BEA WebLogic SIP Server™

Configuration Guide

Version 3.1
Revised: July 16, 2007

Copyright

Restricted Rights Legend

Trademarks and Service Marks

Contents

6. Configuring Engine Tier Container Properties

D. 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 3.1 architecture:

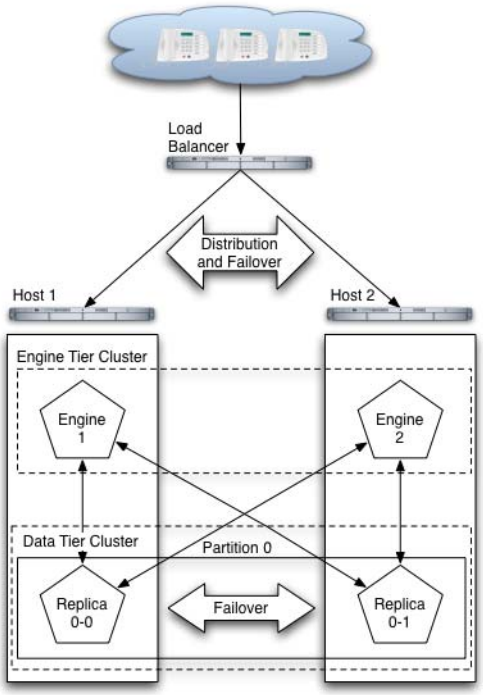
Goals of the WebLogic SIP Server Architecture

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

- Load Balancer
- Engine Tier
- Data tier

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

Figure 1-1 WebLogic SIP Server Architecture



Load Balancer

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

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

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

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

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

Engine Tier

The engine tier is a cluster of WebLogic SIP Server instances that hosts the SIP Servlets and other applications that provide features to SIP clients. The engine tier is a stateless cluster of servers, and it stores no permanent or transient information about the state of SIP dialogs. Instead, all stateful information about SIP dialogs is stored and retrieved from the Data tier, which also provides replication and failover services for SIP session data.

Engine tier servers can optionally cache a portion of the session data managed by the data tier. Caching is most useful in configurations that use a SIP-aware load balancer. See "Using the Engine Tier Cache" on page 7-1.

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

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

Notes: WebLogic SIP Server start scripts use default values for many JVM parameters that affect performance. For example, JVM garbage collection and heap size parameters may be omitted, or may use values that are acceptable only for evaluation or development purposes. In a production system, you must rigorously profile your applications with

different heap size and garbage collection settings in order to realize adequate performance. See "Tuning JVM Garbage Collection for Production Deployments" on page C-1 for suggestions about maximizing JVM performance in a production domain.

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

Data tier

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

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

Within the data tier, session data is managed in one or more "partitions" where each partition manages a fixed portion of the concurrent call state. For example, in a system that uses two partitions, the first partition manages one half of the concurrent call state (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. A simple hashing algorithm is used to ensure that each call state is uniquely assigned to only one data tier partition.

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. 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.

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

Example of Writing and Retrieving Call State Data

When an initial SIP message is received, WebLogic SIP Server uses Servlet mapping rules to direct the message to the appropriate SIP Servlet deployed in the engine tier. The engine tier maintains no stateful information about SIP dialogs, but instead persists the call state to the engine tier at SIP transaction boundaries. A hashing algorithm is applied to the call state to select a single data tier partition in which to store the call state data. The engine tier server then "writes" the call state to each replica within that partition and locks the call state. For example, if the data tier is configured to use two data tier servers within each partition, the engine tier opens a connection to both replicas in the partition, and writes and locks the call state on each replica.

In a default configuration, the replicas maintain the call state information only in memory (available RAM). Call state data can also be configured for longer-term storage in an RDBMS, and it may also be persisted to an off-site WebLogic SIP Server installation for geographic redundancy.

When subsequent SIP messages are generated for the SIP dialog, the engine tier must first retrieve the call state data from the data tier. The hashing algorithm is again applied to determine the partition that stores the call state data. The engine tier then asks each replica in the partition to unlock and retrieve the call state data, after which a Servlet on the engine tier can update the call state data.

RDBMS Storage for Long-Lived Call State Data

WebLogic SIP Server enables you to store long-lived call state data in an Oracle RDBMS in order to conserve RAM. The data tier persists a call state's data to the RDBMS after the call dialog has been established, and retrieves or deletes the persisted call state data as necessary to modify or remove the call state. See "Storing Long-Lived Call State Data in an RDBMS" on page 4-1.

Geographically-Redundant Installations

WebLogic SIP Server can be installed in a geographically-redundant configuration for customers who have multiple, regional data centers, and require continuing operation even after a catastrophic site failure. The geographically-redundant configuration enables multiple WebLogic SIP Server installations (complete with engine and data tier clusters) to replicate call state transactions between one another. If the a particular site's installation were to suffer a critical

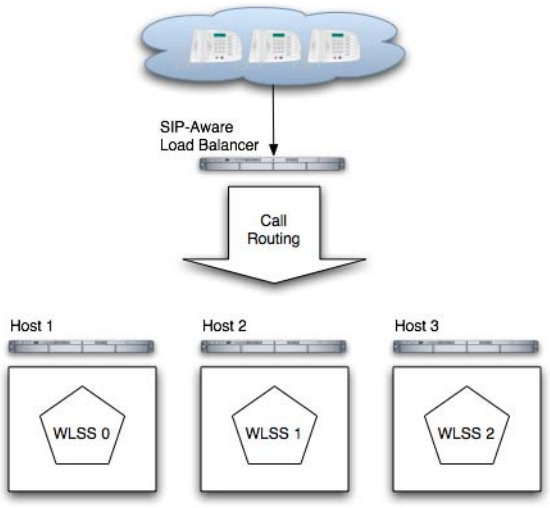
failure, the administrator could choose to redirect all network traffic to the secondary, replicated site to minimize lost calls. See "Configuring Geographically- Redundant Installations" on page 5-1.

Alternate Configurations

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

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

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



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

CHAPTER 2

Overview of WebLogic SIP Server Configuration and Management

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

Shared Configuration Tasks for WebLogic SIP Server and WebLogic Server

WebLogic SIP Server is based on the award-winning WebLogic Server 9.2 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, 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 are addressed in the WebLogic Server 9.2 Documentation.

WebLogic SIP Server Configuration Overview

The SIP Servlet container, data tier replication, and Diameter protocol features of WebLogic SIP Server are implemented in the WebLogic Server 9.2 product as custom resources. A pair of custom resources, `sipserver` and `datatier`, implement the engine tier SIP Servlet container functionality and data tier replication functionality. In production deployments, both resources are generally installed. Specialized deployments may use only the `sipserver` resource in conjunction with a SIP-aware load balancer, as described in "Alternate Configurations" on page 1-6.

Another custom resource, `diameter`, provides Diameter base protocol functionality, and is required only for deployments that utilize one or more Diameter protocol applications.

The WebLogic SIP Server custom resource assignments are visible in the domain configuration file, `config.xml`, and should not be modified. Listing 2-1 shows the definitions for each resource. Note that the `sipserver` and `datatier` resources must each be targeted to the same servers or clusters; in Listing 2-1, the resources are deployed to both the engine tier and data tier cluster.

Listing 2-1 WebLogic SIP Server Custom Resources

```
<custom-resource>
    <name>sipserver</name>
    <target>BEA_DATA_TIER_CLUST,BEA_ENGINE_TIER_CLUST</target>
    <descriptor-file-name>custom/sipserver.xml</descriptor-file-name>

<resource-class>com.bea.wcp.sip.management.descriptor.resource.SipServerRe
source</resource-class>

<descriptor-bean-class>com.bea.wcp.sip.management.descriptor.beans.SipServ
erBean</descriptor-bean-class>
</custom-resource>
<custom-resource>
    <name>datatier</name>
    <target>BEA_DATA_TIER_CLUST,BEA_ENGINE_TIER_CLUST</target>
```

```
    <descriptor-file-name>custom/datatier.xml</descriptor-file-name>

<resource-class>com.bea.wcp.sip.management.descriptor.resource.DataTierRes
ource</resource-class>

<descriptor-bean-class>com.bea.wcp.sip.management.descriptor.beans.DataTie
rBean</descriptor-bean-class>
  </custom-resource>
<custom-resource>
    <name>diameter</name>
    <target>BEA_ENGINE_TIER_CLUST</target>
    <deployment-order>200</deployment-order>
    <descriptor-file-name>custom/diameter.xml</descriptor-file-name>
    <resource-class>com.bea.wcp.diameter.DiameterResource</resource-class>

<descriptor-bean-class>com.bea.wcp.diameter.management.descriptor.beans.Co
nfigurationBean</descriptor-bean-class>
</custom-resource>
```

The WebLogic SIP Server custom resources utilize the basic domain resources defined in `config.xml`, such network channels, cluster and server configuration, and J2EE resources. However, WebLogic SIP Server-specific resources are configured in separate configuration files based on functionality:

- `sipserver.xml` configures SIP container properties and general WebLogic SIP Server engine tier functionality.
- `datatier.xml` identifies servers that participate as replicas in the data tier, and also defines the number and layout of data tier partitions.
- `diameter.xml` configures Diameter nodes and Diameter protocol applications used in the domain.

Keep in mind that the domain configuration file, `config.xml`, defines all of the Managed Servers available in the domain. The `sipserver.xml`, `datatier.xml`, and `diameter.xml` configuration files included in the `sipserver` application determines the role of each server

instance, such as whether they behave as data tier replicas, engine tier nodes, or Diameter client nodes.

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.

