BEA WebLogic SIP Server™

Operations Guide

Version 3.1
Revised: August 8, 2007

Copyright

Restricted Rights Legend

Trademarks and Service Marks

Contents

1. Starting and Stopping Servers

Startup Sequence for a WebLogic SIP Server Domain 1-1

Administration Server Best Practices 1-2

2. Configuring SNMP

Overview of WebLogic SIP Server SNMP 2-1

Browsing the MIB 2-2

Configuring SNMP 2-2

SNMP Port Binding for WebLogic SIP Server 2-2

Understanding and Responding to SNMP Traps 2-3

Files for Troubleshooting 2-3

Trap Descriptions 2-4

connectionLostToPeer 2-4

connectionReestablishedToPeer 2-5

dataTierServerStopped 2-5

overloadControlActivated, overloadControlDeactivated 2-5

replicaAddedToPartition 2-6

replicaRemovedFromPartition 2-6

serverStopped 2-7

sipAppDeployed 2-8

sipAppUndeployed 2-8

sipAppFailedToDeploy 2-9

3. Using the WebLogic Server Diagnostic Framework (WLDF)

Overview of WebLogic SIP Server and the WLDF 3-1

Data Collection and Logging 3-2

Watches and Notifications 3-3
Image Capture 3-3
Instrumentation 3-4
Configuring Server-Scoped Monitors 3-6
Configuring Application-Scoped Monitors 3-7

4. Logging SIP Requests and Responses

Overview of SIP Logging 4-1
Defining Logging Servlets in sip.xml 4-2
Configuring the Logging Level and Destination. 4-3
Specifying the Criteria for Logging Messages 4-5
Using XML Documents to Specify Logging Criteria 4-5
Using Servlet Parameters to Specify Logging Criteria 4-6
Specifying Content Types for Unencrypted Logging 4-8
Managing Logging Performance. 4-9
Enabling Log Rotation and Viewing Log Files 4-10
trace-pattern.dtd Reference 4-10
Adding Tracing Functionality to SIP Servlet Code 4-14
Order of Startup for Listeners and Logging Servlets 4-15

5. Applying Patches

6. Upgrading Deployed SIP Applications

Overview of SIP Application Upgrades 6-1
Requirements and Restrictions for Upgrading Deployed Applications 6-2
Steps for Upgrading a Deployed SIP Application 6-3
Assign a Version Identifier 6-4
Defining the Version in the Manifest 6-4
Deploy the Updated Application Version 6-5

Undeploy the Older Application Version 6-5
Roll Back the Upgrade Process 6-7
Accessing the Application Name and Version Identifier 6-7
Using Administration Mode 6-7

7. Avoiding and Recovering From Server Failures

Failure Prevention and Automatic Recovery Features 7-1
Overload Protection 7-2
Redundancy and Failover for Clustered Services 7-2
Automatic Restart for Failed Server Instances 7-3
Managed Server Independence Mode 7-3
Automatic Migration of Failed Managed Servers 7-3
Geographic Redundancy for Regional Site Failures 7-4
Directory and File Backups for Failure Recovery 7-4
Enabling Automatic Configuration Backups 7-5
Storing the Domain Configuration Offline 7-6
Backing Up Server Start Scripts 7-6
Backing Up Logging Servlet Applications 7-7
Backing Up Security Data 7-7
Backing Up the WebLogic LDAP Repository 7-7
Backing Up SerializedSystemIni.dat and Security Certificates 7-8
Backing Up Additional Operating System Configuration Files 7-8
Restarting a Failed Administration Server 7-8
Restarting an Administration Server on the Same Machine 7-9
Restarting an Administration Server on Another Machine 7-9
Restarting Failed Managed Servers 7-10

8. Upgrading WebLogic SIP Server Software

Overview of System Upgrades 8-1

Requirements for Upgrading a Production System 8-2

Upgrading to a New Version of WebLogic SIP Server 8-3

- Configure the Load Balancer 8-3
- Configure the New Engine Tier Cluster 8-4
- Define the Cluster-to-Load Balancer Mapping 8-5
- Duplicate the SIP Servlet Configuration 8-6
- Upgrade Engine Tier Servers and Target Applications to the New Cluster 8-6
- Upgrade Data Tier Servers 8-8

CHAPTER 1

Starting and Stopping Servers

The following sections describe how to start and stop servers in a WebLogic SIP Server domain:

- "Startup Sequence for a WebLogic SIP Server Domain" on page 1-1
- "Administration Server Best Practices" on page 1-2

Startup Sequence for a WebLogic SIP Server Domain

Note: WebLogic SIP Server start scripts use default values for many JVM parameters that affect performance. For example, JVM garbage collection and heap size parameters may be omitted, or may use values that are acceptable only for evaluation or development purposes. In a production system, you must rigorously profile your applications with different heap size and garbage collection settings in order to realize adequate performance. See "Tuning JVM Garbage Collection for Production Deployments" on page C-1 for suggestions about maximizing JVM performance in a production domain.

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

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

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

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

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

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

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

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

Administration Server Best Practices

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

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

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

- Run the Administration Server instance on a dedicated machine. The Administration Server machine should have a memory capacity similar to Managed Server machines, although a single CPU is generally acceptable for administration purposes.
- Configure all Managed Server instances to use Managed Server Independence. This feature allows the Managed Servers to restart even if the Administration Server is unreachable due to a network, hardware, or software failure. See Replicate domain config files for Managed Server independence in the WebLogic Server 9.2 documentation.
- Configure the Node Manager utility to automatically restart all Managed Servers in the WebLogic SIP Server domain. See Using Node Manager to Control Servers in the WebLogic Server 9.2 documentation.

Should an Administration Server instance or machine fail, remember that only configuration, deployment, and monitoring features are affected, but Managed Servers continue to operate and process client requests. Potential losses incurred due to an Administration Server failure include:

- Loss of in-progress management and deployment operations.
- Loss of ongoing logging functionality.
- Loss of SNMP trap generation for WebLogic Server instances (as opposed to WebLogic SIP Server instances). On Managed Servers, WebLogic SIP Server traps are generated even in the absence of the Administration Server.

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

CHAPTER 2

Configuring SNMP

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

- "Overview of WebLogic SIP Server SNMP" on page 2-1
- "Browsing the MIB" on page 2-2
- "Configuring SNMP" on page 2-2
- "SNMP Port Binding for WebLogic SIP Server" on page 2-2
- "Understanding and Responding to SNMP Traps" on page 2-3

Overview of WebLogic SIP Server SNMP

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

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

Browsing the MIB

Use a Web browser to view the WebLogic SIP Server SNMP MIB Reference on the BEA e-docs Web site. Because the MIB Reference uses Javascript and DHTML to provide browsing capabilities that are similar to a MIB browser, you must use one of the following Web browsers:

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

Configuring SNMP

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

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

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

SNMP Port Binding for WebLogic SIP Server

If you run multiple Managed Server instances on the same machine, each server instance would normally attempt to bind to the same configured SNMP port (for example, port 161). WebLogic SIP Server instances automatically manage SNMP port conflicts by automatically attempting to bind to port 1610, and incrementing the port number as needed if the current port is unavailable. This helps to avoid a SNMP startup failure when multiple WebLogic SIP Server instances are deployed on the same server hardware.

