graceful shutdown of this server...
5. Select Yes to perform the shutdown.

   The server remains active while clients are still accessing the server, but no new connection requests are accepted. After all existing client connections have ended or timed out, the server shuts down.

   Other server instances in the engine tier process client requests using the older version of the SIP Servlet, and the WebLogic SIP Servlet implementation automatically forwards requests to the appropriate cluster (using the cluster-to-load balancer map) so that each engine tier accesses the correct version of the application's session data.
6. Select the Servers tab in the left pane and verify that the Managed Server has shut down.
7. Change the stopped server's cluster membership so that it is a member of the new engine tier cluster:
   a. Expand the Clusters tab in the left pane.
   b. Select the name of the active engine tier cluster.
   c. Select the Configuration->Servers tab in the right pane.
   d. Select the name of the stopped server in the Chosen column, and use the arrow to move it to the Available column.
   e. Click Apply.
   f. Next, expand the Clusters tab and select the newly-created engine tier cluster ("NewEngineCluster").
   g. Select the Configuration->Servers tab in the right pane.
   h. Select the name of the stopped server in the Available column, and use the arrow to move it to the Chosen column.
   i. Click Apply.

8. After adding the first engine tier server to the new cluster, you can now target the new converged applications to the new cluster.

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

   a. In the left pane, select the Deployments->Applications node.

   b. Select Deploy a new Application...

   c. Using the links in the Location field, select the new version of an application on the Administration Server machine.

   d. Click Target Application.

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

   f. Click Continue.

   g. Select the option, I will make the application accessible from the following location.

   h. Click Deploy.

   i. Repeat this step to deploy all of your newer converged applications to the new cluster. Both the new and old engine tier clusters should be configured similarly, except that the new cluster hosts the newer applications while the existing cluster hosts the older applications.

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

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

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

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

10. In the Administration Console, select the Servers node and verify that the Managed Server has started.

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

12. After all engine tier servers have been upgraded, delete the original engine tier cluster:

    a. Select the Clusters node in the left pane.

    b. Select the trash can icon next to the name of the older cluster in the cluster table.

c. Select Yes to delete the cluster definition.

At this point, all running Managed Servers are using the new version of the SIP Servlet.

APPENDIX C

Applying Patches Using InstallPatch

The following sections provide instructions for applying patches to WebLogic SIP Server instances:

Overview of the InstallPatch Utility

The WebLogic SIP Server container functionality is implemented using an Enterprise Application (EAR) named `sipserver`. To patch the `sipserver` implementation EAR, you add the patch JAR file to the domain directory and then use the `InstallPatch` utility to add the JAR the application.

`InstallPatch` automates the process of editing the `sipserver` MANIFEST class path, which defines the list of JAR files used by the application and their relative order. You can use `InstallPatch` to perform common patching operations such as:

- Installing a new patch (JAR file)

- Removing a previously-applied patch
- Changing the order in which patches are loaded

Although it is possible to manually edit the MANIFEST class path in the `sipserver` application, BEA recommends using `InstallPatch` to avoid errors.

Required Environment for the InstallPatch Utility

To set up your environment for the `InstallPatch` utility:

1. Install the WebLogic SIP Server software 2.2. See Installing WebLogic SIP Server Using Graphical-Mode Installation in *Installing WebLogic SIP Server*.
2. Open a new shell or command prompt window.

   Note: Do not run the `setAdminClientEnv` script and `InstallPatch` utility from a shell that you previously used to start WebLogic SIP Server.
3. Move to the top level of the domain directory that you want to patch:

   ```
   cd BEA_HOME\user_projects\domains\mydomain
   ```

   In the above command, *`BEA_HOME`* refers to the top-level BEA installation directory (for example, `c:\bea`).
4. Set the client environment using the command

   ```
   WLSS_HOME\server\bin\setAdminClientEnv.cmd
   ```

   where *`WLSS_HOME`* is the directory in which you installed WebLogic SIP Server (for example, `c:\bea\wlss220`).

Syntax for Invoking the InstallPatch Utility

The InstallPatch utility can run in either command-line or GUI mode. By default the utility runs in command-line mode. The syntax for using the utility in command-line mode is:

```
java com.bea.wcp.sip.tools.InstallPatch
   [-mode (gui | cmdline)]
   -action (prepend | append | set | view)
   -patch filename.jar [-patch filename2.jar ...]
   [-help] [-verbose]
```

Note: All patch files must be located in the *`DOMAIN_DIR`*`\sipserver\APP-INF\container` directory, where *`DOMAIN_DIR`* is the top-level directory of the domain you are patching.

