BEA WebLogic SIP Server™

Configuring and Managing WebLogic SIP Server

Version 2.1
Revised: December 2, 2005

Copyright

Restricted Rights Legend

This software and documentation is subject to and made available only pursuant to the terms of the BEA Systems License Agreement and may be used or copied only in accordance with the terms of that agreement. It is against the law to copy the software except as specifically allowed in the agreement. This document may not, in whole or in part, be copied, photocopied, reproduced, translated, or reduced to any electronic medium or machine readable form without prior consent, in writing, from BEA Systems, Inc.

Use, duplication or disclosure by the U.S. Government is subject to restrictions set forth in the BEA Systems License Agreement and in subparagraph (c)(1) of the Commercial Computer Software-Restricted Rights Clause at FAR 52.227-19; subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013, subparagraph (d) of the Commercial Computer Software--Licensing clause at NASA FAR supplement 16-52.227-86; or their equivalent.

Information in this document is subject to change without notice and does not represent a commitment on the part of BEA Systems. THE SOFTWARE AND DOCUMENTATION ARE PROVIDED “AS IS” WITHOUT WARRANTY OF ANY KIND INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. FURTHER, BEA Systems DOES NOT WARRANT, GUARANTEE, OR MAKE ANY REPRESENTATIONS REGARDING THE USE, OR THE RESULTS OF THE USE, OF THE SOFTWARE OR WRITTEN MATERIAL IN TERMS OF CORRECTNESS, ACCURACY, RELIABILITY, OR OTHERWISE.

Trademarks or Service Marks

BEA, Jolt, Tuxedo, and WebLogic are registered trademarks of BEA Systems, Inc. BEA Builder, BEA Campaign Manager for WebLogic, BEA eLink, BEA Liquid Data for WebLogic, BEA Manager, BEA WebLogic Commerce Server, BEA WebLogic Enterprise, BEA WebLogic Enterprise Platform, BEA WebLogic Express, BEA WebLogic Integration, BEA WebLogic Personalization Server, BEA WebLogic Platform, BEA WebLogic Portal, BEA WebLogic Server, BEA WebLogic Workshop and How Business Becomes E-Business are trademarks of BEA Systems, Inc.

All other trademarks are the property of their respective companies.

Contents

1. Overview of the WebLogic SIP Server Architecture

2. Overview of WebLogic SIP Server Configuration and Management

3. Configuring Data Tier Partitions and Replicas

4. Configuring Engine Tier Container Properties

5. Capacity Planning for WebLogic SIP Server Deployments

6. Managing WebLogic SIP Server Network Resources

7. Production Network Architectures and WebLogic SIP Server Configuration

8. Overview of WebLogic SIP Server Security Features

9. Configuring Digest Authentication

13.Configuring SNMP

A. Upgrading Software and Applications in a Production Environment

B. Applying Patches Using InstallPatch

C. Upgrading a WebLogic SIP Server 2.0.x Configuration to Version 2.1

D. Engine Tier Configuration Reference (sipserver.xml)

E. Data Tier Configuration Reference (datatier.xml)

F. Tuning JVM Garbage Collection for Production Deployments

G. Avoiding JVM Delays Caused by Random Number Generation

CHAPTER 1

Overview of the WebLogic SIP Server Architecture

The following sections provide an overview of the WebLogic SIP Server 2.1 architecture:

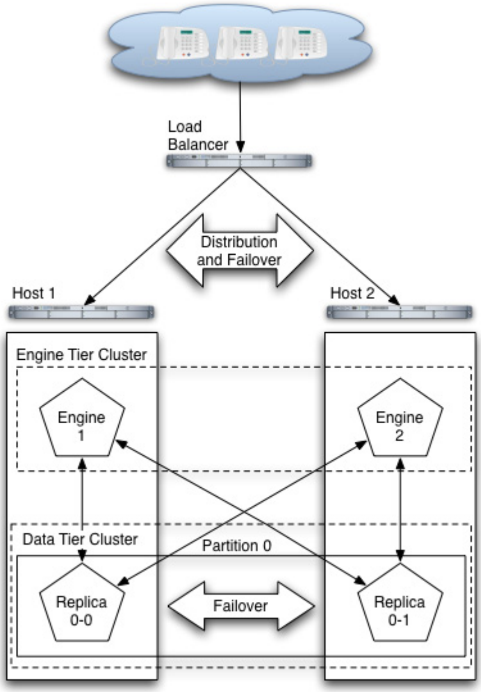
Goals of the WebLogic SIP Server Architecture

WebLogic SIP Server 2.1 is designed to provide a highly scalable, highly available, performant server for deploying SIP applications. The WebLogic SIP Server 2.1 architecture is simple to manage and easily adaptable to make use of available hardware. The basic architecture consists of several major components:

- Load Balancer
- Engine Tier
- Data tier

Figure 1-1shows 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 2.1 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 multiple 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 also provide special 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.

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 to monitor the health and availability of individual server instances in the WebLogic SIP Server engine tier for basic load distribution. See http://dev2dev.bea.com/. See also "Configuring Load Balancer Addresses" on page 6-2 to configure a load balancer used with WebLogic SIP Server and "Upgrading Software and Applications in a Production Environment" on page A-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. 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.

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.

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.

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 2.1, 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 5-2 for information about planning the hardware resources required in the data tier.

Example Hardware Configuration

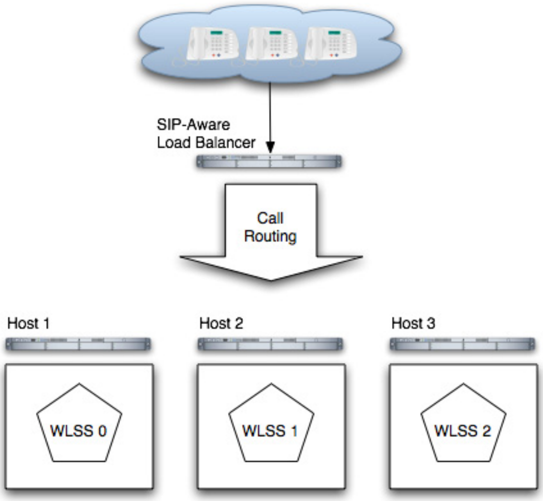
WebLogic SIP Server's flexible architecture enables you to configure engine and data tiers in a variety of ways to support high throughput and/or provide high availability. See "Capacity Planning for WebLogic SIP Server Deployments" on page 5-1 for detailed information about scaling the engine and data 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.

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" on page 2-1
- "Engine and Data Tier Configuration Overview" on page 2-2
- "Startup Sequence for a WebLogic SIP Server Domain" on page 2-4
- "Methods and Tools for Performing Configuration Tasks" on page 2-5
- "Administration Server Best Practices" on page 2-7
- "Common Configuration Tasks" on page 2-9

Shared Configuration Tasks for WebLogic SIP Server and WebLogic Server

WebLogic SIP Server 2.1 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 2.1, 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.

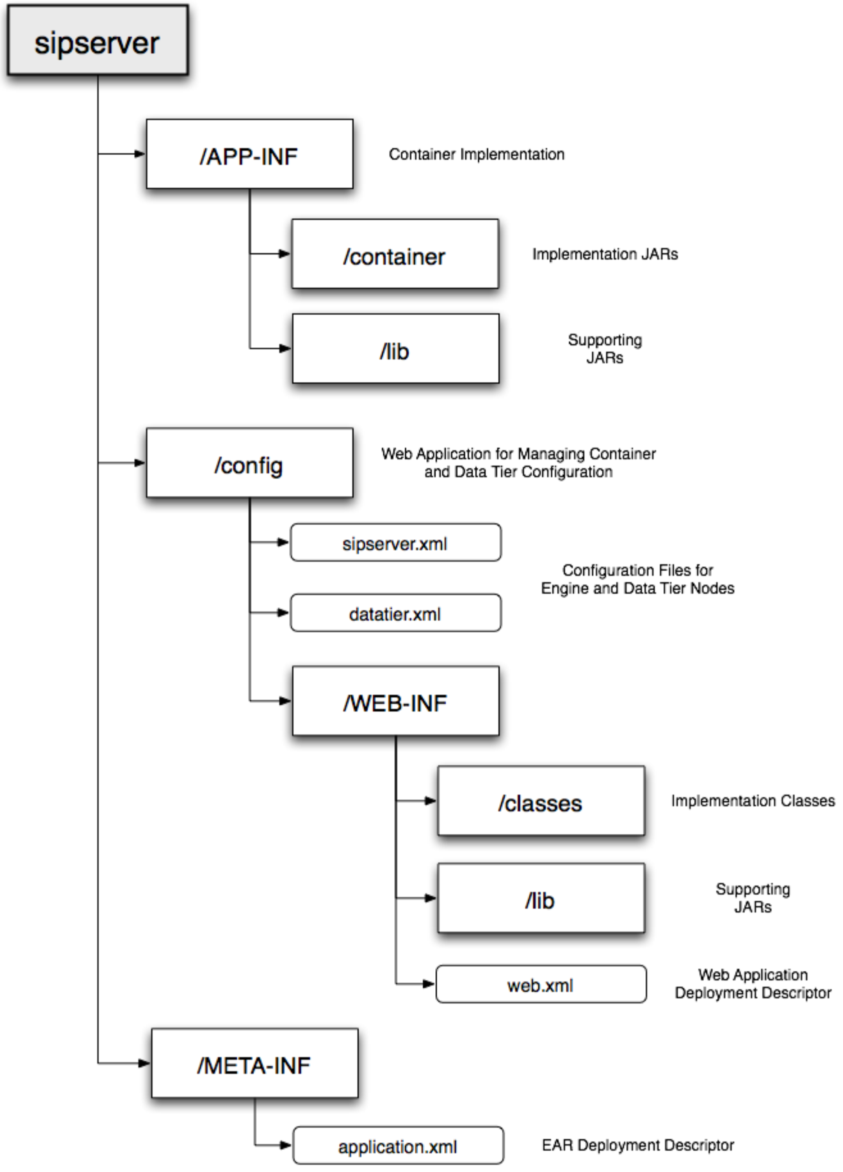
Engine and Data Tier Configuration Overview

The SIP Servlet container and call data replication features of WebLogic SIP Server are implemented in an Enterprise Application (EAR), named `sipserver`, that is deployed on WebLogic Server 8.1. The same `sipserver` application code is deployed to both engine and data tier instances of WebLogic SIP Server, but the XML configuration file(s) included in the `sipserver` application determine 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 maintained per domain, and changes to the configuration are made by modifying the configuration files contained in the `sipserver` application.

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.

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 configuration files 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)" on page D-1 and "Data Tier Configuration Reference (datatier.xml)" on page E-1.

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 2-6
- "Upgrade Utility" on page 2-6
- "WebLogic Scripting Tool (WLST)" on page 2-6

- "Additional Configuration Methods" on page 2-6

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 2.1 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 2.1" on page C-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)" on page D-1 and "Data Tier Configuration Reference (datatier.xml)" on page E-1.

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 6-1	• 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” on page 9-1	• Configuring the LDAP Digest Authentication Provider • Configuring a trusted host list
“Logging SIP Requests and Responses” on page 12-1	• Configuring logging Servlets to record SIP requests and responses. • Defining log criteria for filtering logged messages • Maintaining WebLogic SIP Server log files

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 5-1 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)" on page E-1` for full information about the XML Schema and elements.

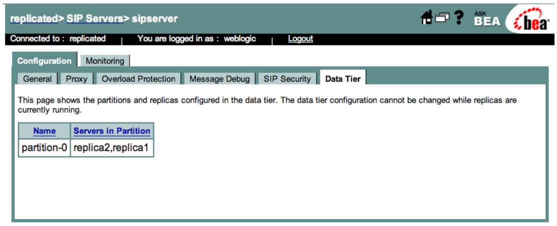
Configuration Requirements and Restrictions

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)



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

Name	Servers in Partition
partition-0	replica2,replica1

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 5-3, “Small Deployment with High Availability,” on page 5-8.

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 5-4, "Medium-Sized Deployment," on page 5-9 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>
```

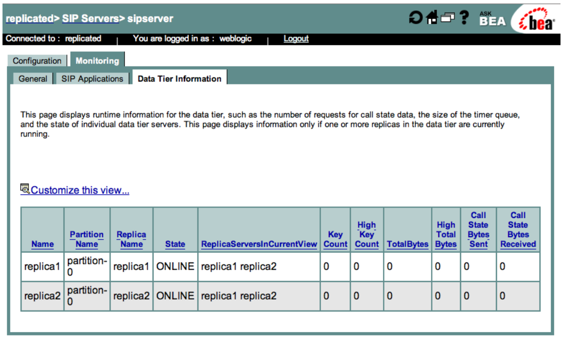
```
      <st:name>Partition0</st:name>
      <st:server-name>DataNode0-0</st:server-name>
      <st:server-name>DataNode0-1</st:server-name>
   </st:partition>
   <st:partition>
      <st:name>Partition1</st:name>
      <st:server-name>DataNode1-0</st:server-name>
      <st:server-name>DataNode1-1</st:server-name>
   </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 3-6.

Figure 3-2 Data Tier Monitoring in the Administration Console



This page displays runtime information for the data tier, such as the number of requests for call state data, the size of the timer queue, and the state of individual data tier servers. This page displays information only if one or more replicas in the data tier are currently running.

Name	Partition Name	Replica Name	State	ReplicaServersInCurrentView	Key Count	High Key Count	TotalBytes	High Total Bytes	Call State Bytes Sent	Call State Bytes Received
replica1	partition-0	replica1	ONLINE	replica1 replica2	0	0	0	0	0	0
replica2	partition-0	replica2	ONLINE	replica1 replica2	0	0	0	0	0	0

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 3-8 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.

Method/Attribute	Description
`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.
`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.

Method/Attribute	Description
`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.
`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 "Engine and Data Tier 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 Applications in a Production Environment" on page A-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

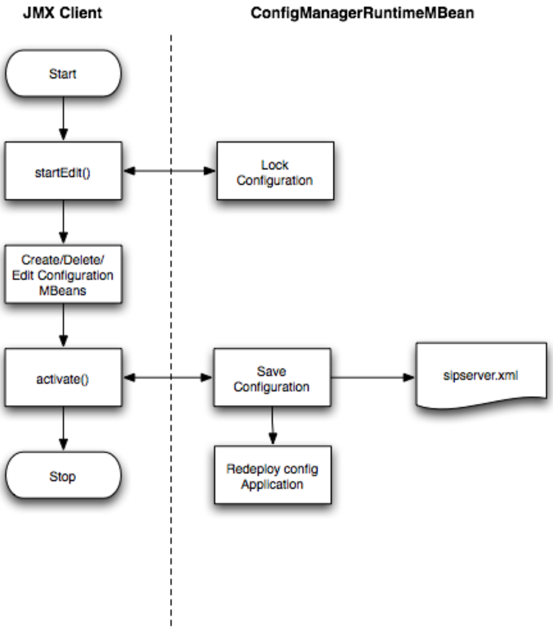
Method	Description
`activate()`	Writes the current configuration MBean attributes (the current SIP Servlet container configuration) to the `sipserver.xml` configuration file and applies changes to the running the `config` application.
`save()`	Writes the current configuration MBean attributes (the current 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


JMX Client
ConfigManagerRuntimeMBean
Start
startEdit()
Lock Configuration
Create/Delete/ Edit Configuration MBeans
activate()
Save Configuration
sipserver.xml
Stop
Redeploy config Application

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" on page D-16.
OverloadProtection	RegulationPolicy, ThresholdValue, ReleaseValue	Manages overload settings for throttling incoming SIP requests. See also "overload" on page D-8.
Proxy	ProxyURIs, RoutingPolicy	Manages the URIs routing policies for proxy servers. See also "proxy—Setting Up an Outbound Proxy Server" on page D-12.