Diameter Configuration

The Diameter protocol implementation is implemented as a custom resource separate from the SIP Servlet container functionality. The Diameter configuration file configures one or more Diameter protocol applications to provide Diameter node functionality. WebLogic SIP Server provides the Diameter protocol applications to support the following node types:

- Diameter Sh interface client node (for querying a Home Subscriber Service)
- Diameter Rf interface client node (for offline charging)
- Diameter Ro interface client node (for online charging)
- Diameter relay node
- HSS simulator node (suitable for testing and development only, not for production deployment)

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

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

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 2-5

Administration Console

WebLogic SIP Server provides Administration Console extensions that allow you to modify and SIP Servlet container, SIP Servlet domain, and Diameter configuration properties using a graphical user interface. The Administration Console extensions for WebLogic SIP Server are similar to the core console available in WebLogic Server 9.2. All WebLogic SIP Server configuration and monitoring is provided via these nodes in the left pane of the console:

- SipServer—configures SIP Servlet container properties and other engine tier functionality. This extension also enables you to view (but not modify) data tier partitions and replicas. See "Configuring Engine Tier Container Properties" on page 6-1 for more information about configuring the SIP Servlet container using the Administration Console.
- Diameter—configures Diameter nodes and applications.

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 6-1 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`, `datatier.xml`, and `diameter.xml` by hand, following the respective schemas described in the *Configuration Reference Manual*.

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

Custom JMX Applications

WebLogic SIP Server properties are represented by JMX-compliant MBeans. You can therefore program JMX applications to configure SIP container properties using the appropriate 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 6-4 (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.

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 6-1	• Configuring SIP Container Properties using the Administration Console • Using WLST to perform batch configuration
"Configuring Data Tier Partitions and Replicas" on page 3-1	• Assigning WebLogic SIP Server instances to the data tier partitions • Replicating call state using multiple data tier instances
"Managing WebLogic SIP Server Network Resources" on page 7-1	• Configuring WebLogic Server network channels to handling SIP and HTTP traffic • Setting up multi-homed server hardware • Configuring load balancers for use with WebLogic SIP Server

Table 2-1 Common WebLogic SIP Server Configuration Tasks

Task	Description
Configuring Digest Authentication in *Configuring Security*	• Configuring the LDAP Digest Authentication Provider • Configuring a trusted host list
"Logging SIP Requests and Responses" on page 10-1	• Configuring logging Servlets to record SIP requests and responses. • Defining log criteria for filtering logged messages • Maintaining WebLogic SIP Server log files

CHAPTER 3

Configuring Data Tier Partitions and Replicas

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

Overview of Data Tier Configuration

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

The data tier cluster is arranged into one or more *partitions*. A partition consists of one or more data tier server instances that manage the same portion of 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.

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

datatier.xml Configuration File

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

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

Configuration Requirements and Restrictions

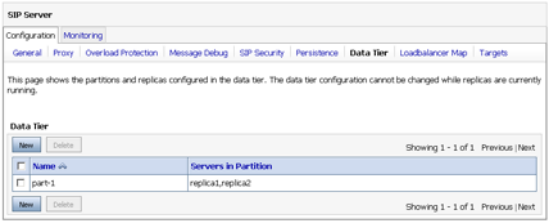
All servers that participate in the data tier should be members of the same WebLogic Server cluster. The cluster configuration enables each server to monitor the status of other servers. Using a cluster also enables you to easily target the `sipserver` and `datatier` custom resources to all servers for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

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

You cannot change the data tier configuration while replicas or engine tier nodes are running. You must restart servers in the domain 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 (SipServer node) of the Administration Console, as shown in Figure 3-1.

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



Best Practices for Configuring and Managing Data Tier Servers

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

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

Data tier servers can have one of three different statuses:

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

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

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

Example Data Tier Configurations and Configuration Files

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

Data Tier with One Partition

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

Listing 3-1 Data Tier Configuration for Small Deployment

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

To add a replica to an existing partition, simply define a second `server-name` entry in the same partition. For example, the `datatier.xml` configuration file shown in Listing 3-2 creates a two-replica configuration.

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

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

```
    <partition>
      <name>Partition0</name>
      <server-name>DataNode0-0</server-name>
      <server-name>DataNode0-1</server-name>
    </partition>
  </data-tier>
```

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

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

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. Listing 3-4 shows the `datatier.xml` configuration file used to configure a system with two partitions and two servers (replicas) in each partition.

Listing 3-4 Data Tier Configuration for Small Deployment

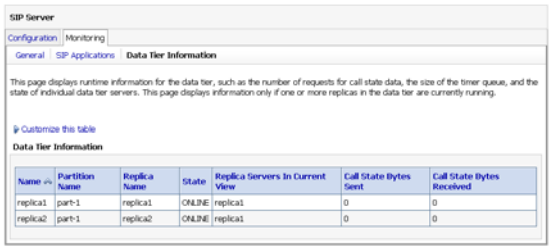
```
<?xml version="1.0" encoding="UTF-8"?>
  <data-tier xmlns="http://www.bea.com/ns/wlcp/wlss/300">
    <partition>
      <name>Partition0</name>
      <server-name>DataNode0-0</server-name>
      <server-name>DataNode0-1</server-name>
    </partition>
    <partition>
      <name>Partition1</name>
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

Monitoring and Troubleshooting Data Tier Servers

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

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

Figure 3-2 Data Tier Monitoring in the Administration Console



Listing 3-5 shows a simple WLST session that queries the current attributes of a single Managed Server instance in a data tier partition. Table 3-1, "ReplicaRuntimeMBean Method and Attribute Summary," on page 3-8 describes the MBean services in more detail.

Listing 3-5 Displaying ReplicaRuntimeMBean Attributes

```
connect('weblogic','weblogic','t3://datahost1:7001')
custom()
cd('com.bea')
cd('com.bea:ServerRuntime=replica1,Name=replica1,Type=ReplicaRuntime')
ls()
-rw-   BackupStoreInboundStatistics                null
-rw-   BackupStoreOutboundStatistics               null
-rw-   BytesReceived                               0
-rw-   BytesSent                                   0
-rw-   CurrentViewId                               2
-rw-   DataItemCount                               0
-rw-   DataItemsToRecover                          0
-rw-   DatabaseStoreStatistics                     null
-rw-   HighKeyCount                                0
-rw-   HighTotalBytes                              0
```

```
-rw-   KeyCount                                   0
-rw-   Name                                       replica1
-rw-   Parent                                    com.bea:Name=replica1,Type=S
erverRuntime
-rw-   PartitionId                                0
-rw-   PartitionName                             part-1
-rw-   ReplicaId                                 0
-rw-   ReplicaName                               replica1
-rw-   ReplicaServersInCurrentView               java.lang.String[replica1,
replica2]
-rw-   ReplicasInCurrentView                     [I@75378c
-rw-   State                                     ONLINE
-rw-   TimerQueueSize                            0
-rw-   TotalBytes                                0
-rw-   Type                                      ReplicaRuntime
```

Table 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.
`BackupStoreInboundStatistics`	Provides statistics about call state data replicated from a remote geographical site.
`BackupStoreOutboundStatistics`	Provides statistics about call state data replicated to a remote geographical site.
`BytesReceived`	The total number of bytes received by this data tier server. Bytes are received as servers in the engine tier provide call state data to be stored.

Table 3-1 ReplicaRuntimeMBean Method and Attribute Summary

Method/Attribute	Description
`BytesSent`	The total number of bytes sent from this data tier server. Bytes are sent to engine tier servers when requested to provide the stored call state.
`CurrentViewId`	The current view ID. Each time the layout of the data tier changes, the view ID is incremented. For example, as multiple servers in a data tier cluster are started for the first time, the view ID is incremented when each server begins participating in the data tier. Similarly, the view is incremented if a server is removed from the data tier, either intentionally or due to a failure.
`DataItemCount`	The total number of stored call state keys for which this server has data. This attribute may be lower than the `KeyCount` attribute if the server is currently recovering data.
`DataItemsToRecover`	The total number of call state keys that must still be recovered from other replicas in the partition. A data tier server may recover keys when it has been taken offline for maintenance and is then restarted to join the partition.
`HighKeyCount`	The highest total number of call state keys that have been managed by this server since the server was started.
`HighTotalBytes`	The highest total number of bytes occupied by call state data that this server has managed since the server was started.
`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.

Table 3-1 ReplicaRuntimeMBean Method and Attribute Summary

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

Storing Long-Lived Call State Data in an RDBMS

The following sections describe how to configure a WebLogic SIP Server domain to use an Oracle or MySQL RDBMS with the data tier cluster, in order to conserve RAM:

Overview of Long-Lived Call State Storage

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

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

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

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

RDBMS call state storage can be used in combination with an engine tier cache, if your domain uses a SIP-aware load balancer to manage connections to the engine tier. See "Using the Engine Tier Cache" on page 7-1.

Requirements and Restrictions

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

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

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

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

Steps for Enabling RDBMS Call State Storage

In order to use the RDBMS call state storage feature, your WebLogic SIP Server domain must include the necessary JDBC configuration, SIP Servlet container configuration, and a database having the schema required to store the call state. You can automate much of the required

configuration by using the Configuration Wizard to set up a new domain with the RDBMS call state template. See "Using the Configuration Wizard RDBMS Store Template" on page 4-3.

If you have an existing WebLogic SIP Server domain, or you want to configure the RDBMS store on your own, see "Configuring RDBMS Call State Storage by Hand" on page 4-5 for instructions to configure JDBC and WebLogic SIP Server to use an RDBMS store.

Using the Configuration Wizard RDBMS Store Template

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

1. Start the Configuration Wizard application:

   ```
   cd ~/bea/sipserver30/common/bin
   ./config.sh
   ```

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

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

   – A JDBC datasource, `wlss.callstate.datasource`, required for storing long-lived call state data. Note that you must modify this configuration to configure the datasource for your own RDBMS server. See "Modify the JDBC Datasource Connection Information" on page 4-4.

  - A persistence configuration (shown in the SipServer node, Configuration->Persistence tab of the Administration Console) that defines default handling of persistence hints for both RDBMS and geographical redundancy.

10. Click Done to exit the configuration wizard.
11. Follow the steps under "Modify the JDBC Datasource Connection Information" on page 4-4 to create the necessary tables in your RDBMS.
12. Follow the steps under "Create the Database Schema" on page 4-6 to create the necessary tables in your RDBMS.

Modify the JDBC Datasource Connection Information

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

1. Use your browser to access the URL http://*address*:*port*/console where *address* is the Administration Server's listen address and *port* is the listen port.
2. Click Lock & Edit to obtain a configuration lock.
3. Select the Services->JDBC->Data Sources tab in the left pane.
4. Select the data source named wlss.callstate.datasource in the right pane.
5. Select the Configuration->Connection Pool tab in the right pane.
6. Modify the following connection pool properties:
   - **URL**: Modify the URL to specify the hostname and port number of your RDBMS server.
   - **Properties**: Modify the value of the user, portNumber, SID, and serverName properties to match the connection information for your RDBMS.
   - **Password** and **Confirm Password**: Enter the password of the RDBMS user you specified.
7. Click Save to save your changes.
8. Select the Targets tab in the right pane.
9. On the Select Targets page, select the name of your data tier cluster (for example, BEA_DATA_TIER_CLUST), then click Save.

10. Click Activate Changes to apply the configuration.
11. Follow the steps under "Create the Database Schema" on page 4-6 to create the necessary tables in your RDBMS.

Configuring RDBMS Call State Storage by Hand

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

Configure JDBC Resources

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

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

10. On the Select Targets page, select the name of your data tier cluster (for example, BEA_DATA_TIER_CLUST), then click Finish.
11. Click Save to save your changes.
12. Click Activate Changes to apply the configuration.

Configure WebLogic SIP Server Persistence Options

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

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

Create the Database Schema

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

The contents of the `callstate.sql` SQL script are shown in Listing 4-1.

Listing 4-1 callstate.sql Script for Call State Storage Schema