Note: If you are patching a replicated domain that uses "stage" mode deployment, you must manually delete the contents of the staging directory for each server (*DOMAIN_DIR*/*servername*/stage by default) to force redeployment of the sipserver application. Or, you must manually copy the newly-installed patch file(s) to the *DOMAIN_DIR*/*servername*/stage/sipserver/APP-INF/container directory of each Managed Server target.

When running the utility in GUI mode, all other options are ignored. To run the utility in GUI mode, use the command:

```
java com.bea.wcp.sip.tools.InstallPatch -mode gui
```

Table C-1 describes the arguments for the com.bea.wcp.sip.tools.InstallPatch utility.

Table C-1 InstallPatch Arguments

Argument	Definition
-mode (gui | cmdline)	Specifies whether to run the utility in GUI mode or in command-line mode. Command-line mode is used by default. If you run the utility in GUI mode, all other arguments are ignored.
-action (prepend | append | set | view)	Specifies a single action to perform on the CLASSPATH in command-line mode: • prepend—Adds one or more JAR files to the beginning of the existing CLASSPATH. The JAR files must be specified in subsequent -patch arguments. • append—Adds one or more JAR files to the end of the existing CLASSPATH. The JAR files must be specified in subsequent -patch arguments. • set—Re-writes the entire CLASSPATH using the JAR files specified in subsequent -patch arguments. If you use the set action, note that you must specify the default implementation JAR files (./wlss_sp.jar and ./wlss.jar) as well as any patch files you want to add. See "Example InstallPatch Commands" on page C-4 for more information. • view—Displays the current CLASSPATH.

Table C-1 InstallPatch Arguments

Argument	Definition
`-patch` *`filename.jar`*	Specifies the filename of a patch file to apply. You can apply multiple patches by specifying multiple `-patch` arguments. Multiple patches are applied in the order in which you specify them on the command line. **Note:** All patch files must be located in the *`DOMAIN_DIR`*`\sipserver\APP-INF\container` directory, where *`DOMAIN_DIR`* is the top-level directory of the domain you are patching.
`-mode gui`	Starts the utility in GUI mode. You cannot use the `-action` or `-patch` arguments when running in GUI mode.
`-help`	Displays usage information.
`-verbose`	Displays verbose output for command-line mode.

Example InstallPatch Commands

The following examples assume an initial MANIFEST classpath of:

```
./wlss_sp.jar ./wlss.jar
```

To add a new patch JAR file to the beginning of the classpath:

```
java com.bea.wcp.sip.tools.InstallPatch -action prepend -patch
CR567890_wlss220.jar
```

This yields the classpath:

```
./CR567890_wlss220.jar ./wlss_sp.jar ./wlss.jar
```

To add multiple patch JAR files to the end of the classpath:

```
java com.bea.wcp.sip.tools.InstallPatch -action append -patch
CR567891_wlss220.jar -patch CR567892_wlss220.jar
```

This yields the classpath:

```
./CR567890_wlss220.jar ./wlss_sp.jar ./wlss.jar ./CR567891_wlss220.jar
./CR567892_wlss220.jar
```

To remove one or more patches, re-write the CLASSPATH using the `set` action, as in:

```
java com.bea.wcp.sip.tools.InstallPatch -action set -patch
CR567890_wlss220.jar -patch wlss_sp.jar -patch wlss.jar -patch
CR567891_wlss220.jar
```

This yields the classpath:

```
./CR567890_wlss220.jar ./wlss_sp.jar ./wlss.jar ./CR567891_wlss220.jar
```

To view the current classpath, use the `-action view` option with the utility:

```
java com.bea.wcp.sip.tools.InstallPatch -action view
```

```
The current Manifest Class-Path is: ./CR567890_wlss220.jar ./wlss_sp.jar
./wlss.jar ./CR567891_wlss220.jar
```

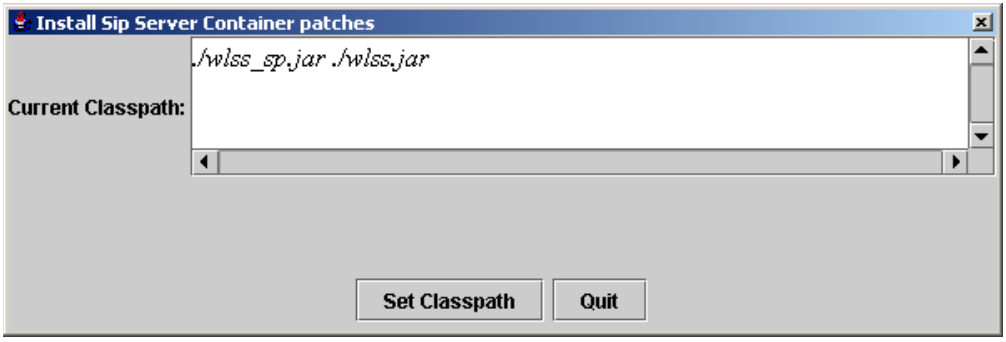
Editing the MANIFEST Classpath in GUI Mode