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

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

Understanding and Responding to SNMP Traps

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

Files for Troubleshooting

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

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

General information that can help the technical support team includes:

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

Trap Descriptions

Table 2-1 lists the WebLogic SIP Server SNMP traps and indicates whether the trap is generated by servers in the engine tier or data tier. Each trap is described in the sections that follow.

Table 2-1 WebLogic SIP Server SNMP Traps

Server Node in which Trap is Generated	Trap Name
Engine Tier Servers	"connectionLostToPeer" on page 2-4
	"connectionReestablishedToPeer" on page 2-5
	"overloadControlActivated, overloadControlDeactivated" on page 2-5
	"sipAppDeployed" on page 2-8
	"sipAppUndeployed" on page 2-8
	"sipAppFailedToDeploy" on page 2-9
Engine and Data Tier Servers, if servers are members of a cluster	"serverStopped" on page 2-7
Data Tier Servers	"dataTierServerStopped" on page 2-5
	"replicaAddedToPartition" on page 2-6
	"replicaRemovedFromPartition" on page 2-6

connectionLostToPeer

Description

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

Recovery Procedure

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

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

connectionReestablishedToPeer

Description

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

Recovery Procedure

See "connectionLostToPeer" on page 2-4.

dataTierServerStopped

Description

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

Recovery Procedure

See the Recovery Procedure for "serverStopped" on page 2-7.

overloadControlActivated, overloadControlDeactivated

Description

Weblogic SIP Server engine tier nodes use a configurable throttling mechanism that helps you control the number of new SIP requests that are processed. After a configured overload condition is observed, WebLogic SIP Server destroys new SIP requests by responding with "503 Service Unavailable" to the caller. The server continues to destroy new requests until the overload

condition is resolved according to a configured threshold control value. This alarm is generated when the throttling mechanism is activated. The throttling behavior should eventually return the server to a non-overloaded state, and further action may be unnecessary. See Overload in the *Configuration Reference Manual*.

Recovery Procedure

1. Check other servers to see if they are nearly overloaded.

2. Check to see if the load balancer is correctly balancing load across the application servers, or if it is overloading one or more servers. If additional servers are nearly overloaded, Notify Tier 4 support immediately.

3. If the issue is limited to one server, notify Tier 4 support within one hour.

Additional Overload FAQs

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

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

replicaAddedToPartition

Description

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

Recovery Procedure

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

replicaRemovedFromPartition

Description

WebLogic SIP Server data tier nodes generate this alarm when a server is removed from the data tier, either as a result of a normal shutdown operation or because of a failure. There must be at least one replica remaining in a partition to generate this trap; if a partition has only a single

replica and that replica fails, the trap cannot be generated. In addition, because engine tier nodes determine when a replica has failed, an engine tier node must be running in order for this trap to be generated.

Recovery Procedure

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

serverStopped

Description

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

Recovery Procedure

1. Use the following command to identify the hung process:

   ```
   ps –ef | grep java
   ```

   There should be only one PID for each WebLogic Server instance running on the machine.

2. After identifying the affected PID, use the following command to kill the process:

   ```
   kill -3 [pid]
   ```

3. This command generates the actual thread dump. If the process is not immediately killed, repeat the command several times, spaced 5-10 seconds apart, to help diagnose potential deadlock problems, until the process is killed.

4. Attempt to restart WebLogic SIP Server immediately. See Restarting a Failed Managed Server in the WebLogic Server 9.2 documentation.

5. Make a backup copy of all SIP logs on the affected server to aid in troubleshooting. The location of the logs varies based on the server configuration.

6. Copy each log to assist Tier 4 support with troubleshooting the problem.

   Note: WebLogic SIP Server logs are truncated according to your system configuration. Make backup logs immediately to avoid losing critical troubleshooting information.

7. Notify Tier 4 support and include the log files with the trouble ticket.
8. Monitor the server closely over next 24 hours. If the source of the problem cannot be identified in the log files, there may be a hardware or network issue that will reappear over time.

Additional Shutdown FAQs

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

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

sipAppDeployed

Description

WebLogic SIP Server engine tier nodes generate this alarm when a SIP Servlet is deployed to the container.

Recovery Procedure

This trap is generated during normal deployment operations and does not indicate an exception.

sipAppUndeployed

Description

WebLogic SIP Server engine tier nodes generate this alarm when a SIP application shuts down, or if a SIP application is undeployed. This generally occurs when WebLogic SIP Server is shutdown while active requests still exist.

Recovery Procedure

During normal shutdown procedures this alarm should be filtered out and should not reach operations. If the alarm occurs during the course of normal operations, it indicates that someone has shutdown the application or server unexpectedly, or there is a problem with the application. Notify Tier 4 support immediately.

sipAppFailedToDeploy

Description

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

Recovery Procedure

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

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

CHAPTER 3

Using the WebLogic Server Diagnostic Framework (WLDF)

The following sections describe how WebLogic SIP Server is integrated with the WebLogic Diagnostic Framework (WLDF):

- "Overview of WebLogic SIP Server and the WLDF" on page 3-1
- "Data Collection and Logging" on page 3-2
- "Watches and Notifications" on page 3-3
- "Image Capture" on page 3-3
- "Instrumentation" on page 3-4
  - "Configuring Server-Scoped Monitors" on page 3-6
  - "Configuring Application-Scoped Monitors" on page 3-7

Overview of WebLogic SIP Server and the WLDF

The WebLogic Diagnostic Framework (WLDF) consists of a number of components that work together to collect, archive, and access diagnostic information about a WebLogic Server instance and its applications. WebLogic SIP Server version 3.1 integrates with several components of the WLDF in order to monitor and diagnose the operation of engine and data tier nodes, as well as deployed SIP Servlets:

- Data Collectors—WebLogic SIP Server integrates with the Harvester service to collect information from runtime MBeans, and with the Logger service to archive SIP requests and responses.

- Watches and Notifications—Administrators can use the Watches and Notifications component to create complex rules, based on WebLogic SIP Server runtime MBean attributes, that trigger automatic notifications using JMS, JMX, SNMP, SMTP, and so forth.
- Image Capture—WebLogic SIP Server instances can collect certain diagnostic data and write the data to an image file when requested by an Administrator. This data can then be used to diagnose problems in a running server.
- Instrumentation—WebLogic SIP Server instruments the server and application code with monitors to help you configure diagnostic actions that are performed on SIP messages (requests and responses) that match certain criteria.