MBean Type	MBean Attributes	Description
SipSecurity	TrustedAuthenticationHosts	Defines trusted hosts for which authentication is not performed. See also "sip-security" on page D-17.
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" on page D-17, "enable-local-dispatch" on page D-15, "max-application-session-lifetime" on page D-15, "t1-timeout-interval" on page D-14, "t2-timeout-interval" on page D-14, "t4-timeout-interval" on page D-14, "timerB-timeout-interval" on page D-15, and "timerF-timeout-interval" on page D-15.

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=Pr
oxy')

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\wlss210\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.

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.

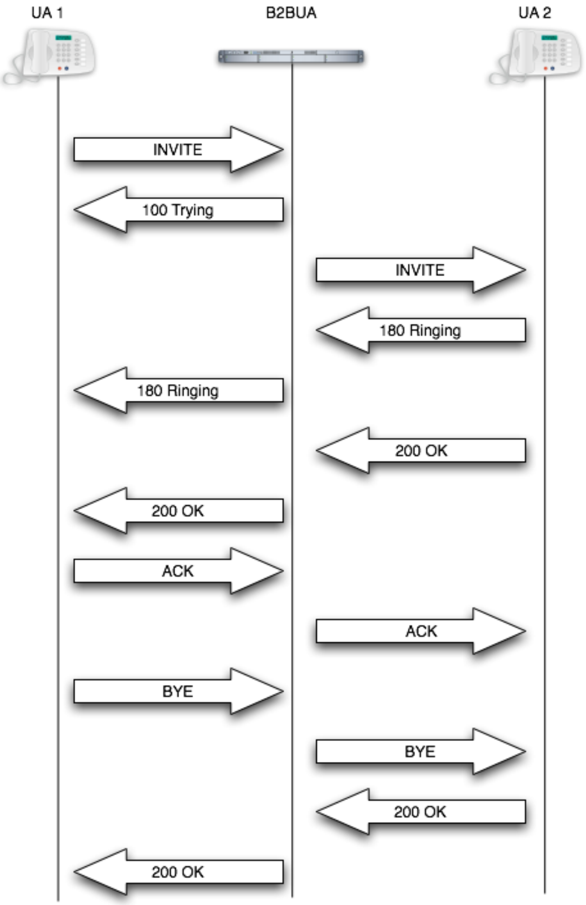
Determining Performance Goals

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 5-1. 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 5-1 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

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).

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.

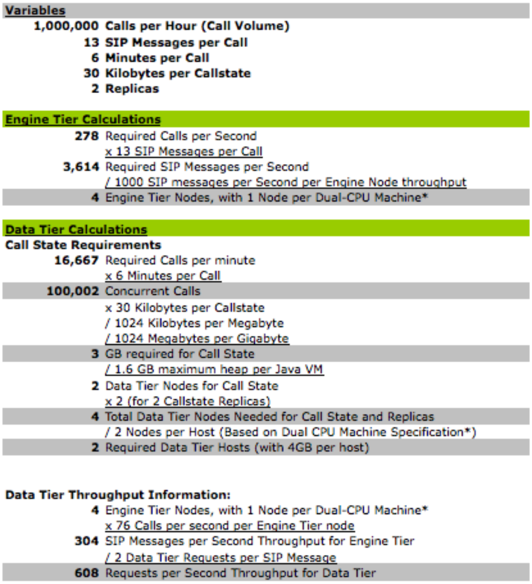
Throughput Values for WebLogic SIP Server Instances

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 5-1). 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 5-2, "Using Throughput Values to Determine Engine and Data Tier Requirements," on page 5-6 shows the sample calculations used to determine the medium-sized deployment described in "Medium Deployment" on page 5-8.

Figure 5-2 Using Throughput Values to Determine Engine and Data Tier Requirements



Variables	
1,000,000	**Calls per Hour (Call Volume)**
13	**SIP Messages per Call**
6	**Minutes per Call**
30	**Kilobytes per Callstate**
2	**Replicas**
Engine Tier Calculations	
278	Required Calls per Second
	x 13 SIP Messages per Call
3,614	Required SIP Messages per Second
	/ 1000 SIP messages per Second per Engine Node throughput
4	Engine Tier Nodes, with 1 Node per Dual-CPU Machine*
Data Tier Calculations	
Call State Requirements	
16,667	Required Calls per minute
	x 6 Minutes per Call
100,002	Concurrent Calls
	x 30 Kilobytes per Callstate
	/ 1024 Kilobytes per Megabyte
	/ 1024 Megabytes per Gigabyte
3	GB required for Call State
	/ 1.6 GB maximum heap per Java VM
2	Data Tier Nodes for Call State
	x 2 (for 2 Callstate Replicas)
4	Total Data Tier Nodes Needed for Call State and Replicas
	/ 2 Nodes per Host (Based on Dual CPU Machine Specification*)
2	Required Data Tier Hosts (with 4GB per host)
Data Tier Throughput Information:	
4	Engine Tier Nodes, with 1 Node per Dual-CPU Machine*
	x 76 Calls per second per Engine Tier node
304	SIP Messages per Second Throughput for Engine Tier
	/ 2 Data Tier Requests per SIP Message
608	Requests per Second Throughput for Data Tier

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:

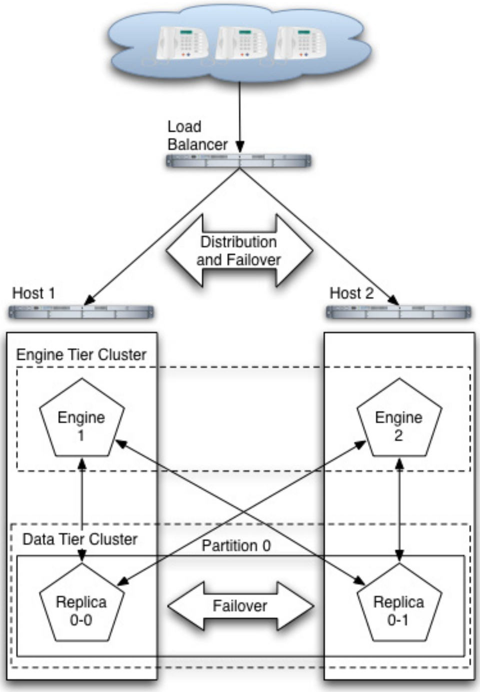
- Small Deployment describes a simple system that supports 500,000 calls per hour.
- Medium Deployment describes an average call system that utilizes multiple clustered servers to support 1 million calls per hour, with data replication to ensure high availability.
- Large Deployment describes a high 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 5-4. 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 5-3.

Note: The exact configuration shown in Figure 5-3 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.

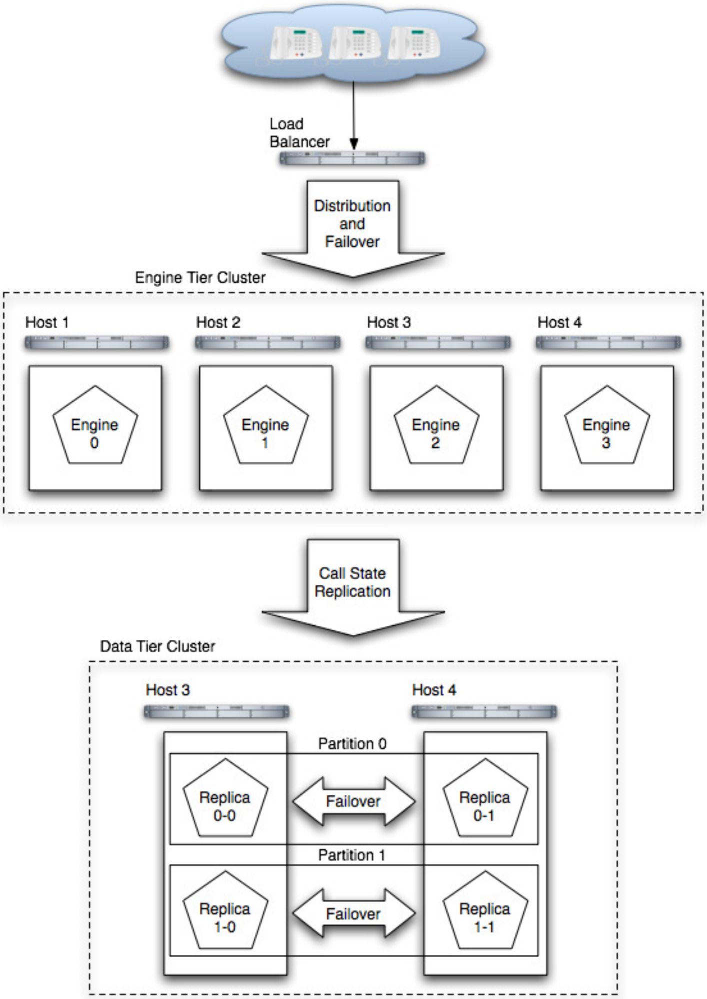
Figure 5-3 Small Deployment with High Availability



Medium Deployment

A typical medium deployment, shown in Figure 5-4, 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 5-4 Medium-Sized Deployment



Large Deployment

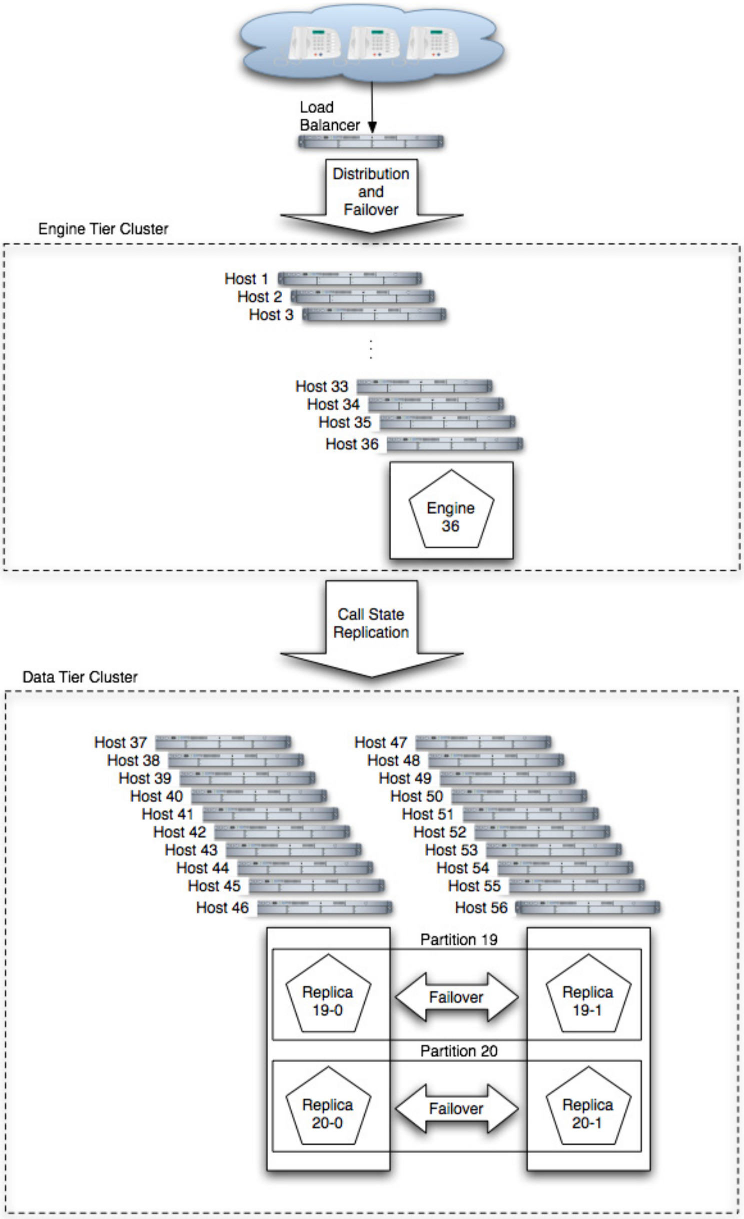
In the sample large-scale deployment, both the engine tier and data tier clusters have been expanded to support a call rate of 10 million calls per hour, as shown in Figure 5-5, "Large-Scale Deployment," on page 5-11.

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 5-5 Large-Scale Deployment



CHAPTER 6

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 6-4 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 6-4.

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 6-4 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 6-2.

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 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.

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

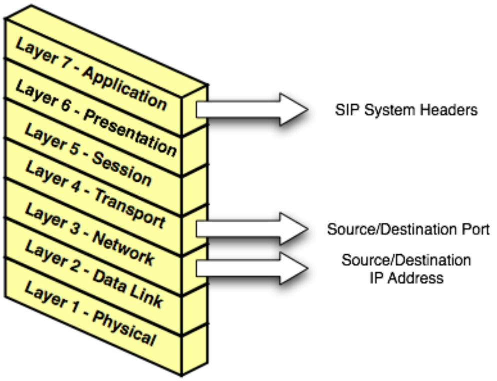
Most production installations of WebLogic SIP Server are contain one or more of the following characteristics:

- 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. XREF shows the typical Open Systems Interconnect (OSI) model layers that can be affected by different network configurations.

Figure 7-1 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 7-2 Single-NIC Network Channel Configuration

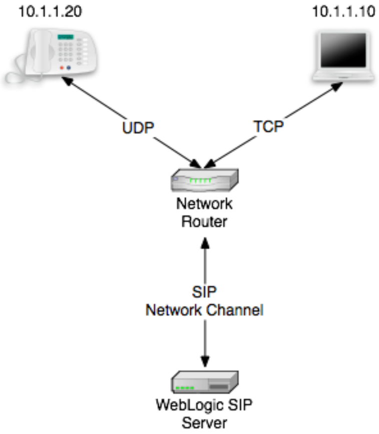


Figure 7-2 shows a single engine tier server instance with a single NIC. The server is configured with one network channel supporting SIP over UDP and TCP. (SIP channels always support both UDP and TCP transports; you cannot support only one of the two.) Figure 7-2 also shows two clients communicating 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 7-1 for an example configuration.

Listing 7-1 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 7-5.

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 7-6.