```
drop table callstate;
```

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

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

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

   ```
   cd ~/bea/sipserver30/common/templates/scripts/db/oracle
   ```

2. Start the SQL*Plus application, connecting to the Oracle database in which you will create the required tables. Use the same username, password, and connect to the same database that you specified when configuring the JDBC driver in "Configure JDBC Resources" on page 4-5. For example:

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

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

3. Execute the WebLogic SIP Server SQL script, `callstate.sql`:

   ```
   START callstate.sql
   ```

4. Exit SQL*Plus:

   ```
   EXIT
   ```

Using Persistence Hints in SIP Applications

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

To use the API, simply obtain a `WlssSipApplicationSession` instance and use the `setPersist` method to enable or disable persistence. Note that you can enable or disable persistence either to an RDBMS store, or to as geographically-redundant WebLogic SIP Server installation (see "Configuring Geographically- Redundant Installations" on page 5-1).

For example, some SIP-aware load balancing products use the SIP OPTIONS message to determine if a SIP Server is active. To avoid persisting these messages to an RDBMS and to a geographically-redundant site, a Servlet might implement a `doOptions` method to echo the request and turn off persistence for the message, as shown in Listing 4-2.

Listing 4-2 Disabling RDBMS Persistence for Option Methods

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

CHAPTER 5

Configuring Geographically-Redundant Installations

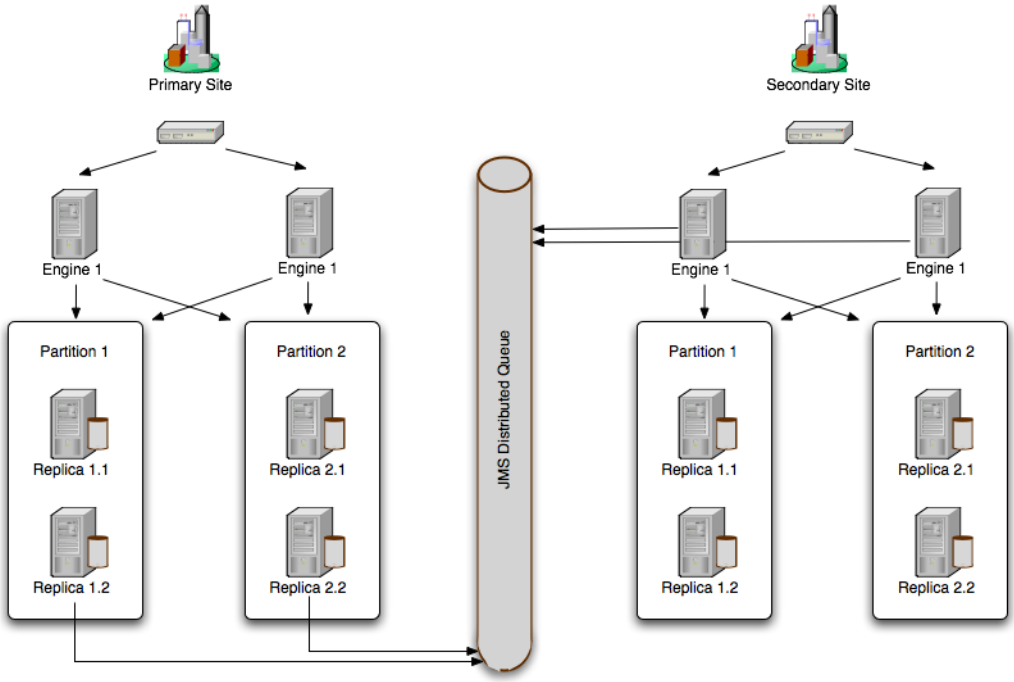
The following sections describe how to replicate call state transactions across multiple, regional WebLogic SIP Server installations (“sites”):

Overview of Geographic Persistence

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

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

Figure 5-1 WebLogic SIP Server Geographic Persistence

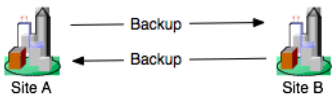


When using geographic persistence, a single replica in the primary site places modified call state data on a distributed JMS queue. By default, data is placed on the queue only at SIP dialog boundaries. (A custom API is provided for application developers that want to replicate data using a finer granularity, as described in "Using Persistence Hints in SIP Applications" on page 4-7.) In a secondary site, engine tier servers use a message listener to monitor the distributed queue to receive messages and write the data to its own data tier cluster. If the secondary site uses an RDBMS to store long-lived call states (recommended), then all data writes from the distribute queue go directly to the RDBMS, rather than to the in-memory storage of the data tier.

Example Domain Configurations

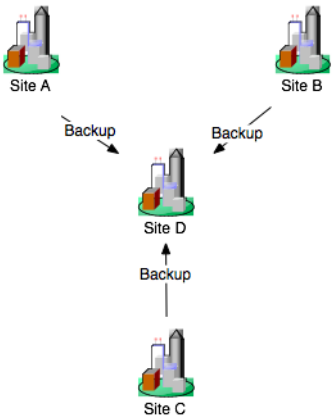
A secondary WebLogic SIP Server domain that persists data from another domain may itself process SIP traffic, or it may exist solely as an active standby domain. In the most common configuration, two sites are configured to replicate each other's call state data, with each site processing its own local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

Figure 5-2 Common Geographically-Redundant Configuration



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

Figure 5-3 Alternate Geographically-Redundant Configuration



Requirements and Limitations

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

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

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

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

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

Steps for Configuring Geographic Persistence

In order to use the WebLogic SIP Server geographic persistence features, you must perform certain configuration tasks on both the primary "home" site and on the secondary replication site. Table 5-1

Table 5-1 Steps for Configuring Geographic Persistence

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

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

WebLogic SIP Server provides domain templates to automate the configuration of most of the resources described in Table 5-1. See "Using the Configuration Wizard Templates for Geographic Persistence" on page 5-5 for information about using the templates.

If you have an existing WebLogic SIP Server domain and want to use geographic persistence, follow the instructions in "Configuring Geographical Redundancy by Hand" on page 5-8 to create the resources.

Using the Configuration Wizard Templates for Geographic Persistence

WebLogic SIP Server provides two Configuration Wizard templates for using geographic persistence features:

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

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

Installing and Configuring the Primary Site

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

1. Start the Configuration Wizard application:

   ```
   cd ~/bea/sipserver30/common/bin
   ./config.sh
   ```

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

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

   – A JDBC datasource, `wlss.callstate.datasource`, required for storing long-lived call state data. If you want to use this functionality, edit the datasource to include your RDBMS connection information as described in “Modify the JDBC Datasource Connection Information” on page 4-4.

  – A persistence configuration (shown in the SipServer node, Configuration->Persistence tab of the Administration Console) that defines:
    - Default handling of persistence hints for both RDBMS and geographic persistence.
    - A Geo Site ID of 1.
    - A Geo Remote T3 URL of `t3://localhost:8011,localhost:8061`, which identifies the engine tier servers in the "geo2" domain as the replication site for geographic redundancy.

10. Click Done to exit the configuration wizard.
11. Follow the steps under "Installing the Secondary Site" on page 5-7 to create the domain that performs the replication.

Installing the Secondary Site

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

1. Start the Configuration Wizard application:

   ```
   cd ~/bea/sipserver30/common/bin
   ./config.sh
   ```

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

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

- A JDBC datasource, `wlss.callstate.datasource`, required for storing long-lived call state data. If you want to use this functionality, edit the datasource to include your RDBMS connection information as described in "Modify the JDBC Datasource Connection Information" on page 4-4.
- A persistence configuration (shown in the SipServer node, Configuration->Persistence tab of the Administration Console) that defines:
  - Default handling of persistence hints for both RDBMS and geographical redundancy.
  - A Geo Site ID of 2.
- A JMS system module, `SystemModule-Callstate`, that includes:
  - `ConnectionFactory-Callstate`, a connection factory required for backing up call state data from a primary site.
  - `DistributedQueue-Callstate`, a uniform distributed queue required for backing up call state data from a primary site.

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

10. Click Done to exit the configuration wizard.

Configuring Geographical Redundancy by Hand

If you have an existing replicated WebLogic SIP Server installation, or pair of installations, you must create by hand the JMS and JDBC resources required for enabling geographical redundancy. You must also configure each site to perform replication. These basic steps for enabling geographical redundancy are:

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

The sections that follow describe each step in detail.

Configuring JDBC Resources (Primary and Secondary Sites)

Follow the instructions in "Storing Long-Lived Call State Data in an RDBMS" on page 4-1 to configure the JDBC resources required for storing long-lived call states in an RDBMS.

Configuring Persistence Options (Primary and Secondary Sites)

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

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

Configuring JMS Resources (Secondary Site Only)

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

Follow these steps to configure JMS resources:

1. Use your browser to access the URL http://*address*:*port*/console where *address* is the Administration Server's listen address and *port* is the listen port.
2. Click Lock & Edit to obtain a configuration lock.
3. Select the Services->Messaging->JMS Servers tab in the left pane.
4. Click New in the right pane.
5. Enter a unique name for the JMS Server or accept the default name. Click Next to continue.
6. In the Target list, select the name of a single engine tier server node in the installation. Click Finish to create the new Server.
7. Repeat Steps 3-6 for to create a dedicated JMS Server for each engine tier server node in your installation.
8. Select the Services->Messaging->JMS Modules node in the left pane.
9. Click New in the right pane.
10. Fill in the fields of the Create JMS System Module page as follows:
    - **Name**: Enter a name for the new module, or accept the default name.
    - **Descriptor File Name**: Enter the prefix a configuration file name in which to store the JMS module configuration (for example, `systemmodule-callstate`).
11. Click Next to continue.
12. Select the name of the engine tier cluster, and choose the option **All servers in the cluster**.
13. Click Next to continue.
14. Select **Would you like to add resources to this JMS system module** and click Finish to create the module.
15. Click New to add a new resource to the module.
16. Select the **Connection Factory** option and click Next.

17. Fill in the fields of the Create a new JMS System Module Resource as follows:
    - **Name**: Enter a descriptive name for the resource, such as ConnectionFactory-Callstate.
    - **JNDI Name**: Enter the name `wlss.callstate.backup.site.connection.factory`.
18. Click Next to continue.
19. Click Finish to save the new resource.
20. Select the name of the connection factory resource you just created.
21. Select the Configuration->Load Balance tab in the right pane.
22. De-select the **Server Affinity Enabled** option, and click Save.
23. Re-select the Services->Messaging->JMS Modules node in the left pane.
24. Select the name of the JMS module you created in the right pane.
25. Click New to create another JMS resource.
26. Select the **Distributed Queue** option and click Next.
27. Fill in the fields of the Create a new JMS System Module Resource as follows:
    - **Name**: Enter a descriptive name for the resource, such as DistributedQueue-Callstate.
28. **JNDI Name**: Enter the name Fill in the fields of the Create a new JMS System Module Resource as follows:
    - **Name**: Enter a descriptive name for the resource, such as ConnectionFactory-Callstate.
    - **JNDI Name**: Enter the name `wlss.callstate.backup.site.queue`.
29. Click Next to continue.
30. Click Finish to save the new resource.
31. Click Save to save your configuration changes.
32. Click Activate Changes to apply your changes to the engine tier servers.

Understanding Geo-Redundant Replication Behavior

This section provides more detail into how multiple sites replicate call state data. Administrators can use this information to better understand the mechanics of geo-redundant replication and to

better troubleshoot any problems that may occur in such a configuration. Note, however, that the internal workings of replication across WebLogic SIP Server installations is subject to change in future releases of the product.

Call State Replication Process

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

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

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

Call State Processing After Failover

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

- Setting the site ID associated with the call to zero (making it appear local).
- Activating all dormant timers present in the call state.

By default, call states are activated only for individual calls, and only after those calls are requested on the backup site. `SipServerRuntimeMBean` includes a method, `activateBackup(byte site)`, that can be used to force a site to take over all call state data that it has replicated from another site. The Administrator can execute this method using a WLST configuration script. Alternatively, an application deployed on the server can detect when a request for replicated site data occurs, and then execute the method. Listing 5-1 shows sample code from a JSP that activates a secondary site, changing ownership of all call state data

replicated from site 1. Similar code could be used within a deployed Servlet. Note that either a JSP or Servlet must run as a privileged user in order to execute the `activateBackup` method.

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

Listing 5-1 Activating a Secondary Site Using JMX