The sections that follow provide more details about how WebLogic SIP Server integrates with each of the above WLDF components. See Configuring and Using the WebLogic Diagnostics Framework in the WebLogic Server 9.2 documentation for more information about how the WLDF works, and how to configure different WLDF components. See also Using the WebLogic Diagnostic Framework Console Extension in the WebLogic Server 9.2 documentation for information about using the WLDF console extension to collect and monitor diagnostic information.

Data Collection and Logging

WebLogic SIP Server uses the WLDF Harvester service to collect data from the attributes of these runtime MBeans:

- `ReplicaRuntimeMBean`
- `SipApplicationRuntimeMBean`
- `SipServerRuntimeMBean`

You can add charts and graphs of this data to your own custom views using the WLDF console extension. To do so, first enable the WLDF console extension by copying the JAR file into the `console-ext` subdirectory of your domain directory:

```
cp
~/bea/sipserver31/server/lib/console-ext/diagnostics-console-extension.jar
~/bea/user_projects/domains/mydomain/console-ext
```

When accessing the WLDF console extension, the WebLogic SIP Server runtime MBean attributes are available in the Metrics tab of the extension. See Using the WebLogic Diagnostic Framework Console Extension in the WebLogic Server 9.2 documentation for more information.

WebLogic SIP Server also uses the WLDF Logger service to archive SIP and Diameter messages to independent, dedicated log files. In previous releases, log information was written to a common log file, along with general WebLogic Server log messages. WebLogic SIP Server 3.1 now logs messages to a dedicated log file (by default, *domain_home*/logs/*server_name*/sipMessages.log). You can configure the name and location of the log file, as well as log rotation policies, using the Configuration->Message Debug tab in the SIP Server Administration Console extension. See Enabling Message Logging in *Developing Applications with WebLogic SIP Server*. Note that a server restart is necessary in order to initiate independent logging and log rotation.

Watches and Notifications

The data collected from WebLogic SIP Server runtime MBeans can be used to create automated monitors, or "watches," that observe a server's diagnostic state. One or more notifications can then be configured for use by a watch, in order to generate a message using SMTP, SNMP, JMX, or JMS when your configured watch conditions and rules occur.

To use watches and notifications, you select the Diagnostics->Diagnostic Modules node in the left pane of the Administration Console and create a new module with the watch rules and notifications required for monitoring your servers. The watch rules can use the metrics collected from WebLogic SIP Server runtime MBeans, messages written to the log file, or events generated by the diagnostic framework. See Configure watches and notifications in the WebLogic SIP Server 9.2 Administration Console Help for more information.

Image Capture

WebLogic SIP Server 3.1 adds its own image capture information to the diagnostic image generated by the WLDF. You can generate diagnostic images either on demand, or automatically by configuring watch rules.

The information contained in diagnostic images is intended for use by BEA technical support personnel when troubleshooting a potential server problem and includes:

- Data tier partition and replica configuration.
- Call state and timer statistics.
- Work manager statistics.

See Configure and capture diagnostic images in the WebLogic SIP Server 9.2 Administration Console Help for more information on generating diagnostic images.

Instrumentation

The WLDF instrumentation system creates diagnostic monitors and inserts them into WebLogic SIP Server or application code at specific points in the flow of execution. WebLogic SIP Server integrates with the instrumentation service to provide a built-in DyeInjection monitor. When enabled, this monitor injects dye flags into the diagnostic context when certain SIP messages enter or exist the system. Dye flags are injected based on the monitor's configuration properties, and on certain request attributes.

WebLogic SIP Server adds the dye flags described in Table 3-1 below, as well as the WebLogic Server dye flags USER and ADDR. The available WebLogic Server dye flags are described in Dyes Supported by the DyeInjection Monitor in the WebLogic Server 9.2 documentation.

Table 3-1 WebLogic SIP Server DyeInjection Flags

Dye Flag	Description
PROTOCOL_SIP	This flag is set in the diagnostic context of all SIP protocol messages.
SIP_REQ	This flag is set in the diagnostic context for all SIP requests that match the value of the property SIP_REQ.
SIP_RES	This flag is set in the diagnostic context for all SIP responses that match the value of the property SIP_RES.
SIP_REQURI	This flag is set if a SIP request's request URI matches the value of property SIP_REQURI.
SIP_ANY_HEADER	This flag is set if a SIP request contains a header matching the value of the property SIP_ANY_HEADER. The value of SIP_ANY_HEADER is specified using the format *`messageType.headerName=headerValue`* where *`headerValue`* is either a value or regular expression. For example, you can specify the property as SIP_ANY_HEADER=request.Contact=sip:sipp@localhost:5061 or SIP_ANY_HEADER=response.Contact=sip:findme@172.17.30.50:5060.

Dye flags can be applied to both incoming and outbound SIP messages. The flags are useful for dye filtering, and can be used by delegating monitors to trigger further diagnostic actions. See Configuring the DyeInjection Monitor to Manage Diagnostic Contexts in the WebLogic Server 9.2 documentation for information about configuring the monitor.

WebLogic SIP Server provides several delegating monitors that can be applied at the application and server scope, and which may examine dye flags set by the DyeInjection monitor. The delegating monitors are described in Table 3-2.

Table 3-2 WebLogic SIP Server Diagnostic Monitors

Monitor Name	Monitor Type	Scope	Pointcuts
Sip_Servlet_Before_Service	Before	Application	At entry of `SipServlet.do*` or `SipServlet.service` methods of all implementing subclasses.
Sip_Servlet_After_Service	After	Application	At exit of `SipServlet.do*` or `SipServlet.service` methods of all implementing subclasses.
Sip_Servlet_Around_Service	Around	Application	At entry and exit of `SipServlet.do*` or `SipServlet.service` methods of all implementing subclasses.
Sip_Servlet_Before_Session	Before	Application	At entry of `getAttribute`, `set`, `remove`, and `invalidate` methods for both `SipSession` and `SipApplicationSession`.
Sip_Servlet_After_Session	After	Application	At exit of `getAttribute`, `set`, `remove`, and `invalidate` methods for both `SipSession` and `SipApplicationSession`.
Sip_Servlet_Around_Session	Around	Application	At entry and exit of `getAttribute`, `set`, `remove`, and `invalidate` methods for both `SipSession` and `SipApplicationSession`.
Sip_Servlet_Before_Message_Send_Internal	Before	Server	At entry of WebLogic SIP Server code that writes messages to the wire.
Sip_Servlet_After_Message_Send_Internal	After	Server	At exit of WebLogic SIP Server code that writes messages to the wire.
Sip_Servlet_Around_Message_Send_Internal	Around	Server	At entry and exit of WebLogic SIP Server code that writes messages to the wire.