Multihomed Servers Listening On All Addresses (IP_ANY)

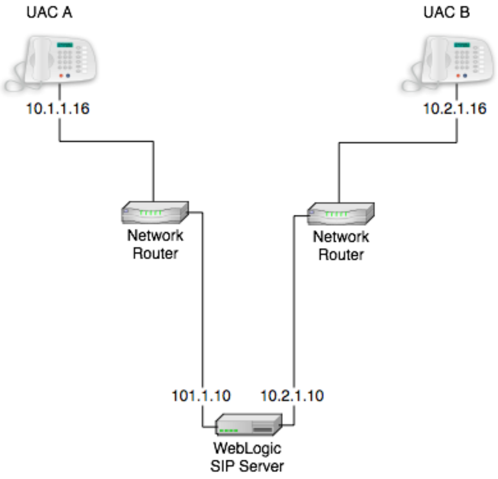
The simplest multihome configuration enables a WebLogic SIP Server instance to listen on all available NICs (physical NICs as well as logical NICs), sometimes described as IP_ANY. To accomplish this, you simply configure a single network channel and specify a channel listen address of 0.0.0.0.

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 6-6.

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 7-3 shows this configuration.

Figure 7-3 Multihomed Configuration for Proxying between Subnets



To configure the WebLogic SIP Server instance in Figure 7-3 you must define a separate network channel for each NIC used on the server machine. Listing 7-2 shows the `config.xml` entries that define channels for the sample configuration.

Listing 7-2 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 7-3, 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 7-3, 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 7-5.

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.

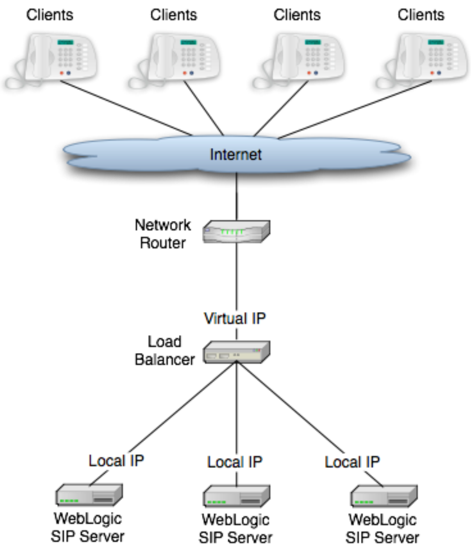
Load Balancer Configurations

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 7-4.

Figure 7-4 SIngle Load Balancer Configuration



To configure WebLogic SIP Server for use with a single load balancer as in Figure 7-4, 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 6-1 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 7-7.

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 8

Overview of WebLogic SIP Server Security Features

The following sections provide an overview of WebLogic SIP Server security:

- "Authentication for SIP Servlets" on page 8-1
- "Overriding Authentication with Trusted Hosts" on page 8-3
- "P-Asserted-Identity Support" on page 8-3
- "Role Assignment for SIP Servlet Declarative Security" on page 8-3
- "Security Event Auditing" on page 8-3
- "Common Security Configuration Tasks" on page 8-3

Authentication for SIP Servlets

WebLogic SIP Server users must be authenticated whenever they request access to a protected resource, such as a protected method within a deployed SIP Servlet. WebLogic SIP Server enables you to implement user authentication for SIP Servlets using any of the following techniques:

- **DIGEST authentication** uses a simple challenge-response mechanism to verify the identity of a user over SIP or HTTP. This technique is described in "Configuring Digest Authentication" on page 9-1.
- **CLIENT-CERT authentication** uses an X509 certificate chain passed to the SIP application to authenticate a user. The X509 certificate chain can be provided in a number of different ways. In the most common case, two-way SSL handshake is performed before

transmitting the chain to ensure secure communication between the client and server. CLIENT-CERT authentication is described fully in "Configuring Client-Cert Authentication" on page 10-1.

- **BASIC authentication** uses the `Authorization` SIP header to transmit the username and password to SIP Servlets. BASIC authentication is not recommended for production systems unless you can somehow ensure that all connections between clients and the WebLogic SIP Server instance are secure. This document does not provide configuration instructions for using BASIC authentication.

Different SIP Servlets deployed on WebLogic SIP Server can use different authentication mechanisms as necessary. The required authentication mechanism is specified in the `auth-method` element of the SIP Servlet's `sip.xml` deployment descriptor. The deployment descriptor may also define which resources are to be protected, listing specific role names that are required for access. See Securing SIP Servlet Resources in *Developing SIP Servlets with WebLogic SIP Server* for information about securing resources and mapping roles in the SIP Servlet deployment descriptor.

Authentication Providers

WebLogic SIP Server authentication services are implemented using one or more authentication providers. An authentication provider performs the work of proving the identity of a user or system process, and then transmitting the identity information to other components of the system.

You can configure and use multiple authentication providers to uses different authentication methods, or to work together to provide authentication. For example, when using Digest authentication you typically configure both a Digest Identity Asserter provider to assert the validity of a digest, and a second LDAP or RDBMS authentication provider that determines the group membership of a validated user.

When linking multiple authentication providers, you must specify the order in which providers are used to evaluate a given user, and also specify how much control each provider has over the authentication process. Each provider can contribute a "vote" that specifies whether or not the provider feels a given user is valid. The provider's control flag indicates how the provider's vote is used in the authentication process.

For more information about configuring providers, see either "Configuring Digest Authentication" on page 9-1 or "Configuring Client-Cert Authentication" on page 10-1.

Overriding Authentication with Trusted Hosts

WebLogic SIP Server also enables you to designate trusted hosts for your system. Trusted hosts are hosts for which WebLogic SIP Server performs no authentication. If the server receives a SIP message having a destination address that matches a configured trusted hostname, the message is delivered without Authentication. See "sip-security" on page D-17 for more information.

P-Asserted-Identity Support

WebLogic SIP Server supports the `P-Asserted-Identity` SIP header as described in RFC3325. This functionality automatically logs in using credentials specified in the `P-Asserted-Identity` header when they are received form a trusted host. When combined with the `privacy` header, `P-Asserted-Identity` also determines whether the message can be forwarded to trusted and non-trusted hosts. See "Configuring P-Asserted-Identity Assertion" on page 11-1 for more information.

Role Assignment for SIP Servlet Declarative Security

The SIP Servlet API specification defines a set of deployment descriptor elements that can be used for providing declarative and programmatic security for SIP Servlets. The primary method for declaring security constraints is to define one or more `security-constraint` elements and role definitions in the `sip.xml` deployment descriptor. WebLogic SIP Server adds additional deployment descriptor elements to help developers easily map SIP Servlet roles to actual principals and/or roles configured in the SIP Servlet container. See Securing SIP Servlet Resources in *Developing SIP Servlets with WebLogic SIP Server* for more information.

Security Event Auditing

WebLogic SIP Server includes an auditing provider that you can configure to monitor authentication events in the security realm. See Configuring a WebLogic Auditing Provider in the WebLogic Server 8.1 documentation for more information.

Common Security Configuration Tasks

Table 8-1 lists WebLogic SIP Server configuration tasks and provides links to additional information.

Table 8-1 Security Configuration Tasks

Task	Description
"Configuring Digest Authentication" on page 9-1	• Understanding the Digest identity assertion providers • Configuring LDAP Digest authentication • Configuring Digest authentication with an RDBMS
"Configuring Client-Cert Authentication" on page 10-1	• Understanding client-cert authentication solutions • Delivering X509 certificates over 2-way SSL • Developing a Perimeter authentication solution • Using the WebLogic SIP Server `WL_Client_Cert` header to deliver X509 certificates
"Configuring P-Asserted-Identity Assertion" on page 11-1	• Understand forwarding rules for SIP messages having the `P-Asserted-Identity` header • Configuring `P-Asserted-Identity` providers
Securing SIP Servlet Resources in *Developing SIP Servlets with WebLogic SIP Server*	• Defining security constraints for a SIP Servlet • Mapping SIP Servlet roles to WebLogic SIP Server roles and principals • Debugging SIP Servlet security constraints
"sip-security" on page D-17	• Configuring trusted hosts

Configuring Digest Authentication

The following sections describe how to configure WebLogic SIP Server to use Digest authentication with a supported LDAP server or RDBMS:

- "Overview of Digest Authentication" on page 9-1
- "Prerequisites for Configuring LDAP Digest Authentication" on page 9-6
- "Steps for Configuring LDAP Digest Authentication" on page 9-8
- "Sample Digest Authentication Configurations" on page 9-16

Overview of Digest Authentication

The following sections provide a basic overview of Digest authentication, and describe Digest authentication support and configuration in WebLogic SIP Server 2.1.

What Is Digest Authentication?

Digest authentication is a simple challenge-response mechanism used to authenticate a user over SIP or HTTP. Digest authentication is fully described in RFC 2617.

When using Digest authentication, if a client makes an un-authenticated request for a protected server resource, the server challenges the client using a nonce value. The client uses a requested algorithm (MD5 by default) to generate an encrypted response—a Digest—that includes a username, password, the nonce value from the challenge, the SIP method, and the requested URI.

The server verifies the client Digest by recreating the Digest value and comparing it with the client's Digest. To recreate the Digest value the server requires a hash of the "A1" value (see RFC 2617) that includes, at minimum, the nonce, username, password and realm name. The server either recreates the hash of the A1 value using a stored clear-text password for the user, or by obtaining a precalculated hash value. Either the clear-text password or precalculated hash value can be stored in an LDAP directory or accessed from an RDBMS using JDBC. The server then uses the hash of the A1 value to recreate the Digest and compare it to the client's Digest to verify the user's identity.

Digest authentication provides secure authorization over HTTP because the clear text password is never transmitted between the client and server. The use of nonce values in the client challenge also ensures that Digest authentication is resistant to replay attacks. See Figure 9-1 for a more detailed explanation of the challenge-response mechanism for a typical request.

Digest Authentication Support in WebLogic SIP Server 2.1

WebLogic SIP Server 2.1 includes LDAP Digest Identity Asserter security providers for asserting the validity of a client's Digest using LDAP or an RDBMS. A separate authorization provider is required to complete the authentication process (see "Configure an Authenticator Provider" on page 9-10).

The Digest Identity Asserter only verifies a user's credentials using the client Digest. After the Digest is verified, the configured authorization provider completes the authentication process by checking for the existence of the user (by username) and also populating group membership for the resulting `javax.security.auth.Subject`.

The Digest Identity Asserter provider requires that user credentials be stored in an LDAP server or RDBMS in one of the following ways:

- **Unencrypted (clear text) passwords.** The simplest configuration stores users' unencrypted passwords in a store. If you choose this method, BEA recommends using an SSL connection to the LDAP store or database to reduce the risk of exposing clear text passwords in server-side network traffic. Some LDAP stores do not support storing unencrypted passwords by default; in this case you must create or use a dedicated credential attribute on the LDAP server for storing the password. See "Configure the LDAP Server or RDBMS" on page 9-8.

- **A pre-calculated hash of each password, username, and realm.** If storing unencrypted passwords is unacceptable, you can instead store a pre-calculated hash value of the username, security-realm, and password in a new or existing attribute in LDAP or an RDBMS. The Digest Identity Asserter then retrieves only the hash value for comparison to the client-generated hash in the Digest. Storing pre-calculated hash values provides additional security.

The LDAP Digest Identity Asserter is compatible with any LDAP provider that permits storage of a clear text password or pre-calculated hash value.

Notes: You cannot change the schema for the built-in LDAP store to add a dedicated field for storing clear text passwords or pre-calculated hash values. However, you can use the predefined “description” field to store password information for testing or demonstration purposes.

If you do not use the DefaultAuthenticator provider for authentication decisions, you must make DefaultAuthenticator an optional provider (ControlFlag=“SUFFICIENT” or lower) before you can use Digest authentication. This will generally be the required configuration in production installations where a separate LDAP store is used to maintain clear text or hashed password information.

Figure 9-1 Digest Authentication in WebLogic SIP Server 2.1

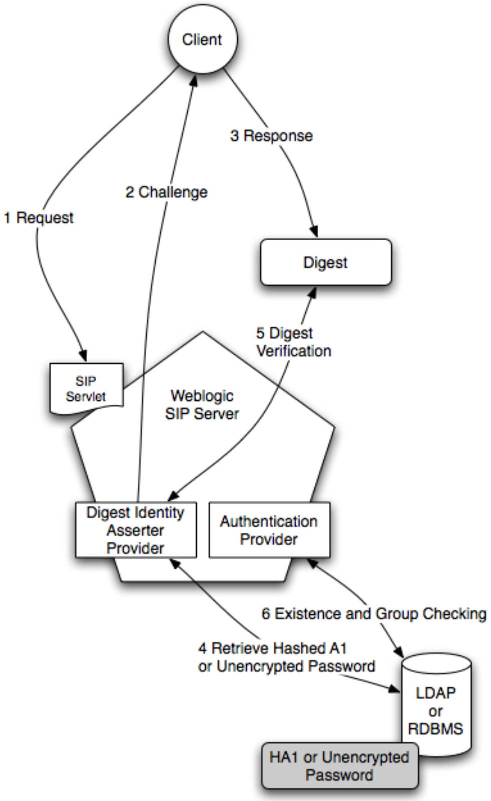

Client
3 Response
2 Challenge
1 Request
Digest
5 Digest Verification
SIP Servlet
Weblogic SIP Server
Digest Identity Asserter Provider
Authentication Provider
6 Existence and Group Checking
4 Retrieve Hashed A1 or Unencrypted Password
LDAP or RDBMS
HA1 or Unencrypted Password

Figure 9-1 shows the basic architecture and use of an Identity Asserter provider for a typical client request:

1. The client makes an unauthorized request for a protected application resource. (SIP Servlet resources can be protected by specifying security constraints in the `sip-xml` deployment descriptor. See Controlling Access to SIP Servlet Resources.)
2. The Digest Identity Asserter provider generates a challenge string consisting of the nonce value, realm name, and encryption algorithm (either MD5 or MD5-sess). The SIP container delivers the challenge string to the client.

Note: The Digest Identity Asserter maintains a cache of used nonces and timestamps for a specified period of time. All requests with a timestamp older than the specified timestamp are rejected, as well as any requests that use the same timestamp/nonce pair as the most recent timestamp/nonce pair still in the cache.

3. The client uses the encryption algorithm to create a Digest consisting of the username, password, real name, nonce, SIP method, request URI, and other information described in RFC 2617.
4. The Digest Identity Asserter verifies the client Digest by recreating the Digest value using a hash of the A1 value, nonce, SIP method, and other information. To obtain a hash of the A1 value, the Identity Asserter either generates HA1 by retrieving a clear-text password from the store, or the Identity Asserter retrieves the pre-calculated HA1 from the store.
5. The generated Digest string is compared to the client's Digest to verify the user's identity.
6. If the user's identity is verified, an authentication provider then determines if the user exists and if it does, the authentication provider populates the `javax.security.auth.Subject` with the configured group information. This step completes the authentication process.

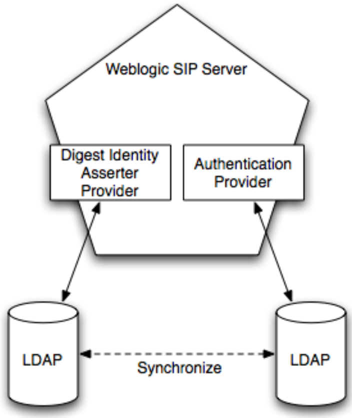
   Note: If you do not require user existence checking or group population, you can use the special "no-op" Identity Assertion Authenticator to avoid an extra connection to the LDAP Server; see "Configure an Authenticator Provider" on page 9-10 for more information.