Running the `InstallPatch` utility in GUI mode enables you to reorder or delete existing patch files from the MANIFEST classpath. You invoke the utility in GUI mode using the command:

```
java com.bea.wcp.sip.tools.InstallPatch -mode gui
```

This yields a simple text editing window that shows the current classpath setting, as shown in Figure C-1.

Figure C-1 InstallPatch GUI Mode



To rearrange the order of JARs in the classpath, simply copy and paste the filenames in the desired order, keeping a space between multiple filenames. Click Set Classpath to persist the changes or Quit to exit without making changes.

Note: You can use GUI mode to add new patch JAR files to the existing classpath only if you first manually copy those files to the `APP-INF\container` subdirectory of the `sipserver` application.

Troubleshooting the InstallPatch Utility

If you patched WebLogic SIP Server but the patch does not seem to take effect, you may have un-patched JAR files listed earlier in your CLASSPATH. This can happen if you run the `setAdminClientEnv` script and `InstallPatch` utility in a shell that you previously used to start WebLogic SIP Server. To avoid this problem, always run `setAdminClientEnv` and `InstallPatch` in a dedicated shell that was not used to start servers.

If you cannot apply a patch, verify that the patch JAR file exists in the `APP-INF\container` subdirectory of the `sipserver` application. You must manually copy patch JAR files to this directory before running `InstallPatch`.

Note: If you are patching a replicated domain that uses "stage" mode deployment, you must manually delete the contents of the staging directory for each server (*`DOMAIN_DIR`*`/`*`servername`*`/stage` by default) to force redeployment of staged applications. Or, you must manually copy the newly-installed patch file(s) to the *`DOMAIN_DIR`*`/`*`servername`*`/stage/sipserver/APP-INF/container` directory of each Managed Server target.

Always verify the ordering of applied patches by using the `-action` command, or by running the utility in GUI mode. See "Syntax for Invoking the InstallPatch Utility" on page C-2.

APPENDIX D

Upgrading a WebLogic SIP Server 2.0.x Configuration to Version 2.2

The following sections provide instructions for upgrading WebLogic SIP Server from a previous release:

About the Upgrade Program

The WebLogic SIP Server upgrade program, `com.bea.wcp.sip.util.UpgradeConfig`, takes a `sipserver.xml` configuration file from a version 2.0.x WebLogic SIP Server release and recreates the configuration in WebLogic SIP Server 2.2 using the latest schemas. For example, `connector` entries from an earlier `sipserver.xml` file are converted into network channels in the WebLogic SIP Server 2.2 `config.xml` file.

In order to use the upgrade program, you must install WebLogic SIP Server 2.2 and create a new WebLogic SIP Server 2.2 domain. The newly domain configuration is then updated to match the earlier configuration using the `com.bea.wcp.sip.util.UpgradeConfig` program.

Steps for Upgrading an Existing Configuration

To upgrade a previous WebLogic SIP Server configuration to a new WebLogic SIP Server 2.2 configuration:

1. Install the WebLogic SIP Server software 2.2. See Installing WebLogic SIP Server Using Graphical-Mode Installation in *Installing WebLogic SIP Server*.
2. Use the Configuration Wizard to create a new **Basic WebLogic SIP Server Domain** on the Administration Server machine. See Using the Configuration Wizard in *Installing WebLogic SIP Server*.
3. Start the Administration Server for the WebLogic SIP Server 2.2 domain.
4. Set the environment required for using the `UpgradeConfig` utility. See "Required Environment for the UpgradeConfig Utility" on page D-2.
5. Use the `com.bea.wcp.sip.util.UpgradeConfig` utility to recreate your earlier WebLogic SIP Server configuration on the new WebLogic SIP Server 2.2 domain. See "UpgradeConfig Reference" on page D-2.