Configuring Server-Scoped Monitors

To use the server-scoped monitors, you must create a new diagnostic module and create and configure one or more monitors in the module. For the built-in DyeInjection monitor, you then add monitor properties to define the specific dye flags. For delegating monitors such as Sip_Servlet_Before_Message_Send_Internal, you add monitor properties to define diagnostic actions.

Follow these steps to configure the WebLogic SIP Server DyeInjection monitor, a delegate monitor, and enable dye filtering:

1. Access the Administration Console for you domain.
2. Click Lock & Edit to start a new configuration session.
3. Select the Diagnostics->Diagnostic Modules node in the left pane of the console.
4. Click New to create a new Diagnostic Module. Give the module a descriptive name, such as “instrumentationModule,” and click OK.
5. Select the new “instrumentationModule” from the list of modules in the table.
6. Select the Targets tab.
7. Select a server on which to target the module and click Save.
8. Return to the Diagnostics->Diagnostic Modules node and select instrumentationModule from the list of modules.
9. Select the Configuration->Instrumentation tab.
10. Select Enabled to enable instrumentation at the server level, then click Save.
11. Add the DyeInjection monitor to the module:
    a. Click Add/Remove.
    b. Select the name of a monitor from the Available list (for example, DyeInjection), and use the arrows to move it to the Chosen list.
    c. Click OK.
    d. Select the newly-created monitor from the list of available monitors.
    e. Ensure that the monitor is enabled, and edit the Properties field to add any required properties. For the DyeInjection monitor, sample properties include:

```
SIP_RES=180

SIP_REQ=INVITE

SIP_ANY_HEADER=request.Contact=sip:sipp@localhost:5061
```

f. Click Save

12. Add one or more delegate monitors to the module:

a. Click Add/Remove.

b. Select the name of a delegate monitor from the Available list (for example, Sip_Servlet_Before_Message_Send_Internal), and use the arrows to move it to the Chosen list.

c. Click OK.

d. Select the newly-created monitor from the list of available monitors.

e. Ensure that the monitor is enabled, then select one or more Actions from the available list, and use the arrows to move the actions to the Chosen list. For the Sip_Servlet_Before_Message_Send_Internal monitor, sample actions include the DisplayArgumentsAction, StackDumpAction, ThreadDumpAction, and TraceAction.

f. Select the check box to EnableDyeFiltering.

g. Select one or more Dye Masks, such as SIP_REQ, from the Available list and use the arrows to move them to the Chosen list.

h. Click Save

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

13. Click Activate Changes to apply your changes.

Configuring Application-Scoped Monitors

You configure application-scoped monitors in an XML configuration file named `weblogic-diagnostics.xml`. You must store the `weblogic-diagnostics.xml` file in the SIP module's or enterprise application's `META-INF` directory.

The XML file enables instrumentation at the application level, defines point cuts, and also defines delegate monitor dye masks and actions. Listing 3-1 shows a sample configuration file that uses the Sip_Servlet_Before_Service monitor.

Listing 3-1 Sample weblogic-diagnostics.xml File

```
<wldf-resource xmlns="http://www.bea.com/ns/weblogic/90/diagnostics">
  <instrumentation>
    <enabled>true</enabled>
    <include>demo.ProxyServlet</include>
    <wldf-instrumentation-monitor>
      <name>Sip_Servlet_Before_Service</name>
      <enabled>true</enabled>
      <dye-mask>SIP_ANY_HEADER</dye-mask>
      <dye-filtering-enabled>true</dye-filtering-enabled>
      <action>DisplayArgumentsAction</action>
    </wldf-instrumentation-monitor>
   </instrumentation>
</wldf-resource>
```

In this example, if an incoming request's diagnostic context contains the `SIP_ANY_HEADER` dye flag, then the `Sip_Servlet_Before_Service` monitor is triggered and the `DisplayArgumentsAction` is executed.

See Configuring Application-Scoped Instrumentation in the WebLogic Server 9.2 documentation for more information about creating the `weblogic-diagnostics.xml` configuration file.

CHAPTER 4

Logging SIP Requests and Responses

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

- "Overview of SIP Logging" on page 4-1
- "Defining Logging Servlets in sip.xml" on page 4-2
- "Configuring the Logging Level and Destination" on page 4-3
- "Specifying the Criteria for Logging Messages" on page 4-5
- "Specifying Content Types for Unencrypted Logging" on page 4-8
- "Managing Logging Performance" on page 4-9
- "Enabling Log Rotation and Viewing Log Files" on page 4-10
- "trace-pattern.dtd Reference" on page 4-10
- "Adding Tracing Functionality to SIP Servlet Code" on page 4-14
- "Order of Startup for Listeners and Logging Servlets" on page 4-15

Overview of SIP Logging

WebLogic SIP Server enables you to perform Protocol Data Unit (PDU) logging for the SIP requests and responses it processes. Logged SIP messages are placed either in the domain-wide log file for WebLogic SIP Server, or in the log files for individual Managed Server instances.

Because SIP messages share the same log files as WebLogic SIP Server instances, you can use advanced server logging features such as log rotation, domain log filtering, and maximum log size configuration when managing logged SIP messages.

Administrators configure SIP PDU logging by defining one or more SIP Servlets using the `com.bea.wcp.sip.engine.tracing.listener.TraceMessageListenerImpl` class. Logging criteria are then configured either as parameters to the defined servlet, or in separate XML files packaged with the application.

As SIP requests are processed or SIP responses generated, the logging Servlet compares the message with the filtering patterns defined in a standalone XML configuration file or Servlet parameter. SIP requests and responses that match the specified pattern are written to the log file along with the name of the logging servlet, the configured logging level, and other details. To avoid unnecessary pattern matching, the Servlet marks new SIP Sessions when an initial pattern is matched and then logs subsequent requests and responses for that session automatically.

Logging criteria are defined either directly in `sip.xml` as parameters to a logging Servlet, or in external XML configuration files. See "Specifying the Criteria for Logging Messages" on page 4-5.

Note: Engineers can implement PDU logging functionality in their Servlets either by creating a delegate with the `TraceMessageListenerFactory` in the Servlet's `init()` method, or by using the tracing class in deployed Java applications. Using the delegate enables you to perform custom logging or manipulate incoming SIP messages using the default trace message listener implementation. See "Adding Tracing Functionality to SIP Servlet Code" on page 4-14 for an example of using the factory in a Servlet's `init()` method.

Defining Logging Servlets in sip.xml

Logging Servlets for SIP messages are created by defining Servlets having the implementation class `com.bea.wcp.sip.engine.tracing.listener.TraceMessageListenerImpl`. The definition for a sample `msgTraceLogger` is shown in Listing 4-1.