```
<%
    byte site = 1;


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


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


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

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

Removing Backup Call States

You may also choose to stop replicating call states to a remote site in order to perform maintenance on the remote site or to change the backup site entirely. Replication can be stopped by setting the **Site Handling** attribute to "none" on the primary site as described in "Configuring Persistence Options (Primary and Secondary Sites)" on page 5-9.

After disabling geographical replication on the primary site, you also may want to remove backup call states on the secondary site. `SipServerRuntimeMBean` includes a method, `deleteBackup(byte site)`, that can be used to force a site to remove all call state data that it has replicated from another site. The Administrator can execute this method using a WLST configuration script or via an application deployed on the secondary site. The steps for executing this method are similar to those for using the `activateBackup` method, described in "Call State Processing After Failover" on page 5-12.

Monitoring Replication Across Regional Sites

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

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

See the JavaDoc for more information about `ReplicaRuntimeMBean`.

Troubleshooting Geographical Replication

In addition to using the `ReplicaRuntimeMBean` methods described in "Monitoring Replication Across Regional Sites" on page 5-14, Administrators should monitor any SNMP traps that indicate failed database writes on a secondary site installation.

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

CHAPTER 6

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

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 6-2 describes how to configure container properties using the Administration Console graphical user interface.

"Configuring Container Properties Using WLST (JMX)" on page 6-4 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*:*port*/console where *address* is the Administration Server's listen address and *port* is the listen port.
2. Select the SipServer node in the left pane.

   The right pane of the console provides two levels of tabbed pages that are used for configuring and monitoring WebLogic SIP Server. Table 6-1 summarizes the available

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

Table 6-1 WebLogic SIP Server Configuration and Monitoring Pages

Page		Function
Configuration->	General	Configure SIP timer values, session timeout duration, default WebLogic SIP Server behavior (proxy or user agent), server header format, call state caching, DNS name resolution, domain aliases, rport support, and diagnostic image format.
	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.
	Persistence	Configure persistence options for storing long-lived session data in an RDBMS, or for replicating long-lived session data to a remote, geographically-redundant site.
	Data Tier	View the current configuration of data tier servers.
	LoadBalancer Map	Configure the mapping of multiple clusters to internal virtual IP addresses during a software upgrade.
	Targets	Configure the list of servers or clusters that receive the engine tier configuration. The target server list determines which servers and/or clusters provide SIP Servlet container functionality.
	Connection Pools	Configure connection reuse pools to minimize communication overhead with a Session Border Control (SBC) function or Serving Call Session Control Function (S-CSCF).
Monitoring->	General	View runtime information about messages and sessions processed in engine tier servers.
	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 Lock & Edit button. Locking a configuration prevents other Administrators from modifying the configuration at the same time.

If you obtain a lock on the configuration, you can change SIP Servlet container attribute values on multiple configuration pages, saving the changes as needed. You then have two options depending on whether you want to keep or discard the changes you have made:

- Activate Changes—Persists all saved current changes to the `sipserver.xml` file.
- Undo All Changes—Discards your current changes, deleting any temporary configuration files that were written with previous Save operations.

Note that WebLogic SIP Server automatically saves the original boot configuration in the file `sipserver.xml.booted` in the `config/custom` subdirectory of the domain directory. 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. Full documentation for WLST is available at http://e-docs.bea.com/wls/docs92/config_scripting/index.html.

Before using WLST to configure a WebLogic SIP Server domain, set you environment to add required WebLogic SIP Server classes to your classpath. Use either a domain environment script or the setWLSEnv.sh script located in *`WLSS_HOME`*`/server/bin` where *`WLSS_HOME`* is the root of your WebLogic SIP Server installation.

Managing Configuration Locks

Table 6-2 summarizes the WLST methods used to lock a configuration and apply changes.

Table 6-2 ConfigManagerRuntimeMBean Method Summary

Method	Description
`activate()`	Writes the current configuration MBean attributes (the current SIP Servlet container configuration) to the `sipserver.xml` configuration file and applies changes to the running servers.
`cancelEdit()`	Cancels an edit session, releasing the edit lock, and discarding all unsaved changes. This operation can be called by any user with administrator privileges, even if the user did not start the edit session.
`save()`	Writes the current configuration MBean attributes (the current SIP Servlet container configuration) to a temporary configuration file.
`startEdit()`	Locks changes to the SIP Servlet container configuration. Other JMX applications cannot alter the configuration until you explicitly call `stopEdit()`, or until your edit session is terminated. If you attempt to call `startEdit()` when another user has obtained the lock, you receive an error message that states the user who owns the lock.
`stopEdit()`	Releases the lock obtained for modifying SIP container properties and rolls back any pending MBean changes, discarding any temporary files.

A typical configuration session involves the following tasks:

1. Call `startEdit()` to obtain a lock on the active configuration.
2. Modify existing SIP Servlet container configuration MBean attributes (or create or delete configuration MBeans) to modify the active configuration. See "Configuration MBeans for the SIP Servlet Container" on page 6-6 for a summary of the configuration MBeans.
3. Call `save()` to persist all changes to a temporary configuration file named `sipserver.xml.saved`, or

4. Call `activate()` to persist changes to the `sipserver.xml.saved` file, rename `sipserver.xml.saved` to `sipserver.xml` (copying over the existing file), and apply changes to the running engine tier server nodes.

   Note: When you boot the Administration Server for a 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 6-11.

Configuration MBeans for the SIP Servlet Container

`ConfigManagerRuntimeMBean` manages access to and persists the configuration MBean attributes described in Table 6-3. 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 6-3 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 Upgrading Software in the *Operations Guide* for more information.
`OverloadProtection`	`RegulationPolicy`, `ThresholdValue`, `ReleaseValue`	Manages overload settings for throttling incoming SIP requests. See also overload in the *Configuration Reference Manual*.
`Proxy`	`ProxyURIs`, `RoutingPolicy`	Manages the URIs routing policies for proxy servers. See also proxy—Setting Up an Outbound Proxy Server in the *Configuration Reference Manual*.

Table 6-3 SIP Container Configuration MBeans

MBean Type	MBean Attributes	Description
`SipSecurity`	`TrustedAuthenticationHosts`	Defines trusted hosts for which authentication is not performed. See also sip-security in the *Configuration Reference Manual*.