Required Environment for the UpgradeConfig Utility

To set up your environment for the `UpgradeConfig` utility:

1. Install the WebLogic SIP Server software 2.2. See Installing WebLogic SIP Server Using Graphical-Mode Installation in *Installing WebLogic SIP Server*.
2. Move to the top level of the WebLogic SIP Server 2.2 domain directory that you created:

   ```
   cd BEA_HOME\user_projects\domains\mydomain
   ```

   In the above command, *`BEA_HOME`* refers to the top-level BEA installation directory (for example, `c:\bea`).
3. Set the client environment using the command:

   ```
   setAdminClientEnv.cmd
   ```

UpgradeConfig Reference

The `UpgradeConfig` program uses the syntax:

```
java com.bea.wcp.sip.util.UpgradeConfig -username adminuser -password
adminpassword -adminurl url -sipserverconfigfile sipserver_old.xml
```

where:

- *adminuser* is the username of the WebLogic SIP Server 2.2 administrator
- *adminpassword* is the password of the WebLogic SIP Server 2.2 administrator
- *url* is the URL of the WebLogic SIP Server 2.2 Administration Server
- *sipserver_old.xml* is the full path to the `sipserver.xml` file from the earlier WebLogic SIP Server installation

For example:

```
java com.bea.wcp.sip.util.UpgradeConfig -username weblogic -password
weblogic -adminurl t3://localhost:7001 -sipserverconfigfile
c:\bea\user_projects\domains\wlss202_domain\sipserver.xml
```

The upgrade utility modifies the `sipserver.xml` and `config.xml` files in the WebLogic SIP Server 2.2 domain as necessary to match the earlier configuration.

Notes: The version 2.2 Administration Server must be running in order to upgrade the configuration.

`UpgradeConfig` can only update a WebLogic SIP Server 2.2 configuration from a single version 2.0.x `sipserver.xml` file. You cannot perform multiple upgrades against the same version 2.2 Administration Server using different `sipserver.xml` files.

APPENDIX E

Improving Failover Performance for Physical Network Failures

The following sections describe how to configure use the WebLogic SIP Server "echo server" process to improve data tier failover performance when a server becomes physically disconnected from the network:

Overview of Failover Detection

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

By default, an engine tier server uses its RMI connection to the replica to determine if the replica has failed or become disconnected. The algorithms used to determine a failure of an RMI connection are reliable, but they ultimately they depend on the TCP protocol's retransmission timers to diagnose a disconnection (for example, if the network cable to the replica is removed).

Because the TCP retransmission timer generally lasts a full minute or longer, WebLogic SIP Server provides an alternate method of detecting failures that can diagnose a disconnected replica in a matter of a few seconds.

WlssEchoServer Failure Detection

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

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

Also, should a data tier server notice that its local `WlssEchoServer` process has died, it automatically shuts down. This behavior ensures even quicker failover because avoids the time it takes engine nodes to notice and report the failure as described in "Overview of Failover Detection" on page E-1.

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

WlssEchoServer Requirements and Restrictions

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

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

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

Starting WlssEchoServer on Data Tier Server Machines

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

```
java -classpath WLSS_HOME/telco/lib/wlss.jar options
com.bea.wcp.util.WlssEchoServer
```

Where *`WLSS_HOME`* is the path to the WebLogic SIP Server installation and *`options`* may include one of the options described in Table E-1.

Table E-1 WlssEchoServer Options

Option	Description
`-Dwlss.ha.echoserver.port`	Specifies the port number used to listen for heartbeat messages. Ensure that the port number you specify is not used by any other process on the server machine. By default `WlssEchoServer` uses port 6734.
`-Dwlss.ha.echoserver.logfile`	Specifies the log file location and name. By default, log messages are written to `./echo_server`*`time`*`.log` where *`time`* is the time expressed in milliseconds.

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

```
"$JAVA_HOME/bin/java" ${JAVA_VM} ${MEM_ARGS} ${JAVA_OPTIONS}      \
```

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

to read:

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

Enabling and Configuring the Heartbeat Mechanism on Servers

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

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

to read:

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

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

Table E-2 WlssEchoServer Options

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

APPENDIX F

Tuning JVM Garbage Collection for Production Deployments

The following sections describe how to tune Java Virtual Machine (JVM) garbage collection performance for engine tier servers:

Goals for Tuning Garbage Collection Performance

Production installations of WebLogic SIP Server generally require extremely small response times (under 50 milliseconds) for clients at all times, even under peak server loads. A key factor in maintaining brief response times is the proper selection and tuning of the JVM's Garbage Collection (GC) algorithm for WebLogic SIP Server instances in the engine tier.

Whereas certain tuning strategies are designed to yield the lowest average garbage collection times or to minimize the frequency of full GCs, those strategies can sometimes result in one or more very long periods of garbage collection (often several seconds long) that are offset by shorter GC intervals. With a production SIP Server installation, all long GC intervals must be avoided in order to maintain response time goals.