Listing 4-1 Sample Logging Servlet

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

```
    <init-param>
      <param-name>domain</param-name>
      <param-value>true</param-value>
    </init-param>
    <init-param>
      <param-name>level</param-name>
      <param-value>full</param-value>
    </init-param>
    <load-on-startup/>
  </servlet>
```

Configuring the Logging Level and Destination

Logging attributes such as the level of logging detail and the destination log file for SIP messages are passed as initialization parameters to the logging Servlet. Table 4-1 lists the parameters and parameter values that you can specify as `init-param` entries. Listing 4-1, "Sample Logging

Servlet," on page 4-2 shows the sample `init-param` entries for a Servlet that logs full SIP message information to the domain log file.

Table 4-1 Logging Level and Destination Parameters

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

Specifying the Criteria for Logging Messages

The criteria for selecting SIP messages to log can be defined either in XML files that are packaged with the logging Servlet's application, or as initialization parameters in the Servlet's `sip.xml` deployment descriptor. The sections that follow describe each method.

Using XML Documents to Specify Logging Criteria

If you do not specify logging criteria as an initialization parameter to the logging Servlet, the Servlet looks for logging criteria in a pair of XML descriptor files in the top level of the logging application. These descriptor files, named `request-pattern.xml` and `response-pattern.xml`, define patterns that WebLogic SIP Server uses for selecting SIP requests and responses to place in the log file.

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

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

Listing 4-2 Sample response-pattern.xml for msgTraceLogger Servlet

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

Additional operators and conditions for matching SIP messages are described in "trace-pattern.dtd Reference" on page 4-10. Most conditions, such as the `equal` condition shown in Listing 4-2, require a variable (`var` element) that identifies the portion of the SIP message to evaluate. Table 4-2 lists some common variables and sample values. For additional variable names and examples, see Chapter 11: Mapping Requests to Servlets in the *SIP Servlet API 1.0* specification; WebLogic SIP Server enables mapping of both request and response variables to logging Servlets.

Table 4-2 Pattern-matching Variables and Sample Values

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

Both `request-pattern.xml` and `response-pattern.xml` use the same Document Type Definition (DTD). See "trace-pattern.dtd Reference" on page 4-10 for more information.

Using Servlet Parameters to Specify Logging Criteria

Pattern-matching criteria can also be specified as initialization parameters to the logging Servlet, rather than as separate XML documents. The parameter names used to specify matching criteria are `request-pattern-string` and `response-pattern-string`. They are defined along with the logging level and destination as described in "Configuring the Logging Level and Destination" on page 4-3.

The value of each pattern-matching parameter must consist of a valid XML document that adheres to the DTD for standalone pattern definition documents (see "Using XML Documents to Specify Logging Criteria" on page 4-5). Because the XML documents that define the patterns and values must not be parsed as part of the `sip.xml` descriptor, you must enclose the contents within the `CDATA` tag. Listing 4-3 shows the full `sip.xml` entry for the sample logging Servlet, `invTraceLogger`. The final two `init-param` elements specify that the Servlet log only `INVITE` request methods and `OPTIONS` response methods.

Listing 4-3 Logging Criteria Specified as init-param Elements

```
<servlet>
      <servlet-name>invTraceLogger</servlet-name>
      <servlet-class>com.bea.wcp.sip.engine.tracing.listener.TraceMessageL
istenerImpl</servlet-class>
      <init-param>
        <param-name>domain</param-name>
        <param-value>true</param-value>
      </init-param>
      <init-param>
        <param-name>level</param-name>
        <param-value>full</param-value>
      </init-param>
      <init-param>
        <param-name>request-pattern-string</param-name>
        <param-value>
            <![CDATA[
                <?xml version="1.0" encoding="UTF-8"?>
                <!DOCTYPE pattern
                PUBLIC "Registration//Organization//Type Label//Definition
Language"
                   "trace-pattern.dtd">
                <pattern>
                  <equal>
                    <var>request.method</var>
                    <value>INVITE</value>
                  </equal>
```

```
                    </pattern>
                ]]>
            </param-value>
        </init-param>
        <init-param>
            <param-name>response-pattern-string</param-name>
            <param-value>
                <![CDATA[
                    <?xml version="1.0" encoding="UTF-8"?>
                    <!DOCTYPE pattern
                    PUBLIC "Registration//Organization//Type Label//Definition
Language"
                       "trace-pattern.dtd">
                    <pattern>
                      <equal>
                        <var>response.method</var>
                        <value>OPTIONS</value>
                      </equal>
                    </pattern>
                ]]>
            </param-value>
        </init-param>
        <load-on-startup/>
    </servlet>
```

Specifying Content Types for Unencrypted Logging

By default WebLogic SIP Server uses String format (UTF-8 encoding) to log the content of SIP messages having a text or application/sdp Content-Type value. For all other Content-Type

values, WebLogic SIP Server attempts to log the message content using the character set specified in the `charset` parameter of the message, if one is specified. If no `charset` parameter is specified, or if the `charset` value is invalid or unsupported, WebLogic SIP Server uses Base-64 encoding to encrypt the message content before logging the message.

If you want to avoid encrypting the content of messages under these circumstances, specify a list of String-representable Content-Type values using the `string-rep` element in `sipserver.xml`. The `string-rep` element can contain one or more `content-type` elements to match. If a logged message matches one of the configured `content-type` elements, WebLogic SIP Server logs the content in String format using UTF-8 encoding, regardless of whether or not a `charset` parameter is included.

Note: You do not need to specify text/* or application/sdp content types as these are logged in String format by default.

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

Listing 4-4 Logging String Content for Additional Content Types

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

Managing Logging Performance

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

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

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

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

Enabling Log Rotation and Viewing Log Files

The WebLogic SIP Server logging infrastructure enables you to automatically write to a new log file when the existing log file reaches a specified size. You can also view log contents using the Administration Console or configure additional server-level events that are written to the log. See Rotate log file in the WebLogic Server 9.2 documentation for more information about basic log management.

trace-pattern.dtd Reference

`trace-pattern.dtd` defines the required contents of the `request-pattern.xml` and `response-pattern.xml`, documents, as well as the values for the `request-pattern-string` and `response-pattern-string` Servlet `init-param` variables.