   After authentication is complete, the SIP Servlet container performs an authorization check for the logged in `javax.security.auth.Subject` against the declarative security-constraints defined in the Servlet's `sip.xml` deployment descriptor.

The LDAP Digest Identity Asserter and the configured Authentication provider can either use the same LDAP store or different stores.

Note: If you use multiple LDAP stores, you must also create some infrastructure to keep both stores synchronized in response to adding, removing, or changing user credential changes, as shown in Figure 9-2. Maintaining LDAP stores in this manner is beyond the scope of this documentation.

Figure 9-2 Multiple LDAP Servers



Prerequisites for Configuring LDAP Digest Authentication

In order to configure Digest authentication you must understand the basics of LDAP servers and LDAP administration. You must also understand the requirements and restrictions of your selected LDAP server implementation, and have privileges to modify the LDAP configuration as well as the WebLogic SIP Server configuration.

Table 9-1 summarizes all of the information you will need in order to fully configure your LDAP server for Digest authentication with WebLogic SIP Server 2.1.

Note that the LDAP authentication provider and the Digest Authentication Identity Asserter provider can be configured with multiple LDAP servers to provide failover capabilities. If you want to use more than one LDAP server for failover, you will need to have connection information for each server when you configure Digest Authentication. See “Steps for Configuring LDAP Digest Authentication” on page 9-8.

Table 9-1 Digest Identity Asserter Checklist

Item	Description	Sample Value
Host	The host name of the LDAP server.	MyLDAPServer
Port	The port number of the LDAP server. Port 389 is used by default.	389
Principal	A Distinguished Name (DN) that WebLogic SIP Server can use to connect to the LDAP Server.	cn=ldapadminuser
Credential	A credential for the above principal name (generally a password).	ldapadminuserpassword
LDAP Connection Timeout	The configured timeout value for connections to the LDAP server (in seconds). For best performance, there should be no timeout value configured for the LDAP server. If a timeout value is specified for the LDAP server, you should configure the Digest Identity Asserter provider timeout to a value equal to or less than the LDAP server's timeout.	30 seconds
User From Name Filter	An LDAP search filter that WebLogic SIP Server will use to locate a given username. If you do not specify a value for this attribute, the server uses a default search filter based on the user schema.	(&(cn=%u)(objectclass=person))
User Base DN	The base Distinguished Name (DN) of the tree in the LDAP directory that contains users.	cn=users,dc=mycompany, dc=com
Credential Attribute Name	The credential attribute name used for Digest calculation. This corresponds to the attribute name used to store unencrypted passwords or pre-calculated hash values. See "Configure the LDAP Server or RDBMS" on page 9-8.	hashvalue
Digest Realm Name	The realm name to use for Digest authentication.	mycompany.com
Digest Algorithm	The algorithm that clients will use to create encrypted Digests. WebLogic SIP Server supports both MD5 and MD5-sess algorithms. MD5 is used by default.	MD5
Digest Timeout	The Digest authentication timeout setting. By default this value is set to 120 seconds.	120

Steps for Configuring LDAP Digest Authentication

Follow these steps to configure Digest authentication with WebLogic SIP Server 2.1:

1. "Configure the LDAP Server or RDBMS" on page 9-8.
2. "Reconfigure the DefaultAuthenticator Provider" on page 9-10.

   Note: DefaultAuthenticator is set up as a required authentication provider by default. If the DefaultAuthentication provider, which works against the embedded LDAP store, is not used for authentication decisions, you must change the Control Flag to "SUFFICIENT".

3. "Configure an Authenticator Provider" on page 9-10.
4. "Configure a New Digest Identity Asserter Provider" on page 9-11.

The sections that follow describe each step in detail.

Configure the LDAP Server or RDBMS

The LDAP server or RDBMS used for Digest verification must store either unencrypted, clear text passwords, pre-calculated hash values, or passwords encrypted by a standard encryption algorithm (3DES_EDE/CBC/PKCS5Padding by default). The sections below provide general information about setting up your LDAP server or RDBMS to store the required information. Keep in mind that LDAP server uses different schemas and different administration tools, and you may need to refer to your LDAP server documentation for information about how to perform the steps below.

If you are using multiple LDAP servers to enable failover capabilities for the security providers, you must configure each LDAP server as described below.

Using Unencrypted Passwords

If you are using an RDBMS, or if your LDAP server's schema allows storing unencrypted passwords in the user's password attribute, no additional configuration is needed. The Digest Identity Asserter provider looks for unencrypted passwords in the password field by default.

If the schema does not allow unencrypted passwords in the password attribute, you have two options:

- Store the unencrypted password in an existing, unused credential attribute in the LDAP directory.

- Create a new credential attribute to store the unencrypted password.

See your LDAP server documentation for more information about credential attributes available in the schema. Regardless of which method you use, record the exact attribute name used to store unencrypted passwords. You must enter the name of this attribute when configuring the LDAP Digest Identity Asserter provider.

Using Precalculated Hash Values

If you want to use precalculated hash values, rather than unencrypted passwords, you can store the hash values in one of two places in your LDAP directory:

- In an existing, unused credential attribute.
- In a new credential attribute that you create for the hash value.

See your LDAP server documentation for more information using or creating new credential attributes.

For RDBMS stores, you can place the hash values in any column in your schema; you will define the SQL command used to obtain the hash values when configuring the RDBMS Identity Assertion Provider.

WebLogic SIP Server provides a simple method call to generate a hash of the A1 value from a given username, realm name, and unencrypted password. The built-in method is available at `com.bea.wcp.sip.util.DigestUtils.getHA1("username", "realm-name", "password")`, and is packaged in the *WLSS_HOME*`\telco\lib\wlss.jar` file. You can use also use 3rd-party utilities for generating the hash value, or create your own method using information from RFC 2617.

Note that you must also create the necessary infrastructure to update the stored hash value automatically when the user name, password, or realm name values change. Maintaining the password information in this manner is beyond the scope of this documentation.

Using Reverse-Encrypted Passwords

WebLogic SIP Server provides a utility to help you compute the Encryption Key, Encryption Init Vector, and Encrypted Passwords values used when you configure the Digest Authorization Identity Asserter provider. The utility is named `com.bea.wcp.sip.security.utils.JSafeEncryptionServiceImpl` and is packaged in the `wlss.jar` file in the *WLSS_HOME*`/telco/lib` directory.

To view usage instructions and syntax:

1. Add `wlss.jar` to your classpath:

   ```
   set CLASSPATH=%CLASSPATH%;c:\bea\wlss210\telco\lib\wlss.jar
   ```

2. Execute the utility without specifying options:

   ```
   java com.bea.wcp.sip.security.utils.JsafeEncryptionServiceImpl
   ```

Reconfigure the DefaultAuthenticator Provider

In most production environments you will use a separate LDAP provider for storing password information, and therefore the DefaultAuthenticator, which works against the embedded LDAP store, must not be required for authentication. Follow the instructions in this section to change the provider's control flag to "sufficient".

> **Note:** DefaultAuthenticator is set up as a required authentication provider by default. If the DefaultAuthentication provider, which works against the embedded LDAP store, is not used for authentication decisions, you must change the Control Flag to "SUFFICIENT".

To reconfigure the DefaultAuthenticator provider:

1. Log in to the Administration Console for the WebLogic SIP Server domain you want to configure.
2. In the left pane of the Console, expand the Security->Realms->myrealm->Providers->Authentication node.
3. Select the DefaultAuthenticator node in the left pane.
4. In the General tab on the right pane, change the Control Flag value to SUFFICIENT.
5. Click Apply to apply your changes.
6. Reboot the server to realize the changed security configuration.

Configure an Authenticator Provider

In addition to the Digest Identity Asserter providers, which only validate the client digest, you must configure an "authentication" provider, which checks for a user's existence and populates the user's group information. Follow the instructions in Configuring an LDAP Authentication Provider in the WebLogic Server 8.1 SP5 documentation set to create an LDAP authentication provider for your LDAP server. Use the information from Table 9-1, "Digest Identity Asserter Checklist," on page 9-7 to configure the provider.

If you do not require user existence checking or group population, then, in addition to a Digest Identity Asserter provider, you can configure and use the special “no-op” authentication provider, packaged by the name “IdentityAssertionAuthenticator.” This provider is helpful to avoid an extra round-trip connection to the LDAP server. Note that the provider performs no user validation and should be used when group information is not required for users.

To configure the “no-op” authorization provider:

1. Log in to the Administration Console for the WebLogic SIP Server domain you want to configure.
2. In the left pane of the Console, expand the Security->Realms->myrealm->Providers->Authentication node.
3. Select the Authentication node in the left pane.
4. Click Configure a new Identity Assertion Authenticator...
5. Enter a name for the new provider, and set the Control Flag to SUFFICIENT.
6. Click Create to create the new provider.

Configure a New Digest Identity Asserter Provider

Follow these instructions in one of the sections below to create the Digest Identity Asserter provider and associate it with your LDAP server or RDBMS store:

- “Configure an LDAP Digest Identity Asserter Provider” on page 9-11
- “Configure an RDBMS Digest Identity Asserter Provider” on page 9-14

Configure an LDAP Digest Identity Asserter Provider

Follow these instructions to create a new LDAP Digest Identity Asserter Provider:

1. Log in to the Administration Console for the WebLogic SIP Server domain you want to configure.
2. In the left pane of the Console, expand the Security->Realms->myrealm->Providers->Authentication node.
3. Select the Authentication node in the left pane.
4. In the right pane of the Console, select Configure a new LDAP Digest Identity Asserter...

5. Enter a name for the new provider in the Name field, or accept the default, and click Create.
6. In the Active Types Chooser area, select both of the available types (WWW-Authenticate.DIGEST and Authorization.DIGEST) and use the arrow to move them to the Chosen column.
7. Click Apply to create the new provider.
8. Select the Details tab in the right pane to further configure the new provider.
9. In the Details tab, enter LDAP server and Digest authentication information into the fields as follows (use the information from Table 9-1):
   - **User From Name Filter**: Enter an LDAP search filter that WebLogic SIP Server will use to locate a given username. If you do not specify a value for this attribute, the server uses a default search filter based on the user schema.
   - **User Base DN**: Enter the base Distinguished Name (DN) of the tree in the LDAP directory that contains users.
   - **Credential Attribute Name**: Enter the credential attribute in the LDAP directory that stores either the pre-calculated hash value or the unencrypted password. By default WebLogic SIP Server uses the password attribute of the user entry. If you use a pre-calculated has value instead of an unencrypted password, or if the unencrypted password is stored in a different attribute, you must specify the correct attribute name here.
   - **Group Attribute Name**: Enter the group attribute in the LDAP directory that stores a the set of group names to which the user belongs.
   - **Password Encryption Type**: Select the format in which the password is stored: `PLAINTEXT`, `PRECALCULATEDHASH`, or `REVERSIBLEENCRYPTED`.
   - **Encryption Algorithm**: If you have stored encrypted passwords, enter the encryption algorithm that the Digest identity assertion provider will use for reverse encryption.
   - **Encryption Key** and **Confirm Encryption Key**: If you have stored encrypted passwords, enter the base-64 encrypted key used as part of the reverse encryption algorithm.
   - **Encryption Init Vector** and **Confirm Encryption Init Vector**: If you have stored encrypted passwords, enter the base-64 encrypted init vector string used as part of the reverse encryption algorithm.
   - **Digest Realm Name**: Enter the realm name to use for Digest authentication.

- **Digest Algorithm**: Select either MD5 or MD5-sess as the algorithm to use for encrypting Digests.
- **Digest Timeout**: This value defines the nonce timeout value for the digest challenge. If the nonce timeout is reached before the client responds, the client is re-challenged with a new nonce. By default, the Digest Timeout is set to 120 seconds.
- **LDAP Connection Pool Size**: Enter the number of connections to use for connecting to the LDAP Server. This value should be equal to or less than the total number of execute threads configured for WebLogic SIP Server. To view the current number of configured threads, right-click on the WebLogic SIP Server name in the left pane of the Administration Console and select View Execute Queues; the SIP Container uses the Thread Count value of the queue named sip.transport.Default. The default value of LDAP Connection Pool Size is 10.

  Note that stale connections (for example, LDAP connections that are timed out by a load balancer) are automatically removed from the connection pool.
- **Host**: Enter the host name of the LDAP server to use for Digest verification. If you are using multiple LDAP servers for failover capabilities, enter the *hostname:port* value for each server separated by spaces. For example: `ldap1.mycompany.com:1050 ldap2.mycompany.com:1050`

  See Configuring Failover for LDAP Authentication Providers in the WebLogic Server 8.1 SP5 documentation for more information about configuring failover.
- **Port**: Enter the port number of the LDAP server.
- **SSL Enabled**: Select this option if you are using SSL to communicate unencrypted passwords between WebLogic SIP Server and the LDAP Server.
- **Principal**: Enter the name of a principal that WebLogic SIP Server uses to access the LDAP server.
- **Credential**: Enter the credential for the above principal name (generally a password).
- **Confirm Credential**: Re-enter the principal's credential.
- **Cache Enabled**: Specifies whether a cache should be used with the associated LDAP server.
- **Cache Size**: Specifies the size of the cache, in Kilobytes, used to store results from the LDAP server. By default the cache size is 32K.
- **Cache TTL**: Specifies the time-to-live (TTL) value, in seconds, for the LDAP cache. By default the TTL value is 60 seconds.

- **Results Time Limit**: Specifies the number of milliseconds to wait for LDAP results before timing out. Accept the default value of 0 to specify no time limit.
- **Connect Timeout**: Specifies the number of milliseconds to wait for an LDAP connection to be established. If the time is exceeded, the connection times out. The default value of 0 specifies no timeout value.
- **Parallel Connect Delay**: Specifies the number of seconds to delay before making concurrent connections to multiple, configured LDAP servers. If this value is set to 0, the provider connects to multiple servers in a serial fashion. The provider first tries to connect to the first configured LDAP server in the Host list. If that connection attempt fails, the provider tries the next configured server, and so on.

  If this value is set to a non-zero value, the provider waits the specified number of seconds before spawning a new thread for an additional connection attempt. For example, if the value is set to 2, the provider first tries to connect to the first configured LDAP server in the Host list. After 2 seconds, if the connection has not yet been established, the provider spawns a new thread and tries to connect to the second server configured in the Host list, and so on for each configured LDAP server.
- **Connection Retry Limit**: Specifies the number of times the provider tries to reestablish a connection to an LDAP server if the LDAP server throws an exception while creating a connection.
- **Base64 Decoding Required**: This field is not applicable to the LDAP Digest Identity Asserter provider.