`SipServer`	`DefaultBehavior`, `EnableLocalDispatch`, `MaxApplicationSessionLifeTime`, `OverloadProtectionMBean`, `ProxyMBean`, `T1TimeoutInterval`, `T2TimeoutInterval`, `T4TimeoutInterval`, `TimerBTimeoutInterval`, `TimerFTimeoutInterval`
	
	`SipServer also has several helper methods: createProxy(), destroyProxy(), createOverloadProtection(), destroyOverloadProtection(), createClusterToLoadBalancerMap(), destroyClusterToLoadBalancerMap()`
		Configuration MBean that represents the entire `sipserver.xml` configuration file. You can use this MBean to obtain and manage each of the individual MBeans described in this table, or to set SIP timer or SIP Session timeout values. See also "Creating and Deleting MBeans" on page 6-10, `default-behavior`, `enable-local-dispatch`, `max-application-session-lifetime`, `t1-timeout-interval`, `t2-timeout-interval`, `t4-timeout-interval`, `timerB-timeout-interval`, and `timerF-timeout-interval` in the *Configuration Reference Manual*.

Locating the WebLogic SIP Server MBeans

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

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

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

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

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

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

Certain configuration settings, such as proxy and overload protection settings, are defined by default in `sipserver.xml`. Configuration MBeans are generated for these settings when you boot the associated server, so you can immediately browse the `Proxy` and `OverloadProtection` MBeans. Other configuration settings are not configured by default and you will need to create the associated MBeans before they can be accessed. See "Creating and Deleting MBeans" on page 6-10.

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. Follow these steps:

1. Set your WebLogic SIP Server environment:

   ```
   cd ~/bea/sipserver31/server/bin
   ./setWLSEnv.sh
   ```

2. Start WLST:

   ```
   java weblogic.WLST
   ```

3. 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 6-1 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 6-1 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 6-2 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 6-3, "SIP Container Configuration MBeans," on page 6-6` for a listing of other helper methods in `SipServer`, or refer to the WebLogic SIP Server JavaDocs.

Listing 6-2 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 6-3 modifies the sample shown in Listing 6-2, "WLST Commands for Creating and Deleting MBeans," on page 6-10 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 6-3 Invoking Helper Methods for Setting URI Attributes

```
# Import helper method for converting strings to URIs.
from com.bea.wcp.sip.util.URIHelper import stringToSipURIs
connect()
custom()
cd('mydomain:Location=myserver,Name=sipserver,ServerRuntime=myserver,Type=
ConfigManagerRuntime')
cmo.startEdit()
cd('mydomain:DomainConfig=mydomain,Location=myserver,Name=sipserver,Type=S
ipServer')
cmo.createProxy()
cd('mydomain:DomainConfig=mydomain,Location=myserver,Name=sipserver,SipSer
ver=sipserver,Type=Proxy')
stringarg =
jarray.array([java.lang.String("sip://siplb.bea.com:5060")],java.lang.Stri
ng)
uriarg = stringToSipURIs(stringarg)
set('ProxyURIs',uriarg)
cd('mydomain:Location=myserver,Name=sipserver,ServerRuntime=myserver,Type=
ConfigManagerRuntime')
cmo.save()
```

Reverting to the Original Boot Configuration

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

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

Configuring NTP for Accurate SIP Timers

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

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

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

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

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

CHAPTER 7

Using the Engine Tier Cache

The following sections describe how to enable the engine tier cache for improved performance with SIP-aware load balancers:

Overview of Engine Tier Caching

As described in "Overview of the WebLogic SIP Server Architecture" on page 1-1, in the default WebLogic SIP Server configuration the engine tier cluster is stateless. A separate data tier cluster manages call state data in one or more partitions, and engine tier servers fetch and write data in the data tier as necessary. Engines can write call state data to multiple replicas in each partition to provide automatic failover should a data tier replica going offline.

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

The engine tier cache stores only the call state data that has been most recently used by engine tier servers. Call state data is moved into an engine's local cache as necessary in order to respond

to client requests or to refresh out-of-date data. If the cache is full when a new call state must be written to the cache, the least-recently accessed call state entry is first removed from the cache. The size of the engine tier cache is not configurable.

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

Configuring Engine Tier Caching

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

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

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

Monitoring and Tuning Cache Performance

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

Table 7-1 SipPerformanceRuntime Attribute Summary

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

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

APPENDIX A

Upgrading a WebLogic SIP Server 2.2 Configuration to Version 3.1

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

About the Upgrade Process

Upgrading a WebLogic SIP Server 2.2 domain to version 3.1 involves these basic steps:

- Using the WebLogic Server 9.2 upgrade wizard and procedures to upgrade the basic domain configuration (`config.xml` file and startup scripts) to comply with WebLogic Server 9.2, upon which WebLogic SIP Server is based.
- Manually editing the `config.xml` file to specify the custom resources required by WebLogic SIP Server 3.1.
- Manually upgrading WebLogic SIP Server configuration files (`sipserver.xml`, `diameter.xml`) and resources to use the new WebLogic SIP Server 3.1 schemas.

The sections that follow describe these steps in more detail, and reference the WebLogic Server 9.2 documentation where appropriate.

Step 1: Install Software and Prepare Domain

Begin by installing the WebLogic SIP Server 3.1 software into a new BEA home directory on your Administration Server machine. You will need to access the version 3.1 software as well as the domain directory for your existing WebLogic SIP Server 2.2 installation.

In the next section, you will use the WebLogic Server 9.2 upgrade procedure and utility to upgrade the underlying WebLogic Server configuration to version 9.2. Before doing so, prepare your running WebLogic SIP Server 2.2 domain by doing the following:

1. Shut down all Managed Servers in your domain (engine and data tier servers), leaving only the Administration Server running.
2. Undeploy the WebLogic SIP Server 2.2 `sipserver` implementation application (EAR file) from all servers. WebLogic SIP Server uses custom resources to implement SIP Servlet container functionality, and the older `sipserver` application is no longer used. Only the configuration files (`sipserver.xml` and `datatier.xml`) are required in the upgraded domain.
3. Undeploy all Diameter applications and other user applications. The WebLogic Server 9.2 upgrade process requires that you first undeploy all applications from the domain.
4. Shut down the Administration Server.
5. Verify that your `config.xml` file contains no application deployments (no `app-deployment` stanzas) before continuing the upgrade.

Step 2: Use the WebLogic Server 9.2 Upgrade Wizard

Follow the instructions in Upgrading a WebLogic Domain in the WebLogic Server 9.2 documentation to upgrade the underlying WebLogic Server configuration to 9.2. This process updates the `config.xml` file, supporting files (such as startup scripts) and domain structure to comply with WebLogic Server 9.2.

For more information about the WebLogic Server 9.2 procedure, see Upgrading WebLogic Application Environments.

Step 3: Edit the config.xml File to Specify WebLogic SIP Server Resources

WebLogic SIP Server 3.1 implements SIP Servlet container functionality, Diameter functionality, and Administration Console support using custom resources. After upgrading your domain's `config.xml` file to be compliant with WebLogic Server 9.2, manually edit the file to specify the required WebLogic SIP Server custom resources:

1. Move to the `config` subdirectory of the upgraded domain directory:

   ```
   cd ~/bea/user_projects/mydomain/config
   ```

2. Open the `config.xml` file with a text editor.

3. Add the WebLogic SIP Server custom resource definitions to the end of the `config.xml` file, before the final `<admin-server-name>` definition. Listing 7-1 shows the required entries. Substitute *`italicized`* entries with the names of actual engine and data tier clusters configured in your upgraded domain.

Listing 7-1 WebLogic SIP Server Custom Resource Definitions

```
...
  <custom-resource>
    <name>sipserver</name>
    <target>BEA_DATA_TIER_CLUST,BEA_ENGINE_TIER_CLUST</target>
    <descriptor-file-name>custom/sipserver.xml</descriptor-file-name>

<resource-class>com.bea.wcp.sip.management.descriptor.resource.SipServerRe
source</resource-class>

<descriptor-bean-class>com.bea.wcp.sip.management.descriptor.beans.SipServ
erBean</descriptor-bean-class>
  </custom-resource>
  <custom-resource>
    <name>datatier</name>
```

```
    <target>BEA_DATA_TIER_CLUST,BEA_ENGINE_TIER_CLUST</target>
    <descriptor-file-name>custom/datatier.xml</descriptor-file-name>

<resource-class>com.bea.wcp.sip.management.descriptor.resource.DataTierRes
ource</resource-class>

<descriptor-bean-class>com.bea.wcp.sip.management.descriptor.beans.DataTie
rBean</descriptor-bean-class>
  </custom-resource>
  <custom-resource>
    <name>diameter</name>
    <target>BEA_ENGINE_TIER_CLUST</target>
    <deployment-order>200</deployment-order>
    <descriptor-file-name>custom/diameter.xml</descriptor-file-name>
    <resource-class>com.bea.wcp.diameter.DiameterResource</resource-class>

<descriptor-bean-class>com.bea.wcp.diameter.management.descriptor.beans.Di
ameterBean</descriptor-bean-class>
  </custom-resource>
  <custom-resource>
    <name>ProfileService</name>
    <target>BEA_ENGINE_TIER_CLUST</target>
    <deployment-order>300</deployment-order>
    <descriptor-file-name>custom/profile.xml</descriptor-file-name>

<resource-class>com.bea.wcp.profile.descriptor.resource.ProfileServiceReso
urce</resource-class>

<descriptor-bean-class>com.bea.wcp.profile.descriptor.beans.ProfileService
Bean</descriptor-bean-class>
```

```
  </custom-resource>
  <admin-server-name>AdminServer</admin-server-name>
</domain>
```

4. Save your changes and exit the text editor.

Step 4: Relocate and Edit WebLogic SIP Server Configuration Files

The existing WebLogic SIP Server configuration files (`sipserver.xml`, `datatier.xml`, and `diameter.xml`) must be placed in the `config/custom` subdirectory of the upgraded domain directory. Additionally, `sipserver.xml` and `diameter.xml` must be manually edited to conform to the updated WebLogic SIP Server schema.