Listing 4-5 trace-pattern.dtd

```
<!--
```

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

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

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

<!ELEMENT pattern (%condition;)>

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

<!ELEMENT and (%condition;)+>

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

<!ELEMENT or (%condition;)+>

<!--
```

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

    <!ELEMENT not (%condition;)>

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

    <!ELEMENT equal (var, value)>

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

    <!ELEMENT contains (var, value)>

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

    <!ELEMENT exists (var)>

    <!--
    -->

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

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

<!ELEMENT var (#PCDATA)>

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

<!ELEMENT value (#PCDATA)>

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

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

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

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

<!--
```

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

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

Adding Tracing Functionality to SIP Servlet Code

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

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

To implement tracing in a Servlet, you use the factory class to create a delegate in the Servlet's `init()` method as shown in Listing 4-6.

Listing 4-6 Using the TraceMessageListenerFactory

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

  private MessageListener delegate;


  public void init() throws ServletException {

    ServletContext sc = (ServletContext) getServletContext();

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

    delegate = factory.createTraceMessageListener(getServletConfig());

  }

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

    delegate.onRequest(req,incoming);

  }

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

    delegate.onResponse(resp,incoming);

  }

}
```

Order of Startup for Listeners and Logging Servlets

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

CHAPTER 5

Applying Patches

WebLogic SIP Server 3.1 does not use the `InstallPatch` utility that was included in WebLogic SIP Server 2.2. Instead, refer to the instructions that were provided with your patch.

CHAPTER 6

Upgrading Deployed SIP Applications

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

- "Overview of SIP Application Upgrades" on page 6-1
- "Requirements and Restrictions for Upgrading Deployed Applications" on page 6-2
- "Steps for Upgrading a Deployed SIP Application" on page 6-3
- "Assign a Version Identifier" on page 6-4
- "Deploy the Updated Application Version" on page 6-5
- "Undeploy the Older Application Version" on page 6-5
- "Roll Back the Upgrade Process" on page 6-7
- "Accessing the Application Name and Version Identifier" on page 6-7
- "Using Administration Mode" on page 6-7

Overview of SIP Application Upgrades

With WebLogic SIP Server, you can upgrade a deployed SIP application to a newer version without losing existing calls being processed by the application. This type of application upgrade is accomplished by deploying the newer application version alongside the older version. WebLogic SIP Server automatically manages the SIP Servlet mapping so that new requests are directed to the new version. Subsequent messages for older, established dialogs are directed to

the older application version until the calls complete. After all of the older dialogs have completed and the earlier version of the application is no longer processing calls, you can safely undeploy it.

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

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

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

WebLogic SIP Server application upgrades provide the same functionality as WebLogic Server 9.2 application upgrades, with the following exceptions:

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

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

See Redeploying Applications in a Production Environment in the WebLogic Server 9.2 documentation for general information and instructions regarding production application redeployment.

Requirements and Restrictions for Upgrading Deployed Applications

To use the application upgrade functionality of WebLogic SIP Server:

- You must assign version information to your updated application in order to distinguish it from the older application version. Note that only the newer version of a deployed application requires version information; if the currently-deployed application contains no version designation, WebLogic SIP Server automatically treats this application as the "older" version. See "Assign a Version Identifier" on page 6-4.
- A maximum of two different versions of the same application can be deployed at one time.
- If your application hard-codes the use of an application name (for example, in composed applications where multiple SIP Servlets process a given call), you must replace the application name with calls to a helper method that obtains the base application name. WebLogic Server provides `ApplicationRuntimeMBean` methods for obtaining the base application name and version identifier, as well as determining whether the current application version is active or retiring. See Accessing the Application Name and Version Identifier.
- When applications take part in a composed application (using application composition techniques), WebLogic SIP Server always uses the latest version of an application when only the base name is supplied.
- If you want to deploy an application in administration mode, you must configure a `sips-admin` channel that uses TLS transport. See Managing WebLogic SIP Server Network Resources in *Configuring Network Resources*.

Steps for Upgrading a Deployed SIP Application

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

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

Each procedure is described in the sections that follow. You can also Roll Back the Upgrade Process if you discover a problem with the newly-deployed application. Applications that are composed of multiple SIP Servlets may also need to use the `ApplicationRuntimeMBean` for Accessing the Application Name and Version Identifier.

Assign a Version Identifier

WebLogic SIP Server uses a version identifier—a string value—appended to the application name to distinguish between multiple versions of a given application. The version string can be a maximum of 215 characters long, and must consist of valid characters as identified in Table 0-1.

Table 0-1 Valid and Invalid Characters

Valid ASCII Characters	Invalid Version Constructs
a-z	..
A-Z	.
0-9	
period (“.”), underscore (“_”), or hyphen (“-”) in combination with other characters	

For deployable SIP Servlet WAR files, you must define the version identifier in the MANIFEST.MF file of the application or specify it on the command line at deployment time. See Specifying an Application Version Identifier in the WebLogic Server 9.2 documentation for more information.

Defining the Version in the Manifest

Both WAR and EAR deployments must specify a version identifier in the MANIFEST.MF file. Listing 0-1 shows an application with the version identifier “v2”:

Listing 0-1 Version Identifier in Manifest

```
Manifest-Version: 1.0

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

Weblogic-Application-Version: v2
```

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

Deploy the Updated Application Version

To begin the upgrade process, simply deploy the updated application archive using either the Administration Console or `weblogic.Deployer` utility. Use the `-retiretimeout` option to the `weblogic.Deployer` utility if you want to automatically undeploy the older application version after a fixed amount of time. For example:

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

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

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

WebLogic SIP Server also distinguishes between a deployment that has no version identifier (no version string in the manifest) and a subsequent version that does specify a version identifier. This enables you to easily upgrade applications that were packaged before you began including version information as described in "Assign a Version Identifier" on page 6-4.

Undeploy the Older Application Version

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

To determine whether a deployed application is processing messages, you can obtain the active session count from the application's `SipApplicationRuntimeMBean` instance. Listing 0-2 shows the sample WLST commands for viewing the active session count for the `findme` sample application on the default single-server domain.

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

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

Listing 0-2 Sample WLST Session for Examining Session Count

```
connect()

custom()

cd
('examples:Location=myserver,Name=myserver_myserver_findme_findme,ServerRuntim
e=myserver,Type=SipApplicationRuntime')

ls()