10. Click Apply to apply your changes.
11. Reboot the server to realize the changed security configuration.

Configure an RDBMS Digest Identity Asserter Provider

Follow these instructions to create a new RDBMS Digest Identity Asserter Provider:

1. Log in to the Administration Console for the WebLogic SIP Server domain you want to configure.
2. In the left pane of the Console, expand the Security->Realms->myrealm->Providers->Authentication node.
3. Select the Authentication node in the left pane.
4. In the right pane of the Console, select Configure a new DBMS Digest Identity Asserter...
5. Enter a name for the new provider in the Name field, or accept the default, and click Create.

6. In the Active Types Chooser area, select both of the available types (WWW-Authenticate.DIGEST and Authorization.DIGEST) and use the arrow to move them to the Chosen column.
7. Click Apply to create the new provider.
8. Select the Details tab in the right pane to further configure the new provider.
9. In the Details tab, enter RDBMS server and Digest authentication information into the fields as follows:
   - **Data Source Name**: Enter the name of the JDBC DataSource used to access the password information.
   - **SQLGet Users Password**: Enter the SQL statement used to obtain the password or hash value from the database. The SQL statement must return a single record result set.
   - **SQLList Member Groups**: Enter a SQL statement to obtain the group information from a specified username. The username is supplied as a variable to the SQL statement, as in `SELECT G_NAME FROM groupmembers WHERE G_MEMBER = ?`.
   - **Password Encryption Type**: Select the format in which the password is stored: `PLAINTEXT`, `PRECALCULATEDHASH`, or `REVERSIBLEENCRYPTED`.
   - **Encryption Algorithm**: If you have stored encrypted passwords, enter the encryption algorithm that the Digest identity assertion provider will use for reverse encryption.
   - **Encryption Key** and **Confirm Encryption Key**: If you have stored encrypted passwords, enter the base-64 encrypted key used as part of the reverse encryption algorithm.
   - **Encryption Init Vector** and **Confirm Encryption Init Vector**: If you have stored encrypted passwords, enter the base-64 encrypted init vector string used as part of the reverse encryption algorithm.
   - **Digest Realm Name**: Enter the realm name to use for Digest authentication.
   - **Digest Algorithm**: Select either MD5 or MD5-sess as the algorithm to use for encrypting Digests.
   - **Digest Timeout**: This value defines the nonce timeout value for the digest challenge. If the nonce timeout is reached before the client responds, the client is re-challenged with a new nonce. By default, the Digest Timeout is set to 120 seconds.
   - **Base64 Decoding Required**: This field is not applicable to the RDBMS Digest Identity Asserter provider.
10. Click Apply to apply your changes.

11. Reboot the server to realize the changed security configuration.

Sample Digest Authentication Configurations

After configuring Digest authentication using the preceding steps, you can verify the configuration by examining the `config.xml` file for your domain. The sections that follow show sample excerpts from `config.xml`.

The Administration Console automatically encrypts credential information stored in `config.xml`. If you are editing `config.xml` manually, you can use the `weblogic.management.EncryptionHelper` utility to encrypt the credentials as described in the WebLogic Server 8.1 documentation.

Oracle Internet Directory Server

Listing 9-1 shows the security provider configuration in `config.xml` for a domain that uses LDAP Digest authentication. Note that although the IPlanetAuthenticator provider was selected, the provider is configured to use an Oracle Internet Directory Server.

Listing 9-1 Sample Security Provider Configuration for Oracle

```
<weblogic.security.providers.authentication.IPlanetAuthenticator
   ControlFlag="SUFFICIENT"
   Credential="{3DES}uQtI9MFcc7yR9hqgx39J2g=="
   DisplayName="OIDAuthenticator"
   GroupBaseDN="cn=Groups,dc=bea,dc=com" Host="lcw2k18.bea.com"
   Name="Security:Name=myrealmOIDAuthenticator" Port="389"
   Principal="cn=orcladmin" Realm="Security:Name=myrealm"
   UserBaseDN="cn=users,dc=bea,dc=com"
   UserFromNameFilter="(&amp;(cn=%u)(objectclass=person))"/>

<com.bea.wcp.sip.security.authentication.LdapDigestIdentityAsserter
   ActiveTypes="Authorization.DIGEST|WWW-Authenticate.DIGEST"
   Credential="{3DES}uQtI9MFcc7yR9hqgx39J2g=="
```

```
CredentialAttributeName="middlename"

DigestRealmName="wcp.bea.com" Host="lcw2k18.bea.com"

Name="Security:Name=myrealmLdapDigestIdentityAsserter"

Principal="cn=orcladmin" Realm="Security:Name=myrealm"

UserBaseDN="cn=users,dc=bea,dc=com"

UserFromNameFilter="(&amp;(cn=%u)(objectclass=person))"/>
```

WebLogic SIP Server Embedded LDAP

You can use WebLogic SIP Server's embedded LDAP implementation to use Digest authentication in a test or demo environment. Because you cannot change the schema of the embedded LDAP store, you must store password information in the existing "description" field.

To use the embedded LDAP store for Digest authentication, follow the instructions in the sections that follow.

Store User Password Information in the Description Field

To create new users with password information in the existing "description" field:

1. Log in to the Administration Console for the WebLogic SIP Server domain you want to configure.
2. In the left pane of the Console, select the Security->Realms->myrealm->Users node.
3. Click Configure a new User...
4. Enter a name for the new user in the Name field.
5. Enter the Digest password information for the user in the Description field. The password information can be either the clear-text password, a pre-calculated hash value, or a reverse-encrypted password.
6. Enter an 8-character password in the Password and Confirm Password fields. You cannot proceed without adding a standard password entry.
7. Click Apply.

Set the Embedded LDAP Password

Follow these instructions to set the password for the embedded LDAP store to a known password. You will use this password when configuring the Digest Identity Asserter provider as described in "Configure an LDAP Digest Identity Asserter Provider" on page 9-11:

1. Log in to the Administration Console for the WebLogic SIP Server domain you want to configure.
2. In the left pane of the Console, select the Security node.
3. In the right pane, select Configuration->Embedded LDAP.
4. Enter the password you would like to use in the Credential and Confirm Credential fields.
5. Click Apply.
6. Reboot the server.

Configure the Digest Identity Asserter Provider

Listing 9-2 shows the security provider configuration in `config.xml` for a domain that uses LDAP implementation embedded in WebLogic SIP Server. Note that such a configuration is recommended only for testing or development purposes. Listing 9-2 highlights values that you must define when configuring the provider using the instructions in "Configure an LDAP Digest Identity Asserter Provider" on page 9-11.

Listing 9-2 Sample Security Provider Configuration with Embedded LDAP

```
<com.bea.wcp.sip.security.authentication.LdapDigestIdentityAsserter
   ActiveTypes="Authorization.DIGEST|WWW-Authenticate.DIGEST"
   Credential="{3DES}449oKgbalpo65cVYXzKhBg=="
   CredentialAttributeName="description"
   DigestRealmName="wcp.bea.com" Host="myserver.mycompany.com"
   Name="Security:Name=myrealmLdapDigestIdentityAsserter"
   Port="7001" Principal="cn=Admin"
   Realm="Security:Name=myrealm"
   UserBaseDN="ou=people, ou=myrealm, dc=mydomain" />
```

Configuring Client-Cert Authentication

The following sections describe how to configure WebLogic SIP Server to use Client-Cert authentication:

Overview of Client-Cert Authentication

Client-Cert authentication uses a certificate or other custom tokens in order to authenticate a user. The token is "mapped" to a user present in the WebLogic SIP Server security realm in which the Servlet is deployed. SIP Servlets that want to use Client-Cert authentication must set the `auth-method` element to `CLIENT-CERT` in their `sip.xml` deployment descriptor.

The token used for Client-Cert authentication can be obtained in several different ways:

- **X509 Certificate from SSL**—In the most common case, an X509 certificate is derived from a client token during a two-way SSL handshake between the client and the server. The SIP Servlet can view the resulting certificate in the `javax.servlet.request.X509Certificate` request attribute. This method for performing Client-Cert authentication is the most common and is described in the SIP Servlet specification (JSR-116). WebLogic SIP Server provides two security providers that can be used to validate the X509 certificate; see "Configuring SSL and X509 for WebLogic SIP Server" on page 10-2.

- **WL-Proxy-Client-Cert Header**—WebLogic SIP Server provides an alternate method for supplying a Client-Cert token that does not require a two-way SSL handshake between the client and server. Instead, the SSL handshake can be performed between a client and a proxy server or load balancer before reaching the destination WebLogic SIP Server. The proxy generates the resulting X509 certificate chain and encrypts it using base-64 encoding, and finally adds it to a special `WL-Proxy-Client-Cert` header in the SIP message. The server hosting the destination SIP Servlet then uses the `WL-Proxy-Client-Cert` header to obtain the certificate. The certificate is also made available by the container to Servlets via the `javax.servlet.request.X509Certificate` request attribute.

  To use this alternate method of supplying client tokens, you must configure WebLogic SIP Server to enable use of the `WL-Proxy-Client-Cert` header; see "Configuring WebLogic SIP Server to Use WL-Proxy-Client-Cert" on page 10-6. You must also configure an X509 Identity Asserter provider as described in "Configuring SSL and X509 for WebLogic SIP Server" on page 10-2.

SIP Servlets can also use the `CLIENT-CERT auth-method` to implement perimeter authentication. Perimeter authentication uses custom token names and values, along with a custom security provider, to authenticate clients. See "Supporting Perimeter Authentication with a Custom IA Provider" on page 10-7 for a summary of steps required to implement perimeter authentication.

Configuring SSL and X509 for WebLogic SIP Server

WebLogic SIP Server includes two separate Identity Assertion providers that can be used with X509 certificates. The LDAP X509 Identity Asserter provider receives an X509 certificate, looks up the LDAP object for the user associated with that certificate in a separate LDAP store, ensures that the certificate in the LDAP object matches the presented certificate, and then retrieves the name of the user from the LDAP object. The Default Identity Asserter provider maps the user according to its configuration, but does not validate the certificate.

With either provider, WebLogic SIP Server uses two-way SSL to verify the digital certificate supplied by the client. You must ensure that a SIPS transport (SSL) has been configured in order to use Client-Cert authentication. See "Managing WebLogic SIP Server Network Resources" on page 6-1 if you have not yet configured a secure transport.

See "Configuring the Default Identity Asserter" on page 10-3 to configure the Default Identity Asserter provider. In most production installations you will have a separate LDAP store and will need to configure the LDAP X509 Identity Asserter provider to use client-cert authentication; see "Configuring the LDAP X509 Identity Asserter" on page 10-4.

Configuring the Default Identity Asserter

The Default Identity Asserter can be configured to verify an X509 certificate passed to it by a client over a secure (SSL) connection. The Default Identity Asserter requires a separate user name mapper to map the associated client "certificate" to a user configured in the default security realm. You can use the default user name mapper installed with WebLogic SIP Server, or you can create a custom user name mapper class as described in Configuring a User Name Mapper in the WebLogic Server 8.1 Documentation.

Follow these instructions to configure the Default Identity Asserter:

1. Log in to the Administration Console for the WebLogic SIP Server domain you want to configure.
2. In the left pane of the Console, expand the Security->Realms->myrealm->Providers->Authentication node.
3. Select the Authentication node in the left pane.
4. In the right pane of the Console, select DefaultIdentityAsserter from the table of configured providers.
5. In the Types table, select X.509 and use the arrow to move this type to the Chosen column.
6. Select Base64 Decoding Required if the client token is being passed via two-way SSL or a `WL-Proxy-Client-Cert` header.
7. Click Apply to apply the change.
8. You can use either a custom Java class to map names in the X509 certificate to usernames in the built-in LDAP store, or you can use the default user name mapper. To specify a custom Java class to perform user name mapping:

    a. Enter the name of the custom class in the User Name Mapper Class Name field.

    b. Click Apply.

    To use the default user name mapper:

    a. Click the Details tab.

    b. Select Use Default User Name Mapper

c. In the Default User Name Mapper Attribute Type field, select either CN-Common Name or E-Email Address depending on the user name attribute you have stored in the security realm.

d. In the Default User Name Mapper Attribute Delimiter field, accept the default delimiter of “@”. This delimiter is used with the E-Email Address attribute type to extract the email portion from the client token. For example, a token of “joe@mycompany.com” would be mapped to a username “joe” configured in the default security realm.

e. Click Apply.

Configuring the LDAP X509 Identity Asserter

Follow these steps to create and configure the X509 Authentication Provider.

1. Log in to the Administration Console for the WebLogic SIP Server domain you want to configure.
2. In the left pane of the Console, expand the Security->Realms->myrealm->Providers->Authentication node.
3. Select the Authentication node in the left pane.
4. In the right pane of the Console, select Configure a new LDAP Digest p Asserter...
5. Enter a name for the new provider in the Name field, or accept the default, and click Create.
6. In the Active Types Chooser area, select X.509 and use the arrow to move this type to the Chosen column.
7. Click Apply to create the new provider.
8. Select the Details tab in the right pane to further configure the new provider.
9. In the Details tab, enter LDAP server information into the fields as follows:
   – **User Field Attributes**: Enter an LDAP search filter that WebLogic SIP Server will use to locate a given username. The filter is applied to LDAP objects beneath the base DN defined in the **Certificate Mapping** attribute described below.
   – **Username Attribute**: Enter the LDAP attribute that stores the user’s name.
   – **Certificate Attribute**: Enter the LDAP attribute that stores the certificate for the user name.

- **Certificate Mapping**: Specify how a query string to construct the base LDAP DN used to locate the LDAP object for the user.
- **Base64 Decoding Required**: Select this field if the client token is being passed via two-way SSL or a `WL-Proxy-Client-Cert` header.
- **Host**: Enter the host name of the LDAP server to verify the incoming certificate. If you are using multiple LDAP servers for failover capabilities, enter the *hostname:port* value for each server separated by spaces. For example: `ldap1.mycompany.com:1050 ldap2.mycompany.com:1050`

  See Configuring Failover for LDAP Authentication Providers in the WebLogic Server 8.1 SP5 documentation for more information about configuring failover.
- **Port**: Enter the port number of the LDAP server.
- **SSL Enabled**: Select this option if you are using SSL to communicate unencrypted passwords between WebLogic SIP Server and the LDAP Server.
- **Principal**: Enter the name of a principal that WebLogic SIP Server uses to access the LDAP server.
- **Credential**: Enter the credential for the above principal name (generally a password).
- **Confirm Credential**: Re-enter the principal's credential.
- **Cache Enabled**: Specifies whether a cache should be used with the associated LDAP server.
- **Cache Size**: Specifies the size of the cache, in Kilobytes, used to store results from the LDAP server. By default the cache size is 32K.
- **Cache TTL**: Specifies the time-to-live (TTL) value, in seconds, for the LDAP cache. By default the TTL value is 60 seconds.
- **Follow Referrals**: Select this to specify that a search for a user or group within the LDAP X509 Identity Assertion provider should follow referrals to other LDAP servers or branches within the LDAP directory.
- **Bind Anonymously On Referrals**: By default, the LDAP X509 Identity Assertion provider uses the same DN and password used to connect to the LDAP server when following referrals during a search. If you want to connect as an anonymous user, check this box.
- **Results Time Limit**: Specifies the number of milliseconds to wait for LDAP results before timing out. Accept the default value of 0 to specify no time limit.