Upgrade sipserver.xml and datatier.xml Files

In WebLogic SIP Server 2.2, `sipserver.xml` and `datatier.xml` are stored in the `config` subdirectory of the `sipserver` implementation application. Copy them to the new location using a command similar to:

```
cp ~/bea/user_projects/domains/mydomain/sipserver/config/*.xml
~/bea/user_projects/domains/mydomain/custom/config
```

Next, use a text editor to modify `sipserver.xml` to use the new WebLogic SIP Server 3.1 schema. Table A-1 summarizes important schema changes. See also the Engine Tier Configuration Reference (sipserver.xml).

Table A-1 sipserver.xml Schema Updates

Version 2.2 Element	Version 3.1 Equivalent	Notes
`targetNamespace= "http://www.bea.com/ns/wlcp/wlss/220"`	`targetNamespace= "http://www.bea.com/ns/wlcp/wlss/300"`	Change the `targetNamespace` attribute to the new URL for version 3.1.
`overload`	`overload`	Ensure that the order of elements in the `overload` stanza is: `threshold-policy`, `threshold-value`, `release-value`. Also ensure that the previously-deprecated `regulation-policy` element is renamed to `threshold-policy`.
`timerB-timeout-interval`	`timer-b-timeout-interval`	Rename the element.
`timerF-timeout-interval`	`timer-f-timeout-interval`	Rename the element.

Upgrade diameter.xml Files

In WebLogic SIP Server 2.2, each Diameter application uses a distinct `diameter.xml` file. In WebLogic SIP Server 3.1, multiple Diameter node configurations are stored in a single `diameter.xml` file in the `config/custom` directory. Begin by copying all existing `diameter.xml` files to distinct names in the `config/custom` directory, as in:

```
cp
~/bea/user_projects/domains/replicated/diameter_relay1/WEB-INF/config/diam
eter.xml
~/bea/user_projects/domains/replicated/config/custom/diameter_relay.xml

cp
~/bea/user_projects/domains/replicated/diameter_hssclient/WEB-INF/config/d
```

`iameter.xml`
`~/bea/user_projects/domains/replicated/config/custom/diameter_client.xml`

Next, merge the contents of the multiple `diameter.xml` files from your version 2.2 directory into a single, new `diameter.xml` file. Key version 3.1 schema changes to consider are:

- Multiple nodes are defined in multiple `configuration` elements in `diameter.xml`.
- Each node has to be targeted to an engine tier server configured in the domain, using the `target` element.
- Each node can have one or more `application` stanzas, which configure the Diameter applications that run on the host.

Table A-1 summarizes important schema changes. See also the Diameter Configuration Reference (diameter.xml). Listing 7-2 provides a sample `diameter.xml` file that you can use as a template when merging the older files.

Table A-2 diameter.xml Schema Updates

Version 2.2 Element	Version 3.1 Equivalent	Notes
`node`	`configuration`	Multiple `configuration` stanzas are used to configure multiple Diameter nodes. Change each `node` stanza to be a `configuration` stanza in the configuration file.
`n/a`	`target`	Add a `target` element to each `configuration` stanza to target the node configuration to a particular engine tier server. The server(s) listed in the target must also be defined as servers in the domain `config.xml` file.
`debug`	`debug-enabled`	Rename the element.
`message-debug`	`message-debug-enabled`	Rename the element.

Table A-2 diameter.xml Schema Updates

Version 2.2 Element	Version 3.1 Equivalent	Notes
`applications`	`n/a`	There is no high-level `applications` stanza in 3.1. Configure multiple Diameter applications using multiple `application` stanzas.
`auth-application-id`	`n/a`	This ID is defined within the application code itself.
`acct-application-id`	`n/a`	This ID is defined within the application code itself.
`vendor-id`	`n/a`	This ID is defined within the application code itself.
`peers`	`n/a`	There is no high-level `peers` stanza in 3.1. Configure multiple Diameter applications using multiple `peer` stanzas.
`retry-delay`	`peer-retry-delay`	Rename the element.

Listing 7-2 Sample diameter.xml File

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://www.bea.com/ns/wlcp/diameter/300"
elementFormDefault="qualified" attributeFormDefault="unqualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:complexType name="diameter-descriptorType">
    <xs:annotation>
      <xs:documentation>Corresponds to DiameterDescriptorBean

(Interface=com.bea.wcp.diameter.management.descriptor.beans.DiameterDescri
ptorBean)</xs:documentation>
```

```
    </xs:annotation>
    <xs:sequence>
      <xs:element name="name" type="xs:string" minOccurs="0"
nillable="true">
        <xs:annotation>
          <xs:documentation>&lt;p>The name of the WLSS bean.&lt;/p>

(Interface=com.bea.wcp.diameter.management.descriptor.beans.DiameterDescri
ptorBean Attribute=getName)</xs:documentation>
        </xs:annotation>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="paramType">
    <xs:annotation>
      <xs:documentation>Corresponds to ParamBean

(Interface=com.bea.wcp.diameter.management.descriptor.beans.ParamBean)</xs
:documentation>
    </xs:annotation>
    <xs:complexContent>
      <xs:extension base="ns:diameter-descriptorType"
xmlns:ns="http://www.bea.com/ns/wlcp/diameter/300">
        <xs:sequence>
          <xs:element name="value" type="xs:string" minOccurs="0"
nillable="true">
            <xs:annotation>

<xs:documentation>(Interface=com.bea.wcp.diameter.management.descriptor.be
ans.ParamBean Attribute=getValue)</xs:documentation>
```

```
            </xs:annotation>
          </xs:element>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="applicationType">
    <xs:annotation>
      <xs:documentation>Corresponds to ApplicationBean

(Interface=com.bea.wcp.diameter.management.descriptor.beans.ApplicationBea
n)</xs:documentation>
    </xs:annotation>
    <xs:complexContent>
      <xs:extension base="ns:diameter-descriptorType"
xmlns:ns="http://www.bea.com/ns/wlcp/diameter/300">
        <xs:sequence>
          <xs:element name="class-name" type="xs:string" minOccurs="0"
nillable="true">
            <xs:annotation>

<xs:documentation>(Interface=com.bea.wcp.diameter.management.descriptor.be
ans.ApplicationBean Attribute=getClassName)</xs:documentation>
            </xs:annotation>
          </xs:element>
        <xs:element name="param" maxOccurs="unbounded" type="ns:paramType"
minOccurs="0" nillable="true">
            <xs:annotation>
```

```
<xs:documentation>(Interface=com.bea.wcp.diameter.management.descriptor.be
ans.ApplicationBean Attribute=getParams)</xs:documentation>
            </xs:annotation>
          </xs:element>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="peerType">
    <xs:annotation>
      <xs:documentation>Corresponds to PeerBean

(Interface=com.bea.wcp.diameter.management.descriptor.beans.PeerBean)</xs:
documentation>
    </xs:annotation>
    <xs:sequence>
      <xs:element name="host" type="xs:string" minOccurs="0"
nillable="true">
        <xs:annotation>

<xs:documentation>(Interface=com.bea.wcp.diameter.management.descriptor.be
ans.PeerBean Attribute=getHost)</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="address" type="xs:string" minOccurs="0"
nillable="true">
        <xs:annotation>
```

```
<xs:documentation>(Interface=com.bea.wcp.diameter.management.descriptor.be
ans.PeerBean Attribute=getAddress)</xs:documentation>
        </xs:annotation>
      </xs:element>
     <xs:element name="port" type="xs:int" minOccurs="0" nillable="false"
default="3588">
        <xs:annotation>

<xs:documentation>(Interface=com.bea.wcp.diameter.management.descriptor.be
ans.PeerBean Attribute=getPort)</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="protocol" minOccurs="0" nillable="true">
        <xs:annotation>

<xs:documentation>(Interface=com.bea.wcp.diameter.management.descriptor.be
ans.PeerBean Attribute=getProtocol)</xs:documentation>
        </xs:annotation>
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="tcp"/>
            <xs:enumeration value="sctp"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="watchdog-enabled" type="xs:boolean" minOccurs="0"
nillable="false" default="true">
        <xs:annotation>
```

```
<xs:documentation>(Interface=com.bea.wcp.diameter.management.descriptor.be
ans.PeerBean Attribute=getWatchdogEnabled)</xs:documentation>
        </xs:annotation>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="diameterType">
    <xs:annotation>
      <xs:documentation>Corresponds to DiameterBean

(Interface=com.bea.wcp.diameter.management.descriptor.beans.DiameterBean)<
/xs:documentation>
    </xs:annotation>
    <xs:complexContent>
      <xs:extension base="ns:diameter-descriptorType"
xmlns:ns="http://www.bea.com/ns/wlcp/diameter/300">
        <xs:sequence>
          <xs:element name="configuration" maxOccurs="unbounded"
type="ns:configurationType" minOccurs="0" nillable="true">
            <xs:annotation>

<xs:documentation>(Interface=com.bea.wcp.diameter.management.descriptor.be
ans.DiameterBean Attribute=getConfigurations)</xs:documentation>
            </xs:annotation>
          </xs:element>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
```

```
  <xs:complexType name="configurationType">
    <xs:annotation>
      <xs:documentation>Corresponds to ConfigurationBean

(Interface=com.bea.wcp.diameter.management.descriptor.beans.ConfigurationB
ean)</xs:documentation>
    </xs:annotation>
    <xs:complexContent>
      <xs:extension base="ns:diameter-descriptorType"
xmlns:ns="http://www.bea.com/ns/wlcp/diameter/300">
        <xs:sequence>
          <xs:element name="target" maxOccurs="unbounded" type="xs:string"
minOccurs="0" nillable="true">
            <xs:annotation>

<xs:documentation>(Interface=com.bea.wcp.diameter.management.descriptor.be
ans.ConfigurationBean Attribute=getTargets)</xs:documentation>
            </xs:annotation>
          </xs:element>
          <xs:element name="host" type="xs:string" minOccurs="0"
nillable="true">
            <xs:annotation>

<xs:documentation>(Interface=com.bea.wcp.diameter.management.descriptor.be
ans.ConfigurationBean Attribute=getHost)</xs:documentation>
            </xs:annotation>
          </xs:element>
          <xs:element name="realm" type="xs:string" minOccurs="0"
nillable="true">
            <xs:annotation>
```