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

Tuning Garbage Collection with JRockit

When using BEA's JRockit JVM, the best response time performance is generally obtained by using the single-spaced, concurrent garbage collector with a very small (1%) compaction rate. These settings can be obtained with the following startup options:

- `-Xgc:singlecon` specifies the use of the single-spaced, concurrent garbage collector
- `-XXcompactratio:1` specifies that only one percent of the heap is compacted after each garbage collection.

It is important to use the low compaction ratio setting along with the single-spaced GC algorithm to obtain the best response time performance. Note, however, that using the indicated compaction ratio may be problematic if you deploy applications that periodically allocate drastically different sizes of memory in the heap (for example a 10K allocation followed by byte-sized increments).

WARNING: If you deploy applications that allocate varying sizes of memory, using a small compaction ratio may lead to Out Of Memory errors or forced compaction, both of which must be avoided in a production system. You must thoroughly analyze deployed applications in a stage environment to determine which JVM settings are acceptable for your system.

JRockit provides several monitoring tools that you can use to analyze the JVM heap at any given moment, including:

- JRockit Runtime Analyzer—provides a view into the runtime behavior of garbage collection and pause times.
- JRockit Stack Dumps—reveals applications' thread activity to help you troubleshoot and/or improve performance.

Use these and other tools in a controlled environment to determine the effects of JVM settings before you use the settings in a production deployment. See the BEA WebLogic JRockit 1.4.2 SDK Documentation for more information about JRockit and JRockit profiling tools.

Tuning Garbage Collection with Sun JDK

When using Sun's JDK, the goal in tuning garbage collection performance is to reduce the time required to perform a full garbage collection cycle. You should not attempt to tune the JVM to minimize the frequency of full garbage collections, because this generally results in an eventual forced garbage collection cycle that may take up to several full seconds to complete.

The simplest and most reliable way to achieve short garbage collection times over the lifetime of a production server is to use a fixed heap size with the default collector and the parallel young generation collector, restricting the new generation size to at most one third of the overall heap. The following example JVM settings highlights the key garbage collection options used in this strategy:

```
-XX:+UseTLAB -XX:+UseParNewGC -Xms768m -Xmx768m -XX:NewSize=256m
-XX:MaxTenuringThreshold=0 -XX:SurvivorRatio=128
```

The above options have the following effect:

- `-XX:+UseTLAB`—Uses thread-local object allocation blocks. This improves concurrency by reducing contention on the shared heap lock.
- `-XX:+UseParNewGC`—Uses a parallel version of the young generation copying collector alongside the default collector. This minimizes pauses by using all available CPUs in parallel. The collector is compatible with both the default collector and the Concurrent Mark and Sweep (CMS) collector.
- `-Xms768m, -Xmx768m`—Fixes the heap size to increase the predictability of garbage collection. `-Xmx768m` limits the heap size so that even Full GCs do not trigger SIP retransmissions. `-Xms` sets the starting size to match to prevent pauses caused by heap expansion.
- `-XX:NewSize=256m`—Defines the minimum young generation size. BEA recommends testing your production applications starting with a young generation size of 1/3 the total heap size. Using a larger young generation size causes fewer minor collections to occur but may compromise response time goals by cause longer-running full collections.

  You can fine-tune the frequency of minor collections by gradually reducing the size of the heap allocated to the young generation to a point below which the observed response time becomes unacceptable.
- `-XX:MaxTenuringThreshold=0`—Makes the full NewSize available to every NewGC cycle, and reduces the pause time by not evaluating tenured objects. Technically, this setting promotes all live objects to the older generation, rather than copying them.
- `-XX:SurvivorRatio=128`—Specifies a high survivor ratio, which goes along with the zero tenuring threshold to ensure that little space is reserved for absent survivors.

Avoiding JVM Delays Caused by Random Number Generation

The library used for random number generation in Sun's JVM relies on `/dev/random` by default for UNIX platforms. This can potentially block the WebLogic SIP Server process because on some operating systems `/dev/random` waits for a certain amount of "noise" to be generated on the host machine before returning a result. Although `/dev/random` is more secure, BEA recommends using `/dev/urandom` if the default JVM configuration delays WebLogic SIP Server startup.

To determine if your operating system exhibits this behavior, try displaying a portion of the file from a shell prompt:

```
head -n 1 /dev/random
```

If the command returns immediately, you can use `/dev/random` as the default generator for SUN's JVM. If the command does not return immediately, use these steps to configure the JVM to use `/dev/urandom`:

1. Open the `$JAVA_HOME/jre/lib/security/java.security` file in a text editor.
2. Change the line:

   ```
   securerandom.source=/dev/random
   ```

   to read:

   ```
   securerandom.source=/dev/urandom
   ```

3. Save your change and exit the text editor.