- **Connect Timeout**: Specifies the number of milliseconds to wait for an LDAP connection to be established. If the time is exceeded, the connection times out. The default value of 0 specifies no timeout value.
- **Parallel Connect Delay**: Specifies the number of seconds to delay before making concurrent connections to multiple, configured LDAP servers. If this value is set to 0, the provider connects to multiple servers in a serial fashion. The provider first tries to connect to the first configured LDAP server in the Host list. If that connection attempt fails, the provider tries the next configured server, and so on.

  If this value is set to a non-zero value, the provider waits the specified number of seconds before spawning a new thread for an additional connection attempt. For example, if the value is set to 2, the provider first tries to connect to the first configured LDAP server in the Host list. After 2 seconds, if the connection has not yet been established, the provider spawns a new thread and tries to connect to the second server configured in the Host list, and so on for each configured LDAP server.
- **Connection Retry Limit**: Specifies the number of times the provider tries to reestablish a connection to an LDAP server if the LDAP server throws an exception while creating a connection.

10. Click Apply to apply your changes.

11. Reboot the server to realize the changed security configuration.

Configuring WebLogic SIP Server to Use WL-Proxy-Client-Cert

In order for WebLogic SIP Server to use the `WL-Proxy-Client-Cert` header, a proxy server or load balancer must first transmit the X509 certificate for a client request, encrypt it using base-64 encoding, and then add the resulting token `WL-Proxy-Client-Cert` header in the SIP message. If your system is configured in this way, you can enable the local WebLogic SIP Server instance (or individual SIP Servlet instances) to examine the `WL-Proxy-Client-Cert` header for client tokens.

To configure the server instance to use the `WL-Proxy-Client-Cert` header:

1. Log in to the Administration Console for the WebLogic SIP Server domain you want to configure.

2. In the left pane, expand the Servers node and select a server to configure. (Alternately, expand the Clusters node and select a cluster name to configure `WL-Proxy-Client-Cert` use for the entire cluster.)

3. Select the Configuration->General tab in the right pane.

4. Select Client Cert Proxy Enabled.
5. Click Apply to apply your changes.
6. Follow the instructions under "Configuring SSL and X509 for WebLogic SIP Server" on page 10-2 to configure either the default identity asserter or the LDAP Identity Asserter provider to manage X509 certificates. Select the Base64 Decoding Required option to decode the token passed in the WL-Proxy-Client-Cert header.
7. Reboot the server to realize the changed configuration.

To enable `WL-Proxy-Client-Cert` header for an individual Web Application, set the `com.bea.wcp.clientCertProxyEnabled` context parameter to true in the `sip.xml` deployment descriptor.

Supporting Perimeter Authentication with a Custom IA Provider

With perimeter authentication, a system outside of WebLogic Server establishes trust via tokens. The system is generally comprised of an authentication agent that creates an artifact or token that must be presented to determine information about the authenticated user at a later time. The actual format of the token varies from vendor to vendor (for example, SAML or SPNEGO).

WebLogic SIP Server supports perimeter authentication through the use of an Identity Assertion provider designed to recognize one or more token formats. When the authentication type of a SIP Servlet is set to `CLIENT-CERT`, the SIP container in WebLogic SIP Server performs identity assertion on values from the request headers. If the header name matches the active token type for a configured provider, the value is passed to the provider for identity assertion.

The provider can then use a user name mapper to resolve the certificate to a user available in the security realm. The user corresponding to the Subject's Distinguished Name (SubjectDN) attribute in the client's digital certificate must be defined in the server's security realm; otherwise the client will not be allowed to access a protected WebLogic resource.

If you want to use custom tokens to pass client certificates for perimeter authentication, you must create and configure a custom Identity Assertion provider in place of the LDAP X509 or Default Identity Asserter providers described above. See Identity Assertion Providers in *Developing Security Providers for WebLogic Server* (WebLogic Server 8.1 Documentation) for information about creating providers for handling tokens passed with perimeter authentication.

Configuring P-Asserted-Identity Assertion

The following sections describe how the `P-Asserted-Identity` and `privacy` headers affect forwarding to trusted and non-trusted hosts, and how to configure a WebLogic SIP Server P-Asserted-Identity Asserter provider:

- "Understanding Trusted Host Forwarding with P-Asserted-Identity" on page 11-1
- "Overview Strict and Non-Strict P-Asserted-Identity Asserter Providers" on page 11-2
- "Configuring a P-Asserted-Identity Assertion Provider" on page 11-3

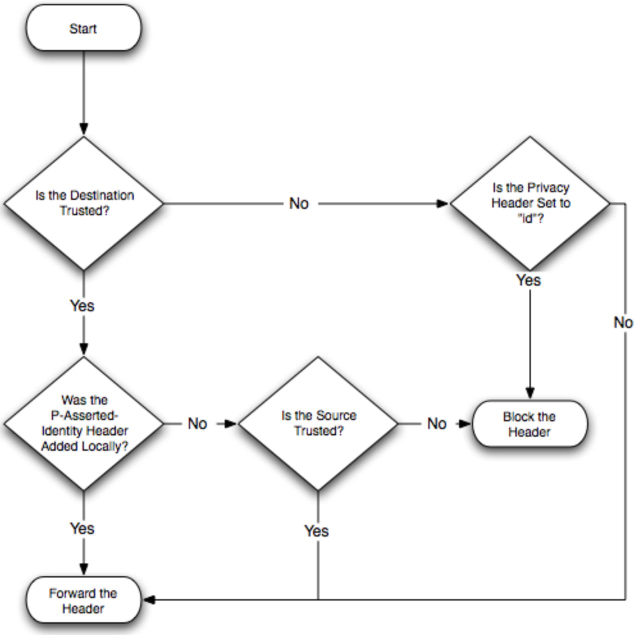
Understanding Trusted Host Forwarding with P-Asserted-Identity

WebLogic SIP Server supports the `P-Asserted-Identity` SIP header as described in RFC3325. To enable use of this header, you must configure one of two available P-Asserted Identity Assertion provider as described in "Configuring a P-Asserted-Identity Assertion Provider" on page 11-3.

When WebLogic SIP Server receives a message having the `P-Asserted-Identity` header from a trusted host configured with the provider, it logs in the user specified in the header to determine group membership and other privileges.

The presence of a `P-Asserted-Identity` header combined with the `Privacy` header also determines whether WebLogic SIP Server forwards a given message to trusted and non-trusted hosts. Figure 11-1 summarizes the forwarding restrictions with `P-Asserted-Identity`.

Figure 11-1 Forwarding Restrictions with P-Asserted-Identity and Privacy Headers


Start
Is the Destination Trusted?
No
Is the Privacy Header Set to "id"?
Yes
Yes
No
Was the P-Asserted-Identity Header Added Locally?
No
Is the Source Trusted?
No
Block the Header
Yes
Yes
Forward the Header

Overview Strict and Non-Strict P-Asserted-Identity Asserter Providers

If the contents of a P-Asserted-Identity header are invalid, or if the header is received from a non-trusted host, then the security provider returns an “anonymous” user to the SIP Servlet container. If you configured the **PAsserted Identity Strict Asserter** provider, an exception is also thrown so that you can audit the substitution of the anonymous user. (If you configured the basic **PAsserted Identity Asserter** provider, no exception is thrown.)

With either provider, if the requested resource is protected, the SIP container then uses the authentication method defined in the `auth-type` element in the Servlet’s `sip.xml` deployment descriptor to authorize the request. (For example, digest authentication may be used if the Servlet specifies the digest authentication method.)

If the requested resource is not protected, the anonymous user is simply passed to the SIP Servlet without authorization.

Configuring a P-Asserted-Identity Assertion Provider

Follow these steps to configure a security provider used to support the `P-Asserted-Identity` header. Note that one of two providers can be selected, as described in "Overview Strict and Non-Strict P-Asserted-Identity Asserter Providers" on page 11-2:

1. Log in to the Administration Console for the WebLogic SIP Server domain you want to configure.
2. In the left pane of the Console, expand the Security->Realms->myrealm->Providers->Authentication node.
3. Select the Authentication node in the left pane.
4. In the right pane of the Console, select one of the following options:
   - Configure a new PAsserted Identity Asserter...—Select this option to configure a provider that does not throw an exception when the `P-Asserted-Identity` header is invalid or is received from a non-trusted host and an anonymous user is substituted.
   - Configure a new PAsserted Identity Strict Asserter...—Select this option to configure a provider that throws an exception when the `P-Asserted-Identity` header is invalid or is received from a non-trusted host and an anonymous user is substituted.

   See "Overview Strict and Non-Strict P-Asserted-Identity Asserter Providers" on page 11-2 for more information.
5. Enter a name for the new provider and click Create.
6. Select the Details tab to display the new provider's configuration.
7. Fill in the fields of the Details tab as follows:
   - **Trusted Hosts**: Enter one or more host names that the provider will treat as trusted hosts. Note that the provider *does not use* trusted hosts configured in the `sipserver.xml` file (`"sip-security" on page D-17`.)
   - **User Name Mapper Class Name**: Enter the name of a custom Java class used to map user names in the `P-Asserted-Identity` header to user names in the default security realm. Or, leave this field blank to use the default user name mapper. See Configuring a User Name Mapper in the WebLogic Server 8.1 Documentation for more information.

   - **Base64Decoding Required**: This field is not used by the provider.

8. Click Apply.

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 12-2. 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 12-6.

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 12-15 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\wlss210\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 12-4 and "Specifying the Criteria for Logging Messages" on page 12-6 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\wlss210\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 12-4 to define a new logging Servlet in your existing application. Then read "Specifying the Criteria for Logging Messages" on page 12-6 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 12-1.

Listing 12-1 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 12-2.

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 12-1 lists the parameters and parameter values that you can specify as `init-param` entries. Listing 12-1, "Template Logging Servlets," on page 12-4 shows the sample `init-param` entries for a Servlet that logs full SIP message information to the domain log file.

Table 12-1 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 12-2 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 12-11. Most conditions, such as the `equal` condition shown in Listing 12-2, require a variable (`var` element) that identifies the portion of the SIP message to evaluate. Table 12-2 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 12-2 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 12-11 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 12-5.

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 12-7). 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 12-3 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 12-3 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>
```

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 12-4 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 instances 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 delegate in the Servlet's `init()` method as shown in Listing 12-5.

Listing 12-5 Using the TraceMessageListenerFactory

```
public void init() throws ServletException {
   ServletContext sc = (ServletContext) getServletContext();

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

   delegate = factory.createTraceMessageListener(getServletConfig());

}
```

CHAPTER 13

Configuring SNMP

The following sections describe how to configure and manage SNMP services with WebLogic SIP Server 2.1:

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 adds several traps and attributes in addition to those provided by the WebLogic Server 8.1 MIB. Engine and data tier traps are generated directly by the Managed Server instances that make up each tier.

The Administration Server may also generate traps on behalf of the WebLogic SIP Server domain as a whole, for example when a server in a cluster fails. WebLogic SIP Server MIB entries can also be monitored directly from the Administration Server of a 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 13-2.

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

The following sections provide detailed information about the following WebLogic SIP Server SNMP traps:

- "sipAppDeployed" on page 13-4
- "sipAppUndeployed" on page 13-4
- "sipAppFailedToDeploy" on page 13-5
- "overloadControlActivated, overloadControlDeactivated" on page 13-5
- "licenseLimitExceeded" on page 13-6
- "serverStopped" on page 13-8
- "dataTierServerStopped" on page 13-9
- "replicaAddedToPartition" on page 13-10
- "replicaRemovedFromPartition" on page 13-10
- "connectionLostToPeer" on page 13-10
- "connectionReestablishedToPeer" on page 13-11

sipAppDeployed

Description

WebLogic SIP Server generates 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 generates 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 generates 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.

overloadControlActivated, overloadControlDeactivated

Description

Weblogic SIP Server uses 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 *Configuring and Managing WebLogic SIP Server*.

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.)

licenseLimitExceeded

Description

WebLogic SIP Server generates 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="2.1">
        <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. Note that WebLogic SIP Server checks the license every four hours, so repairs may not be registered immediately.
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.

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. 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.

dataTierServerStopped

Description

WebLogic SIP Server generates this alarm when an unrecoverable error occurs in a WebLogic Server instance that is part of the data tier.

Recovery Procedure

See the Recovery Procedure for "serverStopped" on page 13-8.

replicaAddedToPartition

Description

WebLogic SIP Server generates 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

This trap is generated when a WebLogic SIP Server instance is removed from the data tier, either as a result of a normal shutdown operation or because of a failure.

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`.

connectionLostToPeer

Description

This trap is generated when an engine tier server instance 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 when an engine tier server 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 13-10.

APPENDIX A

Upgrading Software and Applications in a Production Environment

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

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

- "Overview of System and Application Upgrades" on page A-1
- "Requirements for Upgrading a Production System" on page A-2
- "Upgrading to a New Version of WebLogic SIP Server" on page A-3
- "Upgrading a Deployed Production Application (Compatible Session Data)" on page A-12
- "Upgrading a Deployed Production Application (Incompatible Session Data)" on page A-13

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 A-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 SIP 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 A-12.)

- Cluster-targeted deployments for all SIP 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:

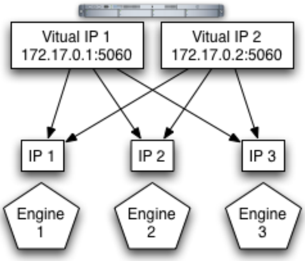
1. Configure the Load Balancer—Define a new, internal Virtual IP address for the new engine tier cluster you will configure.
2. Configure the New Engine Tier Cluster—Create and configure a new, empty engine tier cluster that will host upgraded engine tier servers and your production applications.
3. 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 A-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 A-1 shows a sample configuration for a cluster having three engine tier server instances; both virtual IP addresses define the same servers.

Figure A-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\wlss210\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 A-5.

Note: During the upgrade process you need to target the new sipserver Enterprise Application, as well as your own SIP 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 A-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 1-1 shows an entry for the sample clusters described earlier.

Listing 1-1 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 SIP 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 production SIP 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 A-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 SIP 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 A-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 A-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 A-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 A-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.

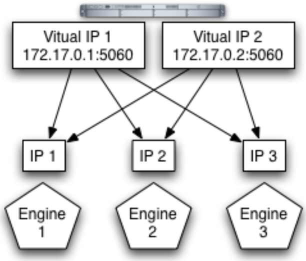
2. Configure the New Engine Tier Cluster—Create and configure a new, empty engine tier cluster that will host the new version of the SIP Servlet.
3. Define the Cluster-to-Load Balancer Mapping—Modify the SIP Servlet container configuration to indicate the virtual IP address of each engine tier cluster.
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 A-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 A-1 shows a sample configuration for a cluster having three engine tier server instances; both virtual IP addresses define the same servers.

Figure A-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\wlss210\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 A-5.

Note: During the upgrade process you need to target your new SIP 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 A-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 1-1 an entry for the sample clusters described earlier.

Listing 1-2 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 production 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 SIP 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 SIP 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.

Applying Patches Using InstallPatch

The following sections provide instructions for applying patches to WebLogic SIP Server instances:

- “Overview of the InstallPatch Utility” on page B-1
- “Required Environment for the InstallPatch Utility” on page B-2
- “Syntax for Invoking the InstallPatch Utility” on page B-2
- “Example InstallPatch Commands” on page B-3
- “Editing the MANIFEST Classpath in GUI Mode” on page B-4

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 2.1. See Installing WebLogic SIP Server Using Graphical-Mode Installation in *Installing WebLogic SIP Server*.
2. 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`).
3. 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\wlss210`).

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.

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 1-1 describes the arguments for the `com.bea.wcp.sip.tools.InstallPatch` utility.

Table 1-1 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 B-3 for more information. • `view`—Displays the current CLASSPATH.
`-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_wlss210.jar
```