```
<xs:documentation>(Interface=com.bea.wcp.diameter.management.descriptor.be
ans.ConfigurationBean Attribute=getRealm)</xs:documentation>
            </xs:annotation>
          </xs:element>
          <xs:element name="address" type="xs:string" minOccurs="0"
nillable="true">
            <xs:annotation>

<xs:documentation>(Interface=com.bea.wcp.diameter.management.descriptor.be
ans.ConfigurationBean Attribute=getAddress)</xs:documentation>
            </xs:annotation>
          </xs:element>
          <xs:element name="port" type="xs:int" minOccurs="0"
nillable="false">
            <xs:annotation>

<xs:documentation>(Interface=com.bea.wcp.diameter.management.descriptor.be
ans.ConfigurationBean Attribute=getPort)</xs:documentation>
            </xs:annotation>
          </xs:element>
          <xs:element name="tls-enabled" type="xs:boolean" minOccurs="0"
nillable="false">
            <xs:annotation>

<xs:documentation>(Interface=com.bea.wcp.diameter.management.descriptor.be
ans.ConfigurationBean Attribute=getTlsEnabled)</xs:documentation>
            </xs:annotation>
          </xs:element>
          <xs:element name="sctp-enabled" type="xs:boolean" minOccurs="0"
nillable="false">
```

```
            <xs:annotation>

<xs:documentation>(Interface=com.bea.wcp.diameter.management.descriptor.be
ans.ConfigurationBean Attribute=getSctpEnabled)</xs:documentation>

            </xs:annotation>

          </xs:element>

         <xs:element name="debug-enabled" type="xs:boolean" minOccurs="0"
nillable="false">

            <xs:annotation>

<xs:documentation>(Interface=com.bea.wcp.diameter.management.descriptor.be
ans.ConfigurationBean Attribute=getDebugEnabled)</xs:documentation>

            </xs:annotation>

          </xs:element>

          <xs:element name="message-debug-enabled" type="xs:boolean"
minOccurs="0" nillable="false">

            <xs:annotation>

<xs:documentation>(Interface=com.bea.wcp.diameter.management.descriptor.be
ans.ConfigurationBean Attribute=getMessageDebugEnabled)</xs:documentation>

            </xs:annotation>

          </xs:element>

          <xs:element name="application" maxOccurs="unbounded"
type="ns:applicationType" minOccurs="0" nillable="true">

            <xs:annotation>

<xs:documentation>(Interface=com.bea.wcp.diameter.management.descriptor.be
ans.ConfigurationBean Attribute=getApplications)</xs:documentation>

            </xs:annotation>

          </xs:element>
```

```
          <xs:element name="peer-retry-delay" type="xs:int" minOccurs="0"
nillable="false" default="30">
            <xs:annotation>

<xs:documentation>(Interface=com.bea.wcp.diameter.management.descriptor.be
ans.ConfigurationBean Attribute=getPeerRetryDelay)</xs:documentation>
            </xs:annotation>
          </xs:element>
          <xs:element name="allow-dynamic-peers" type="xs:boolean"
minOccurs="0" nillable="false">
            <xs:annotation>

<xs:documentation>(Interface=com.bea.wcp.diameter.management.descriptor.be
ans.ConfigurationBean Attribute=getAllowDynamicPeers)</xs:documentation>
            </xs:annotation>
          </xs:element>
          <xs:element name="request-timeout" type="xs:long" minOccurs="0"
nillable="false" default="30000">
            <xs:annotation>

<xs:documentation>(Interface=com.bea.wcp.diameter.management.descriptor.be
ans.ConfigurationBean Attribute=getRequestTimeout)</xs:documentation>
            </xs:annotation>
          </xs:element>
          <xs:element name="watchdog-timeout" type="xs:int" minOccurs="0"
nillable="false" default="30">
            <xs:annotation>

<xs:documentation>(Interface=com.bea.wcp.diameter.management.descriptor.be
ans.ConfigurationBean Attribute=getWatchdogTimeout)</xs:documentation>
            </xs:annotation>
```

```
          </xs:element>
          <xs:element name="include-origin-state-id" type="xs:boolean"
minOccurs="0" nillable="false">
            <xs:annotation>
<xs:documentation>(Interface=com.bea.wcp.diameter.management.descriptor.be
ans.ConfigurationBean
Attribute=getIncludeOriginStateId)</xs:documentation>
            </xs:annotation>
          </xs:element>
         <xs:element name="peer" maxOccurs="unbounded" type="ns:peerType"
minOccurs="0" nillable="true">
            <xs:annotation>
<xs:documentation>(Interface=com.bea.wcp.diameter.management.descriptor.be
ans.ConfigurationBean Attribute=getPeers)</xs:documentation>
            </xs:annotation>
          </xs:element>
        <xs:element name="route" maxOccurs="unbounded" type="ns:routeType"
minOccurs="0" nillable="true">
            <xs:annotation>
<xs:documentation>(Interface=com.bea.wcp.diameter.management.descriptor.be
ans.ConfigurationBean Attribute=getRoutes)</xs:documentation>
            </xs:annotation>
          </xs:element>
          <xs:element name="default-route" type="ns:default-routeType"
minOccurs="0" nillable="true">
            <xs:annotation>
```

```
<xs:documentation>(Interface=com.bea.wcp.diameter.management.descriptor.be
ans.ConfigurationBean Attribute=getDefaultRoute)</xs:documentation>
            </xs:annotation>
          </xs:element>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="routeType">
    <xs:annotation>
      <xs:documentation>Corresponds to RouteBean

(Interface=com.bea.wcp.diameter.management.descriptor.beans.RouteBean)</xs
:documentation>
    </xs:annotation>
    <xs:complexContent>
      <xs:extension base="ns:diameter-descriptorType"
xmlns:ns="http://www.bea.com/ns/wlcp/diameter/300">
        <xs:sequence>
          <xs:element name="realm" type="xs:string" minOccurs="0"
nillable="true">
            <xs:annotation>

<xs:documentation>(Interface=com.bea.wcp.diameter.management.descriptor.be
ans.RouteBean Attribute=getRealm)</xs:documentation>
            </xs:annotation>
          </xs:element>
          <xs:element name="application-id" type="xs:int" minOccurs="0"
nillable="false">
```

```
            <xs:annotation>

<xs:documentation>(Interface=com.bea.wcp.diameter.management.descriptor.be
ans.RouteBean Attribute=getApplicationId)</xs:documentation>
            </xs:annotation>
          </xs:element>
          <xs:element name="action" minOccurs="0" nillable="true">
            <xs:annotation>

<xs:documentation>(Interface=com.bea.wcp.diameter.management.descriptor.be
ans.RouteBean Attribute=getAction)</xs:documentation>
            </xs:annotation>
            <xs:simpleType>
              <xs:restriction base="xs:string">
                <xs:enumeration value="none"/>
                <xs:enumeration value="local"/>
                <xs:enumeration value="relay"/>
                <xs:enumeration value="proxy"/>
                <xs:enumeration value="redirect"/>
              </xs:restriction>
            </xs:simpleType>
          </xs:element>
         <xs:element name="server" maxOccurs="unbounded" type="xs:string"
minOccurs="0" nillable="true">
            <xs:annotation>

<xs:documentation>(Interface=com.bea.wcp.diameter.management.descriptor.be
ans.RouteBean Attribute=getServers)</xs:documentation>
            </xs:annotation>
```

```
          </xs:element>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="default-routeType">
    <xs:annotation>
      <xs:documentation>Corresponds to DefaultRouteBean

(Interface=com.bea.wcp.diameter.management.descriptor.beans.DefaultRouteBe
an)</xs:documentation>
    </xs:annotation>
    <xs:complexContent>
      <xs:extension base="ns:diameter-descriptorType"
xmlns:ns="http://www.bea.com/ns/wlcp/diameter/300">
        <xs:sequence>
          <xs:element name="action" type="xs:string" minOccurs="0"
nillable="true">
            <xs:annotation>

<xs:documentation>(Interface=com.bea.wcp.diameter.management.descriptor.be
ans.DefaultRouteBean Attribute=getAction)</xs:documentation>
            </xs:annotation>
          </xs:element>
         <xs:element name="server" maxOccurs="unbounded" type="xs:string"
minOccurs="0" nillable="true">
            <xs:annotation>

<xs:documentation>(Interface=com.bea.wcp.diameter.management.descriptor.be
ans.DefaultRouteBean Attribute=getServers)</xs:documentation>
```

```
            </xs:annotation>
          </xs:element>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:element name="diameter" type="ns:diameterType"
xmlns:ns="http://www.bea.com/ns/wlcp/diameter/300"/>
</xs:schema>
```

Step 5: Perform Optional Upgrade Tasks

After upgrading the WebLogic SIP Server configuration files, you can remove the older, version 2.2 `sipserver` application directory, as well as any application directories created for Diameter nodes in version 2.2.

Also, if you continue to use the version 2.2 example applications, note that some of the example `build.xml` files reference the files `wlss.jar` and `sipserver.jar`. These libraries are now located in *`WLSS_HOME`*`/server/lib/wlss`. Either update the older `build.xml` files or use the examples installed with WebLogic SIP Server 3.1.

APPENDIX B

Improving Failover Performance for Physical Network Failures

The following sections describe how to configure use the WebLogic SIP Server "echo server" process to improve data tier failover performance when a server becomes physically disconnected from the network:

Overview of Failover Detection

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

By default, an engine tier server uses its RMI connection to the replica to determine if the replica has failed or become disconnected. The algorithms used to determine a failure of an RMI connection are reliable, but ultimately they depend on the TCP protocol's retransmission timers to diagnose a disconnection (for example, if the network cable to the replica is removed). Because

the TCP retransmission timer generally lasts a full minute or longer, WebLogic SIP Server provides an alternate method of detecting failures that can diagnose a disconnected replica in a matter of a few seconds.

WlssEchoServer Failure Detection

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

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

Also, should a data tier server notice that its local `WlssEchoServer` process has died, it automatically shuts down. This behavior ensures even quicker failover because avoids the time it takes engine nodes to notice and report the failure as described in "Overview of Failover Detection" on page B-1.

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

Forced Shutdown for Failed Replicas

If any engine tier server cannot communicate with a particular replica, the engine access another, available replica in the data tier to report the offline server. The replica updates its view of the affected partition to remove the offline server. The updated view is then distributed to all engine

tier servers that later access the partition. Propagating the view in this manner helps to ensure that engine servers do not attempt to access the offline replica.

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

WlssEchoServer Requirements and Restrictions

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

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

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

Starting WlssEchoServer on Data Tier Server Machines

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

```
java -classpath WLSS_HOME/telco/lib/wlss.jar options
com.bea.wcp.util.WlssEchoServer
```

Where *WLSS_HOME* is the path to the WebLogic SIP Server installation and *options* may include one of the options described in Table B-1.

Table B-1 WlssEchoServer Options

Option	Description
`-Dwlss.ha.echoserver.port`	Specifies the port number used to listen for heartbeat messages. Ensure that the port number you specify is not used by any other process on the server machine. By default `WlssEchoServer` uses port 6734.
`-Dwlss.ha.echoserver.logfile`	Specifies the log file location and name. By default, log messages are written to `./echo_server`*time*`.log` where *time* is the time expressed in milliseconds.

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

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

to read:

```
"$JAVA_HOME/bin/java" -classpath WLSS_HOME/telco/lib/wlss.jar    \
  -Dwlss.ha.echoserver.port=6734 com.bea.wcp.util.WlssEchoServer &
"$JAVA_HOME/bin/java" ${JAVA_VM} ${MEM_ARGS} ${JAVA_OPTIONS}     \
  -Dweblogic.Name=${SERVER_NAME}                                  \
  -Dweblogic.management.username=${WLS_USER}                      \
```

```
-Dweblogic.management.password=${WLS_PW}                          \
-Dweblogic.management.server=${ADMIN_URL}                         \
-Djava.security.policy="${WL_HOME}/server/lib/weblogic.policy" \
 weblogic.Server
```

Enabling and Configuring the Heartbeat Mechanism on Servers

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

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

to read:

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

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

Table B-2 WlssEchoServer Options

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

APPENDIX C

Tuning JVM Garbage Collection for Production Deployments

The following sections describe how to tune Java Virtual Machine (JVM) garbage collection performance for engine tier servers:

Goals for Tuning Garbage Collection Performance

Production installations of WebLogic SIP Server generally require extremely small response times (under 50 milliseconds) for clients at all times, even under peak server loads. A key factor in maintaining brief response times is the proper selection and tuning of the JVM's Garbage Collection (GC) algorithm for WebLogic SIP Server instances in the engine tier.

Whereas certain tuning strategies are designed to yield the lowest average garbage collection times or to minimize the frequency of full GCs, those strategies can sometimes result in one or more very long periods of garbage collection (often several seconds long) that are offset by shorter GC intervals. With a production SIP Server installation, all long GC intervals must be avoided in order to maintain response time goals.

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

Modifying JVM Parameters in Server Start Scripts

If you use custom startup scripts to start WebLogic SIP Server engines and replicas, simply edit those scripts to include the recommended JVM options described in the sections that follow.

The BEA Configuration Wizard also installs default startup scripts when you configure a new domain. These scripts are installed in the *BEA_HOME*/user_projects/domains/*domain_name*/bin directory by default, and include:

- `startWebLogic.cmd`, `startWebLogic.sh`—These scripts start the Administration Server for the domain.
- `startManagedWebLogic.cmd`, `startManagedWebLogic.sh`—These scripts start managed engines and replicas in the domain.

If you use the BEA-installed scripts to start engines and replicas, you can override JVM memory arguments by first setting the `USER_MEM_ARGS` environment variable in your command shell.

Note: Setting the `USER_MEM_ARGS` environment variable overrides all default JVM memory arguments specified in the BEA-installed scripts. Always set `USER_MEM_ARGS` to the full list of JVM memory arguments you intend to use. For example, when using the Sun JVM, always add `-XX:MaxPermSize=128m` to the `USER_MEM_ARGS` value, even if you only intend to change the default heap space (`-Xms, -Xmx`) parameters.

Tuning Garbage Collection with JRockit

JRockit provides several monitoring tools that you can use to analyze the JVM heap at any given moment, including:

- JRockit Runtime Analyzer—provides a view into the runtime behavior of garbage collection and pause times.
- JRockit Stack Dumps—reveals applications' thread activity to help you troubleshoot and/or improve performance.

Use these and other tools in a controlled environment to determine the effects of JVM settings before you use the settings in a production deployment. See the BEA WebLogic JRockit 1.4.2 SDK Documentation for more information about JRockit and JRockit profiling tools.

The following sections describe suggested starting JVM options for use with the JRockit JVM. The JRockit JVM is available in a standard edition (included with WebLogic SIP Server) or as part of BEA WebLogic Real Time, available separately. The standard JRockit JVM can be tuned

to provide high throughput and low response times using the information in "Using JRockit without Deterministic Garbage Collection" on page C-3.

The version of JRockit included with BEA WebLogic Real Time uses a deterministic garbage collector, and is recommended when extremely low latency is the overriding requirement of your application. The deterministic garbage collector attempts to maximize throughput, but places the highest priority on guaranteeing short, predictable pause times. This focus on guaranteeing short pause times may result in lower overall throughput when compared to the standard JRockit JVM. See "Using JRockit with Deterministic Garbage Collection (WebLogic Real Time)" on page C-3 for more information.

Using JRockit without Deterministic Garbage Collection

When using BEA's JRockit JVM without deterministic garbage collection (the version included with WebLogic SIP Server), the best response time performance is obtained by using the generational concurrent garbage collector.

The full list of example startup options for an engine tier server are:

```
-Xms1024m -Xmx1024m -Xgc:gencon -XXnosystemgc -XXtlasize:min=3k
-XXkeeparearatio=0 -Xns:48m
```

Note: Fine tune the heap size according to the amount of live data used by deployed applications.

The full list of example startup options for a replica server are:

```
-Xms3072m -Xmx3072m -Xgc:gencon -XXnosystemgc -XXtlasize:min=3k
-XXkeeparearatio=0 -Xns:48m
```

Using JRockit with Deterministic Garbage Collection (WebLogic Real Time)

Very short response times are most easily achieved by using JRockit's deterministic garbage collector, which is available with the WebLogic Real Time product. See JVM Tuning for Real-Time Applications in the WebLogic Real Time documentation for basic information about tuning with deterministic garbage collection.

BEA recommends using the following JVM arguments for engine tier servers in replicated cluster configurations:

```
-Xms1024m -Xmx1024m -XgcPrio:deterministic -XpauseTarget=30ms -XXnosystemgc
```

Note: You may need to increase the `-XpauseTarget` value for allocation-intensive applications. The value can be decreased for smaller applications under light loads.

Note: Adjust the heap size according to the amount of live data used by deployed applications. As a starting point, set the heap size from 2 to 3 times the amount required by your applications. A value closer to 3 times the required amount generally yields the best performance.

For replica servers, use the arguments:

```
-Xms3072m -Xmx3072m -XgcPrio:deterministic -XpauseTarget=30ms -XXnosystemgc
```

These settings fix the heap size and enable the dynamic garbage collector with deterministic garbage collection. `-XpauseTarget` sets the maximum pause time and `-XXtlasize=3k` sets the thread-local area size. `-XXnosystemgc` prevents `System.gc()` application calls from forcing garbage collection.

Tuning Garbage Collection with Sun JDK

When using Sun's JDK, the goal in tuning garbage collection performance is to reduce the time required to perform a full garbage collection cycle. You should not attempt to tune the JVM to minimize the frequency of full garbage collections, because this generally results in an eventual forced garbage collection cycle that may take up to several full seconds to complete.

The simplest and most reliable way to achieve short garbage collection times over the lifetime of a production server is to use a fixed heap size with the default collector and the parallel young generation collector, restricting the new generation size to at most one third of the overall heap.

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

```
-server -Xmx1024m -XX:MaxPermSize=128m -XX:+UseParNewGC
-XX:+UseConcMarkSweepGC -XX:+UseTLAB -XX:+CMSIncrementalMode
-XX:+CMSIncrementalPacing -XX:CMSIncrementalDutyCycleMin=0
-XX:CMSIncrementalDutyCycle=10 -XX:MaxTenuringThreshold=0
-XX:SurvivorRatio=256 -XX:CMSInitiatingOccupancyFraction=60
-XX:+DisableExplicitGC
```

For replica servers, use the example settings:

```
-server -Xmx3072m -XX:MaxPermSize=128m -XX:+UseParNewGC
-XX:+UseConcMarkSweepGC -XX:+UseTLAB -XX:+CMSIncrementalMode
-XX:+CMSIncrementalPacing -XX:CMSIncrementalDutyCycleMin=0
-XX:CMSIncrementalDutyCycle=10 -XX:MaxTenuringThreshold=0
```

```
-XX:SurvivorRatio=256 -XX:CMSInitiatingOccupancyFraction=60
-XX:+DisableExplicitGC
```

The above options have the following effect:

- `-XX:+UseTLAB`—Uses thread-local object allocation blocks. This improves concurrency by reducing contention on the shared heap lock.
- `-XX:+UseParNewGC`—Uses a parallel version of the young generation copying collector alongside the concurrent mark-and-sweep collector. This minimizes pauses by using all available CPUs in parallel. The collector is compatible with both the default collector and the Concurrent Mark and Sweep (CMS) collector.
- `-Xms, -Xmx`—Places boundaries on the heap size to increase the predictability of garbage collection. The heap size is limited in replica servers so that even Full GCs do not trigger SIP retransmissions. `-Xms` sets the starting size to prevent pauses caused by heap expansion.
- `-XX:MaxTenuringThreshold=0`—Makes the full NewSize available to every NewGC cycle, and reduces the pause time by not evaluating tenured objects. Technically, this setting promotes all live objects to the older generation, rather than copying them.
- `-XX:SurvivorRatio=128`—Specifies a high survivor ratio, which goes along with the zero tenuring threshold to ensure that little space is reserved for absent survivors.

APPENDIX D

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=file:/dev/random
   ```

   to read:

   ```
   securerandom.source=file:/dev/urandom
   ```
3. Save your change and exit the text editor.