This yields the classpath:

```
./CR567890_wlss210.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_wlss210.jar -patch CR567892_wlss210.jar
```

This yields the classpath:

```
./CR567890_wlss210.jar ./wlss_sp.jar ./wlss.jar ./CR567891_wlss210.jar
./CR567892_wlss210.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_wlss210.jar -patch wlss_sp.jar -patch wlss.jar -patch
CR567891_wlss210.jar
```

This yields the classpath:

```
./CR567890_wlss210.jar ./wlss_sp.jar ./wlss.jar ./CR567891_wlss210.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_wlss210.jar ./wlss_sp.jar
./wlss.jar ./CR567891_wlss210.jar
```

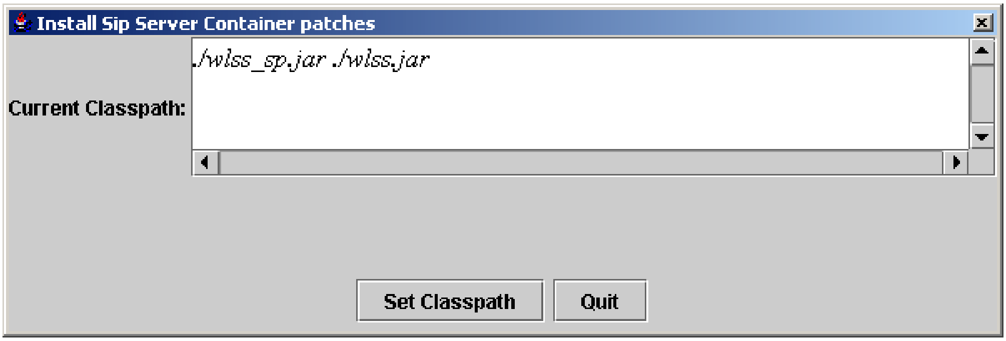
Editing the MANIFEST Classpath in GUI Mode

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 B-3.

Figure B-3 InstallPatch GUI Mode



Current Classpath: ./wlss_sp.jar ./wlss.jar

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.

Upgrading a WebLogic SIP Server 2.0.x Configuration to Version 2.1

The following sections provide instructions for upgrading WebLogic SIP Server from a previous release:

- "About the Upgrade Program" on page C-1
- "Steps for Upgrading an Existing Configuration" on page C-1
- "Required Environment for the UpgradeConfig Utility" on page C-2
- "wlss.UpgradeConfig Reference" on page C-2

About the Upgrade Program

The WebLogic SIP Server upgrade program, `wlss.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 2.1 using the latest schemas. For example, `connector` entries from an earlier `sipserver.xml` file are converted into network channels in the WebLogic SIP Server 2.1 `config.xml` file.

In order to use the upgrade program, you must install WebLogic SIP Server 2.1 and create a new WebLogic SIP Server 2.1 domain. The newly domain configuration is then updated to match the earlier configuration using the `wlss.UpgradeConfig` program.

Steps for Upgrading an Existing Configuration

To upgrade a previous WebLogic SIP Server configuration to a new WebLogic SIP Server 2.1 configuration:

1. Install the WebLogic SIP Server software 2.1. 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 2.1 domain.
4. Set the environment required for using the `UpgradeConfig` utility. See `"Required Environment for the UpgradeConfig Utility" on page C-2`.
5. Use the `wlss.UpgradeConfig` utility to recreate your earlier WebLogic SIP Server configuration on the new WebLogic SIP Server 2.1 domain. See "wlss.UpgradeConfig Reference" on page C-2.

Required Environment for the UpgradeConfig Utility

To set up your environment for the `UpgradeConfig` utility:

1. Install the WebLogic SIP Server software 2.1. 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 2.1 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
   ```

wlss.UpgradeConfig Reference

The `wlss.UpgradeConfig` program uses the syntax:

```
java wlss.UpgradeConfig -username adminuser -password adminpassword
-adminurl url -sipserverconfigfile sipserver_old.xml
```

where:

- *`adminuser`* is the username of the WebLogic SIP Server 2.1 administrator
- *`adminpassword`* is the password of the WebLogic SIP Server 2.1 administrator

- *url* is the URL of the WebLogic SIP Server 2.1 Administration Server
- *sipserver_old.xml* is the full path to the sipserver.xml file from the earlier WebLogic SIP Server installation

For example:

```
java wlss.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 2.1 domain as necessary to match the earlier configuration.

Notes: The version 2.1 Administration Server must be running in order to upgrade the configuration.

wlss.UpgradeConfig can only update a WebLogic SIP Server 2.1 configuration from a single version 2.0.x sipserver.xml file. You cannot perform multiple upgrades against the same version 2.1 Administration Server using different sipserver.xml files.

Engine Tier Configuration Reference (sipserver.xml)

The following sections provide a complete reference to the engine tier configuration file, `sipserver.xml`:

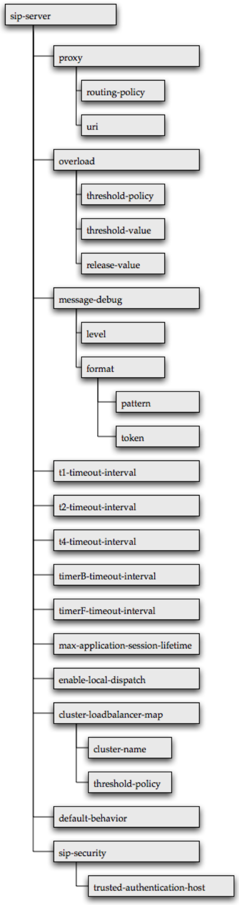
Overview of sipserver.xml

The sipserver.xml file is an XML document that configures the SIP container features provided by a WebLogic SIP Server instance in the engine tier of a server installation. sipserver.xml is stored in the subdirectory *DOMAIN_DIR*/sipserver/config/ where *DOMAIN_DIR* is the root directory of the WebLogic SIP Server domain.

Graphical Representation

Figure D-4 shows the element hierarchy of the sipserver.xml deployment descriptor file.

Figure D-4 Element Hierarchy of sipserver.xml



Editing sipserver.xml

You should never move, modify, or delete the `sipserver.xml` file during normal operations.

BEA recommends using the Administration Console to modify `sipserver.xml` indirectly, rather than editing the file. Using the Administration Console ensures that the `sipserver.xml` document always contains valid XML. See also "Configuring Container Properties Using WLST (JMX)" on page 4-4.

You may need to manually view or edit `sipserver.xml` to troubleshoot problem configurations, repair corrupted files, or to roll out custom configurations to large number machines when installing or upgrading WebLogic SIP Server. When you manually edit `sipserver.xml`, you must reboot WebLogic SIP Server instances to apply your 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.

Steps for Editing sipserver.xml

If you need to modify `sipserver.xml` on a production system, follow these steps:

1. Use a text editor to open the *`DOMAIN_DIR`*`/sipserver/config/sipserver.xml` file, where *`DOMAIN_DIR`* is the root directory of the WebLogic SIP Server domain.
2. Modify the `sipserver.xml` file as necessary. See "XML Schema" on page D-4 for a full description of the XML elements.
3. Save your changes and exit the text editor.
4. Reboot or start servers to have your changes take effect:

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.

5. Test the updated system to validate the configuration.

XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>

<xsd:schema targetNamespace="http://www.bea.com/ns/wlcp/wlss/210"

  xmlns:xsd="http://www.w3.org/2001/XMLSchema"

  xmlns:wlss="http://www.bea.com/ns/wlcp/wlss/210"

  elementFormDefault="qualified">
  <xsd:element name="sip-server">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="loadbalancer" type="wlss:loadBalancerType"
minOccurs="0" maxOccurs="1" nillable="true"/>
        <xsd:element name="proxy" type="wlss:proxyType" minOccurs="0"
maxOccurs="1" nillable="true"/>
        <xsd:element name="overload" type="wlss:overloadControlType"
minOccurs="0" maxOccurs="1" nillable="true"/>
        <xsd:element name="message-debug" type="wlss:messageDebugType"
minOccurs="0" maxOccurs="1" nillable="true"/>
        <xsd:element name="cluster-loadbalancer-map"
type="wlss:clusterLoadBalancerMapType" minOccurs="0" maxOccurs="unbounded"
nillable="true"/>
        <xsd:element name="sip-security" type="wlss:sipSecurityType"
minOccurs="0" maxOccurs="1"/>
        <xsd:element name="default-behavior" type="xsd:string"
default="proxy" minOccurs="0" />
        <xsd:element name="t1-timeout-interval" default="500"
type="xsd:unsignedLong" minOccurs="0"/>
        <xsd:element name="t2-timeout-interval" default="4000"
type="xsd:unsignedLong" minOccurs="0"/>
        <xsd:element name="t4-timeout-interval" default="5000"
type="xsd:unsignedLong" minOccurs="0" />
        <xsd:element name="timerB-timeout-interval" default="32000"
type="xsd:unsignedLong" minOccurs="0"/>
        <xsd:element name="timerF-timeout-interval" default="32000"
type="xsd:unsignedLong" minOccurs="0"/>
        <xsd:element name="max-application-session-lifetime" default="-1"
type="xsd:int" minOccurs="0"/>
        <xsd:element name="enable-local-dispatch" default="false"
type="xsd:boolean" minOccurs="0"/>
```

```
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:complexType name="loadBalancerType">
    <xsd:sequence>
      <xsd:element name="udp-uri" type="xsd:string" minOccurs="0"
maxOccurs="1"/>
      <xsd:element name="tcp-uri" type="xsd:string" minOccurs="0"
maxOccurs="1"/>
      <xsd:element name="tls-uri" type="xsd:string" minOccurs="0"
maxOccurs="1"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="overloadControlType">
    <xsd:sequence>
      <xsd:element name="threshold-policy" default="queue-length"
type="xsd:string"/>
      <xsd:element name="threshold-value" type="xsd:long"/>
      <xsd:element name="release-value" type="xsd:long"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="proxyType">
    <xsd:sequence>
      <xsd:element name="routing-policy" type="xsd:string"/>
      <xsd:element name="uri" type="xsd:string" minOccurs="1"
maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="messageDebugType">
```

```
    <xsd:sequence>
      <xsd:element name="level" type="xsd:string" default="full"/>
      <xsd:element name="format" type="wlss:formatType" minOccurs="0" 
maxOccurs="1"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="formatType">
    <xsd:sequence>
      <xsd:element name="pattern" type="xsd:string"/>
      <xsd:element name="token" type="xsd:string" minOccurs="1" 
maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="clusterLoadBalancerMapType">
    <xsd:sequence>
      <xsd:element name="cluster-name" type="xsd:string"/>
      <xsd:element name="sip-uri" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="sipSecurityType">
    <xsd:sequence>
      <xsd:element name="trusted-authentication-host"
                   type="xsd:string"
                   minOccurs="0"
                   maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

Example sipserver.xml File

The following shows a simple example of a `sipserver.xml` file:

```
<?xml version="1.0" encoding="UTF-8"?>

<sip-server xmlns="http://www.bea.com/ns/wlcp/wlss/210">

  <overload>

    <threshold-policy>queue-length</threshold-policy>

    <threshold-value>200</threshold-value>

    <release-value>150</release-value>

  </overload>

</sip-server>
```

XML Element Description

The following sections describe each element used in the sipserver.xml configuration file. Each section describes an XML element that is contained within the main `sip-server` element shown in Figure D-4.

overload

The `overload` element enables you to throttle incoming SIP requests according to a configured overload condition. When an overload condition occurs, WebLogic SIP Server destroys new SIP requests by responding with “503 Service Unavailable” until the configured release value is observed, or until the capacity of the server’s execute queues is exceeded (see XREF).

User-configured overload controls are applied only to initial SIP requests; SIP dialogues that are already active when an overload condition occurs may generate additional SIP requests that are not throttled.

To configure an overload control, you define the three elements described in Table 4-2.

Table 4-2 Nested overload Elements

Element	Description
`threshold-policy`	A String value that identifies the type of measurement used to monitor overload conditions: • `session-rate` measures the rate at which new SIP requests are generated. WebLogic SIP Server determines the session rate by calculating the number of new SIP application connections that were created in the last 5 seconds of operation. See "Overload Control Based on Session Generation Rate" on page D-11. • `queue-length` measures the sum of the sizes of the execute queues that processes SIP requests and SIP timers. See "Overload Control Based on Execute Queue Length" on page D-11. You must use only one of the above policies to define an overload control.

Element	Description
`threshold-value`	Specifies the measured value that causes WebLogic SIP Server to *start* throttling new SIP requests: • When using the `session-rate` threshold policy, `threshold-value` specifies the number of new SIP requests per second that trigger an overload condition. See "Overload Control Based on Session Generation Rate" on page D-11. • When using the `queue-length` threshold policy, `threshold-value` specifies the size of the combined SIP transport and SIP timer execute queue lengths that triggers an overload condition. See "Overload Control Based on Execute Queue Length" on page D-11. After the `threshold-value` is observed, WebLogic SIP Server throttles new SIP requests until the `release-value` value is observed.
`release-value`	Specifies the measured value that causes WebLogic SIP Server to *stop* throttling new SIP requests: • When using the `session-rate` threshold policy, `release-value` specifies the number of new SIP requests per second that terminates session throttling. See "Overload Control Based on Session Generation Rate" on page D-11. • When using the `queue-length` threshold policy, `release-value` specifies the combined execute queue length that terminates session throttling. See "Overload Control Based on Execute Queue Length" on page D-11.

WebLogic SIP Server provides two different policies for throttling SIP requests:

- The `session-rate` policy throttles sessions when the volume new SIP sessions reaches a configured rate (a specified number of sessions per second).
- The `queue-length` policy throttles requests after the sum of the sizes of the `sip.transport.Default` and `sip.timer.Default` execute queues reaches a configured size.

Note: You can throttle SIP requests either using `session-rate` policy or a `queue-length` policy. You cannot use both policies simultaneously.

The following sections describe each policy in detail.

Overload Control Based on Session Generation Rate

WebLogic SIP Server calculates the session generation rate (sessions per second) by monitoring the number of application sessions created in the last 5 seconds. When the session generation rate exceeds the rate specified in the threshold-value element, WebLogic SIP Server throttles initial SIP requests until the session generation rate becomes smaller than the configured release-value.

The following example configures WebLogic SIP Server to begin throttling SIP requests when the new sessions are created at a rate higher than 50 sessions per second. Throttling is discontinued when the session rate drops to 40 sessions per second:

```
<overload>

  <threshold-policy>session-rate</threshold-policy>

  <threshold-value>50</threshold-value>

  <release-value>40</release-value>

</overload>
```

Overload Control Based on Execute Queue Length

By default, SIP messages are handled by an execute queue named sip.transport.Default and SIP timers are processed by an execute queue named sip.timer.Default. These execute queues are configured in the config.xml file for your WebLogic SIP Server installation.

WebLogic SIP Server performs execute queue-based overload control by monitoring the combined lengths of these default execute queues. When the sum of the lengths of the two execute queues exceeds the length specified in the threshold-value element, WebLogic SIP Server throttles initial SIP requests until the total length is reduced to the configured release-value.

Listing 4-3 shows the default overload configuration from sipserver.xml. In the default configuration, WebLogic SIP Server begins throttling SIP requests when the combined size of the sip.transport.Default and sip.timer.Default queues exceeds 200 requests. Throttling is discontinued when the combined length returns to 200 or fewer simultaneous requests.

Listing 4-3 Sample overload Definition

```
<overload>

  <threshold-policy>queue-length</threshold-policy>
  <threshold-value>200</threshold-value>
  <release-value>150</release-value>
</overload>
```

Two Levels of Overload Protection

User-configured overload controls (defined in `sipserver.xml`) represent the first level of overload protection provided by WebLogic SIP Server. They mark the onset of an overload condition and initiate simple measures to avoid dropped calls (generating 503 responses for new requests).

If the condition that caused the overload persists or worsens, then the execute queues used to perform work in the SIP Servlet container may become full. At this point, the server can no longer generate 503 responses, so new message requests are simply dropped. In this way, the configured size of the SIP container's execute queues (`sip.transport.Default` and `sip.timer.Default`) represent the second and final level of overload protection employed by the server.

Always configure overload controls in `sipserver.xml` conservatively, and resolve the circumstances that caused the overload in a timely fashion. Overload conditions should never be permitted to last until the point where the execute queue capacities are exceeded.

message-debug

The `message-debug` element is used to enable and configure access logging for WebLogic SIP Server. This element should be used only in a development environment, because access logging logs *all* SIP requests and responses. See Enabling Access Logging in *Developing SIP Servlets with WebLogic SIP Server* for information about configuring and using access logging.

If you want to perform more selective logging in a production environment, see "Logging SIP Requests and Responses" on page 12-1.

proxy—Setting Up an Outbound Proxy Server

RFC 3261 defines an outbound proxy as "A proxy that receives requests from a client, even though it may not be the server resolved by the Request-URI. Typically, a UA is manually configured with an outbound proxy, or can learn about one through auto-configuration protocols."

In WebLogic SIP Server an outbound proxy server is specified using the `proxy` element in `sipserver.xml`. The proxy element defines one or more proxy server URIs. You can change the behavior of the proxy process by setting a proxy policy with the `proxy-policy` tag. Listing 4-3 describes the possible values for the `proxy` elements.

The default behavior is as if **domain** policy is in effect. The **proxy** policy means that the request is sent out to the configured outbound proxy and the route headers in the request preserve any routing decision taken by WebLogic SIP Server. This enables the outbound proxy to send the request over to the intended recipient after it has performed its actions on the request. The **proxy** policy comes into effect only for the initial requests. As for the subsequent request the Route Set takes precedence over any policy in a dialog. (If the outbound proxy wants to be in the Route Set it can turn record routing on).

Also if a proxy application written on WebLogic SIP Server wishes to override the configured behavior of outbound proxy traversal, then it can add a special header with name "X-BEA-Proxy-Policy" and value "domain". This header is stripped from the request while sending, but the effect is to ignore the configured outbound proxy. The X-BEA-Proxy-Policy custom header can be used by applications to override the configured policy on a request-by-request basis. The value of the header can be "domain" or "proxy". Note, however, that if the policy is overridden to "proxy," the configuration must still have the outbound proxy URIs in order to route to the outbound proxy.

Table 4-3 Nested proxy Elements

Element	Description
`routing-policy`	An optional element that configures the behavior of the proxy. Valid values are: • **domain** - Proxies messages using the routing rule defined by RFC 3261, ignoring any outbound proxy that is specified. • **proxy** - Sends the message to the downstream proxy specified in the default proxy URI. If there are multiple proxy specifications they are tried in the order in which they are specified. However, if the transport tries a UDP proxy, the settings for subsequent proxies are ignored.
`uri`	The TCP or UDP URI of the proxy server. You must specify at least one URI for a `proxy` element.

Listing 4-4 shows the default proxy configuration for WebLogic SIP Server domains. The request in this case is created in accordance with the SIP routing rules, and finally the request is sent to the outbound proxy "sipoutbound.bea.com".

Listing 4-4 Sample proxy Definition

```
<proxy>
     <routing-policy>proxy</routing-policy>
     <uri>sip:sipoutbound.bea.com:5060</uri>
     <!-- Other proxy uri tags can be added. -->
</proxy>
```

t1-timeout-interval

This element sets the value of the SIP protocol T1 timer, in milliseconds. Timer T1 also specifies the initial values of Timers A, E, and G, which control the retransmit interval for INVITE requests and responses over UDP.

Timer T1 also affects the values of timers F, H, and J, which control retransmit intervals for INVITE responses and requests; these timers are set to a value of 64*T1 milliseconds. See the *SIP: Session Initiation Protocol* for more information about SIP timers. See also "Configuring NTP for Accurate SIP Timers" on page 4-13.

If `t1-timeout-interval` is not configured, WebLogic SIP Server uses the SIP protocol default value of 500 milliseconds.

t2-timeout-interval

This elements sets the value of the SIP protocol T2 timer, in seconds. Timer T2 defines the retransmit interval for INVITE responses and non-INVITE requests. See the *SIP: Session Initiation Protocol* for more information about SIP timers. See also "Configuring NTP for Accurate SIP Timers" on page 4-13.

If `t2-timeout-interval` is not configured, WebLogic SIP Server uses the SIP protocol default value of 4 seconds.

t4-timeout-interval

This elements sets the value of the SIP protocol T4 timer, in seconds. Timer T4 specifies the maximum length of time that a message remains in the network. Timer T4 also specifies the initial values of Timers I and K, which control the wait times for retransmitting ACKs and responses over UDP. See the *SIP: Session Initiation Protocol* for more information about SIP timers. See also "Configuring NTP for Accurate SIP Timers" on page 4-13.

If `t4-timeout-interval` is not configured, WebLogic SIP Server uses the SIP protocol default value of 5 seconds.

timerB-timeout-interval

This elements sets the value of the SIP protocol Timer B, in milliseconds. Timer B specifies the length of time a client transaction attempts to retry sending a request. See the *SIP: Session Initiation Protocol* specification for more information about SIP timers. See also "Configuring NTP for Accurate SIP Timers" on page 4-13.

If `timerB-timeout-interval` is not configured, the Timer B value is derived from timer T1 (64*T1, or 32000 milliseconds by default).

timerF-timeout-interval

This elements sets the value of the SIP protocol Timer F, in milliseconds. Timer F specifies the timeout interval for retransmitting non-INVITE requests. See the *SIP: Session Initiation Protocol* specification for more information about SIP timers. See also "Configuring NTP for Accurate SIP Timers" on page 4-13.

If `timerF-timeout-interval` is not configured, the Timer F value is derived from timer T1 (64*T1, or 32000 milliseconds by default).

max-application-session-lifetime

This element sets the maximum amount of time, in minutes, that a SIP application session can exist before WebLogic SIP Server invalidates the session. By default there is no limit for SIP session lifetimes.

enable-local-dispatch

`enable-local-dispatch` is a server optimization that helps avoid unnecessary network traffic when sending and forwarding messages. You enable the optimization by setting this element "true." When `enable-local-dispatch` enabled, if a server instance needs to send or forward a message and the message destination is the engine tier's cluster address or the local server address, then the message is routed internally to the local server instead of being sent via the network. Using this optimization can dramatically improve performance when chained applications process the same request as described in Composing SIP Applications in *Developing SIP Servlets with WebLogic SIP Server*.

You may want to disable this optimization if you feel that routing internal messages could skew the load on servers in the engine tier, and you prefer to route all requests via a configured load balancer.

By default `enable-local-dispatch` is set to “false.”

cluster-loadbalancer-map

The `cluster-loadbalancer-map` element is used only when upgrading WebLogic SIP Server software, or when upgrading a production SIP Servlet to a new version. It is not required or used during normal server operations.

During a software upgrade, multiple engine tier clusters are defined to host the older and newer software versions. `A cluster-loadbalancer-map` defines the virtual IP address (defined on your load balancer) that correspond to an engine tier cluster configured for an upgrade. WebLogic SIP Server uses this mapping to ensure that engine tier requests for timers and call state data are received from the correct “version” of the cluster. If a request comes from an incorrect version of the software, the `cluster-loadbalancer-map` entries are used to forward the request to the correct cluster.

Each `cluster-loadbalancer-map` entry contains the two elements described in

Table 4-4 Nested cluster-loadbalancer-map Elements

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

Listing 4-5 shows a sample `cluster-loadbalancer-map` entry used during an upgrade.

Listing 4-5 Sample cluster-loadbalancer-map Entry

```
<cluster-loadbalancer-map>
   <cluster-name>EngineCluster</cluster-name>
   <sip-uri>sip:172.17.0.1:5060</sip-uri>
</cluster-loadbalancer-map>
<cluster-loadbalancer-map>
   <cluster-name>EngineCluster2</cluster-name>
   <sip-uri>sip:172.17.0.2:5060</sip-uri>
</cluster-loadbalancer-map>
```

default-behavior

This element defines the default behavior of the WebLogic SIP Server instance if the server cannot match an incoming SIP request to a deployed SIP Servlet (or if the matching application has been invalidated or timed out). Valid values are:

- `proxy`—Act as a stateless proxy server.
- `ua`—Act as a User Agent.

`proxy` is used as the default if you do not specify a value.

When acting as a User Agent (UA), WebLogic SIP Server acts in the following way in response to SIP requests:

- ACK requests are discarded without notice.
- CANCEL or BYE requests receive response code 481 - Transaction does not exist.
- All other requests receive response code 500 - Internal server error.

When acting as a stateless proxy requests are automatically forwarded to an outbound proxy (see "proxy—Setting Up an Outbound Proxy Server" on page D-12) if one is configured. If no proxy is defined, Weblogic SIP Server proxies to a specified Request URI only if the Request URI does not match the IP and port number of a known local address for a SIP Servlet container, or a load balancer address configured for the server. This ensures that the request does not constantly loop to the same servers. When the Request URI matches a local container address or load balancer address, WebLogic SIP Server instead acts as a UA.

sip-security

WebLogic SIP Server enables you to configure one or more trusted hosts for which authentication is not performed. When WebLogic SIP Server receives a SIP message, it calls `getRemoteAddress()`

on the SIP Servlet message. If this address matches an address defined in the server's trusted host list, no further authentication is performed for the message.

The sip-security element defines one or more trusted hosts, for which authentication is not performed. The sip-security element contains one or more trusted-authentication-host elements, each of which contains a trusted host definition. A trusted host definition can consist of an IP address (with or without wildcard placeholders) or a DNS name. Listing 4-6 shows a sample sip-security configuration.

Listing 4-6 Sample Trusted Host Configuration

```
<sip-security>

   <trusted-authentication-host>myhost1.mycompany.com</trusted-authenticat
ion-host>

   <trusted-authentication-host>172.*</trusted-authentication-host>

</sip-security>
```

Data Tier Configuration Reference (datatier.xml)

The following sections provide a complete reference to the data tier configuration file, `datatier.xml`:

Overview of datatier.xml

The `datatier.xml` configuration file identifies servers that manage the concurrent call state for SIP applications, and defines how those servers are arranged into data tier *partitions*. A *partition* refers to one or more data tier server instances that manage the same portion of the call state. Multiple servers in the same partition are referred to as *replicas* because they all manage a copy of the same portion of the call state.

`datatier.xml` is stored in the subdirectory *`DOMAIN_DIR`*`/sipserver/config/` where *`DOMAIN_DIR`* is the root directory of the WebLogic SIP Server domain.

Editing datatier.xml

You can edit `datatier.xml` using either the Administration Console or a text editor. Note that changes to the data tier configuration cannot be applied to servers dynamically; you must restart servers in order to change data tier membership or reconfigure partitions.

XML Schema

```
<?xml version=”1.0” encoding=”UTF-8”?>

<xsd:schema
targetNamespace=”http://bea.com/wcp/sip/management/internal/webapp”

  xmlns:xsd=”http://www.w3.org/2001/XMLSchema”

  elementFormDefault=”qualified”

  xmlns:data-tier=”http://bea.com/wcp/sip/management/internal/webapp”>

  <xsd:element name=”data-tier”>

    <xsd:complexType>

      <xsd:sequence>

        <xsd:element name=”partition” type=”data-tier:partitionType”
minOccurs=”0” maxOccurs=”500”/>

      </xsd:sequence>

    </xsd:complexType>

  </xsd:element>

  <xsd:complexType name=”partitionType”>

    <xsd:sequence>

      <xsd:element name=”name” type=”xsd:string”/>

      <xsd:element name=”server-name” type=”xsd:string” minOccurs=”1”
maxOccurs=”8”/>

    </xsd:sequence>

  </xsd:complexType>

</xsd:schema>
```

Example datatier.xml File

Listing 4-7 shows the template `datatier.xml` file created using the Configuration Wizard. See also “Example Data Tier Configurations and Configuration Files” on page 3-4.

Listing 4-7 Default datatier.xml File

```
<st:data-tier
xmlns:st=”http://bea.com/wcp/sip/management/internal/webapp”>
 <st:partition>
   <st:name>partition-0</st:name>
   <st:server-name>replica1</st:server-name>
   <st:server-name>replica2</st:server-name>
 </st:partition>
</st:data-tier>
```

XML Element Description

`datatier.xml` contains one or more `partition` elements that define servers’ membership in a data tier partition. All data tier clusters must have at least one `partition`.’ Each partition contains the XML elements described in Table 4-5.

Table 4-5 Nested partition Elements

Element	Description
`name`	A String value that identifies the name of the partition. BEA recommends including the number of the partition (starting at 0) in the text of the name for administrative purposes. For example, “partition-0.”
`server-name`	Specifies the name of a WebLogic SIP Server instance that manages call state in this partition. You can define up two three servers per `partition` element. Multiple servers in the same partition maintain the same call state data, and are referred to as *replicas*. BEA recommends including the number of the server (starting with 0) and the number of the partition in the server name for administrative purposes. For example, “replica-0-0.”

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" on page F-1
- "Tuning Garbage Collection with JRockit" on page F-2
- "Tuning Garbage Collection with Sun JDK" on page F-2

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.

APPENDIX G

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.