-rw-   ActiveAppSessionCount                        0

-rw-   ActiveSipSessionCount                        0

-rw-   AppSessionCount                              0

-rw-   CachingDisabled                              true

-rw-   MBeanInfo                                   weblogic.management.tools.In

fo@5ae636

-rw-   Name                                        myserver_myserver_findme_fin

dme

-rw-   ObjectName                                  examples:Location=myserver,N

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

ntime

-rw-   Parent                                      examples:Location=myserver,N

ame=myserver,Type=ServerRuntime

-rw-   Registered                                  false

-rw-   SipSessionCount                            0

-rw-   Type                                        SipApplicationRuntime
```

```
-rwx   preDeregister                                  void :
```

Roll Back the Upgrade Process

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

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

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

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

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

Accessing the Application Name and Version Identifier

If you intend to use WebLogic SIP Server's production upgrade feature, applications that are composed of multiple SIP Servlets should not hard-code the application name. Instead of hard-coding the application name, your application can dynamically access the deployment name or version identifier by using helper methods in `ApplicationRuntimeMBean`. See ApplicationRuntimeMBean in the WebLogic Server 9.2 documentation for more information.

Using Administration Mode

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

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

Note: If using TLS is not feasible with your application, you can alternately change the Servlet role mapping rules to allow only 1 user on the newer version of the application. This

enables you to deploy the newer version alongside the older version, while restricting access to the newer version.

CHAPTER 7

Avoiding and Recovering From Server Failures

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

WebLogic SIP Server uses a highly clustered architecture as the basis for minimizing the impact of failure events. However, even in a clustered environment it is important to prepare for a sound recovery process in the event that an individual server or server machine fails.

The following sections summarize WebLogic SIP Server failure prevention and recovery features, and describe the configuration artifacts that are required in order to restore different portions of a WebLogic SIP Server domain:

- "Failure Prevention and Automatic Recovery Features" on page 7-1
- "Directory and File Backups for Failure Recovery" on page 7-4
- "Restarting a Failed Administration Server" on page 7-8
- "Restarting Failed Managed Servers" on page 7-10

Failure Prevention and Automatic Recovery Features

WebLogic SIP Server, and the underlying WebLogic Server platform, provide many features that protect against server failures. In a production system, all available features should be used in order to ensure uninterrupted service.

Overload Protection

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

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

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

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

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

The underlying WebLogic Server platform also detects increases in system load that can affect deployed application performance and stability. WebLogic Server allows administrators to configure failure prevention actions that occur automatically at predefined load thresholds. Automatic overload protection helps you avoid failures that result from unanticipated levels of application traffic or resource utilization as indicated by:

- A workload manager's capacity being exceeded
- The HTTP session count increasing to a predefined threshold value
- Impending out of memory conditions

See Avoiding and Managing Overload in the WebLogic Server 9.2 documentation for more information.

Redundancy and Failover for Clustered Services

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

See Overview of the WebLogic SIP Server Architecture in the *Configuration Manual* for more information.

Automatic Restart for Failed Server Instances

WebLogic Server self-health monitoring features improve the reliability and availability of server instances in a domain. Selected subsystems within each server instance monitor their health status based on criteria specific to the subsystem. (For example, the JMS subsystem monitors the condition of the JMS thread pool while the core server subsystem monitors default and user-defined execute queue statistics.) If an individual subsystem determines that it can no longer operate in a consistent and reliable manner, it registers its health state as "failed" with the host server.

Each WebLogic Server instance, in turn, checks the health state of its registered subsystems to determine its overall viability. If one or more of its critical subsystems have reached the FAILED state, the server instance marks its own health state FAILED to indicate that it cannot reliably host an application.

When used in combination with Node Manager, server self-health monitoring enables you to automatically reboot servers that have failed. This improves the overall reliability of a domain, and requires no direct intervention from an administrator. For more information, see Using Node Manager to Control Servers in the WebLogic Server 9.2 documentation.

Managed Server Independence Mode

Managed Servers maintain a local copy of the domain configuration. When a Managed Server starts, it contacts its Administration Server to retrieve any changes to the domain configuration that were made since the Managed Server was last shut down. If a Managed Server cannot connect to the Administration Server during startup, it can use its locally-cached configuration information—this is the configuration that was current at the time of the Managed Server's most recent shutdown. A Managed Server that starts up without contacting its Administration Server to check for configuration updates is running in *Managed Server Independence (MSI)* mode. By default, MSI mode is enabled. See Replicating domain config files for Managed Server Independence in the WebLogic Server 9.2 documentation.

Automatic Migration of Failed Managed Servers

When using Linux or UNIX operating systems, you can use WebLogic Server's server migration feature to automatically start a candidate (backup) server if a Network tier server's machine fails or becomes partitioned from the network. The server migration feature uses node manager, in conjunction with the `wlsifconfig.sh` script, to automatically boot candidate servers using a floating IP address. Candidate servers are booted only if the primary server hosting a Network

tier instance becomes unreachable. See Migration in the WebLogic Server 9.2 documentation for more information about using the server migration feature.

Geographic Redundancy for Regional Site Failures

In addition to server-level redundancy and failover capabilities, WebLogic SIP Server enables you to configure peer sites to protect against catastrophic failures, such as power outages, that can affect an entire domain. This enables you to failover from one geographical site to another, avoiding complete service outages. See Configuring Geographically-Redundant Installations in the *Configuration Guide* for more information.

Directory and File Backups for Failure Recovery

Recovery from the failure of a server instance requires access to the domain's configuration data. By default, the Administration Server stores a domain's primary configuration data in a file called *domain_name*`/config/config.xml`, where *domain_name* is the root directory of the domain. The primary configuration file may reference additional configuration files for specific WebLogic Server services, such as JDBC and JMS, and for WebLogic SIP Server services, such as SIP container properties and data tier configuration. The configuration for specific services are stored in additional XML files in subdirectories of the *domain_name*`/config` directory, such as *domain_name*`/config/jms`, *domain_name*`/config/jdbc`, and *domain_name*`/config/custom` for WebLogic SIP Server configuration files.

The Administration Server can automatically archive multiple versions of the domain configuration (the entire *domain-name*`/config` directory). The configuration archives can be used for system restoration in cases where accidental configuration changes need to be reversed. For example, if an administrator accidentally removes a configured resource, the prior configuration can be restored by using the last automated backup.

The Administration Server stores only a finite number of automated backups locally in *domain_name*`/config`. For this reason, automated domain backups are limited in their ability to guard against data corruption, such as a failed hard disk. Automated backups also do not preserve certain configuration data that are required for full domain restoration, such as LDAP repository data and server start-up scripts. BEA recommends that you also maintain multiple backup copies of the configuration and security offline, in a source control system.

This section describes file backups that WebLogic SIP Server performs automatically, as well as manual backup procedures that an administrator should perform periodically.

Enabling Automatic Configuration Backups

Follow these steps to enable automatic domain configuration backups on the Administration Server for your domain:

1. Access the Administration Console for your domain.
2. In the left pane of the Administration Console, select the name of the domain.
3. In the right pane, click the Configuration->General tab.
4. In the Advanced Options bar, click Show.
5. In the Archive Configuration Count box, enter the maximum number of configuration file revisions to save.
6. Click Apply.

When you enable configuration archiving, the Administration Server automatically creates a configuration JAR file archive each time the Administrator uses the Activate Changes button in the Administration Console to change the active configuration. The JAR file contains a complete copy of the previous configuration (the complete contents of the *domain-name*\config directory). JAR file archive files are stored in the *domain-name*\configArchive directory. The files use the naming convention config-*number*.jar, where *number* is the sequential number of the archive.

When you save a change to a domain's configuration, the Administration Server saves the previous configuration in *domain-name*\configArchive\config.xml#*n*. Each time the Administration Server saves a file in the configArchive directory, it increments the value of the #*n* suffix, up to a configurable number of copies—5 by default. Thereafter, each time you change the domain configuration:

- The archived files are rotated so that the newest file has a suffix with the highest number,
- The previous archived files are renamed with a lower number, and
- The oldest file is deleted.

Keep in mind that configuration archives are stored locally within the domain directory, and they may be overwritten according to the maximum number of revisions you selected. For these reasons, you must also create your own off-line archives of the domain configuration, as described in "Storing the Domain Configuration Offline" on page 7-6.

Storing the Domain Configuration Offline

Although automatic backups protect against accidental configuration changes, they do not protect against data loss caused by a failure of the hard disk that stores the domain configuration, or accidental deletion of the domain directory. To protect against these failures, you must also store a complete copy of the domain configuration offline, preferably in a source control system.

BEA recommends storing a copy of the domain configuration at regular intervals. For example, back up a new revision of the configuration when:

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

The domain configuration backup should contain the complete contents of the *domain_name*`/config` directory. For example:

```
cd ~/bea/user_projects/domains/mydomain
```

```
tar cvf domain-backup-06-17-2007.jar config
```

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

Backing Up Server Start Scripts

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

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

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

Backing Up Logging Servlet Applications

If you use WebLogic SIP Server logging Servlets (see "Logging SIP Requests and Responses" on page 4-1) to perform regular logging or auditing of SIP messages, backup the complete application source files so that you can easily redeploy the applications should the staging server fail or the original deployment directory becomes corrupted.

Backing Up Security Data

The WebLogic Security service stores its configuration data `config.xml` file, and also in an LDAP repository and other files.

Backing Up the WebLogic LDAP Repository

The default Authentication, Authorization, Role Mapper, and Credential Mapper providers that are installed with WebLogic SIP Server store their data in an LDAP server. Each WebLogic SIP Server contains an embedded LDAP server. The Administration Server contains the master LDAP server, which is replicated on all Managed Servers. If any of your security realms use these installed providers, you should maintain an up-to-date backup of the following directory tree:

*domain_name**adminServer*\ldap

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

Each WebLogic SIP Server has an LDAP directory, but you only need to back up the LDAP data on the Administration Server—the master LDAP server replicates the LDAP data from each Managed Server when updates to security data are made. WebLogic security providers cannot modify security data while the domain's Administration Server is unavailable. The LDAP repositories on Managed Servers are replicas and cannot be modified.

The `ldap/ldapfiles` subdirectory contains the data files for the LDAP server. The files in this directory contain user, group, group membership, policies, and role information. Other subdirectories under the `ldap` directory contain LDAP server message logs and data about replicated LDAP servers.

Do not update the configuration of a security provider while a backup of LDAP data is in progress. If a change is made—for instance, if an administrator adds a user—while you are backing up the `ldap` directory tree, the backups in the `ldapfiles` subdirectory could become

inconsistent. If this does occur, consistent, but potentially out-of-date, LDAP backups are available.

Once a day, a server suspends write operations and creates its own backup of the LDAP data. It archives this backup in a `ZIP` file below the `ldap\backup` directory and then resumes write operations. This backup is guaranteed to be consistent, but it might not contain the latest security data.

For information about configuring the LDAP backup, see Configuring Backups for the Embedded LDAP Server in the WebLogic Server 9.2 Documentation.

Backing Up SerializedSystemIni.dat and Security Certificates

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

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

Backing Up Additional Operating System Configuration Files

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

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

Restarting a Failed Administration Server

When you restart a failed Administration Server, no special steps are required. Start the Administration Server as you normally would.

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

you can restart the Administration Server on the same machine it was running on when the domain was started.

Restarting an Administration Server on the Same Machine

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

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

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

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

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

Restarting an Administration Server on Another Machine

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

1. Install the WebLogic SIP Server software on the new administration machine (if this has not already been done).

2. Make your application files available to the new Administration Server by copying them from backups or by using a shared disk. Your application files should be available in the same relative location on the new file system as on the file system of the original Administration Server.
3. Make your configuration and security data available to the new administration machine by copying them from backups or by using a shared disk. For more information, refer to "Storing the Domain Configuration Offline" on page 7-6 and "Backing Up Security Data" on page 7-7.
4. Restart the Administration Server on the new machine.

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

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

Restarting Failed Managed Servers

If the machine on which the failed Managed Server runs can contact the Administration Server for the domain, simply restart the Managed Server manually or automatically using Node Manager. Note that you must configure Node Manager and the Managed Server to support automated restarts, as described in Using Node Manager to Control Servers in the WebLogic Server 9.2 documentation.

If the Managed Server cannot connect to the Administration Server during startup, it can retrieve its configuration by reading locally-cached configuration data. A Managed Server that starts in this way is running in Managed Server Independence (MSI) mode. For a description of MSI mode, and the files that a Managed Server must access to start up in MSI mode, see Replicating domain config files for Managed Server Independence in the WebLogic Server 9.2 documentation.

To start up a Managed Server in MSI mode:

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

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

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

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

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

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

   The Managed Server will run in MSI mode until it is contacted by its Administration Server. For information about restarting the Administration Server in this scenario, see "Restarting a Failed Administration Server" on page 7-8.

CHAPTER 8

Upgrading WebLogic SIP Server Software

Notes: The sections that follow provide only general instructions for upgrading WebLogic SIP Server software to a new Service Pack release. Your service pack or new software may contain additional instructions and tools to help you upgrade the software. Defer to those instructions if they are available.

If you are upgrading from one major software release to another (such as upgrading from WebLogic SIP Server 2.2 to WebLogic SIP Server 3.1), refer to Upgrading a WebLogic SIP Server 2.2 Configuration to Version 3.1 in the *Configuration Guide* instead of these instructions.

If you want to upgrade a SIP protocol application or converged HTTP/SIP protocol application, these instructions are not required. See "Upgrading Deployed SIP Applications" on page 6-1 instead.

The following sections describe how to upgrade production WebLogic SIP Server installations to a new Service Pack release of the software:

- "Overview of System Upgrades" on page 8-1
- "Requirements for Upgrading a Production System" on page 8-2
- "Upgrading to a New Version of WebLogic SIP Server" on page 8-3

Overview of System Upgrades

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

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

The sections that follow describe how to use a configured load balancer to perform a “live” upgrade of the WebLogic SIP Server software on a production installation.

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

Requirements for Upgrading a Production System

To upgrade a production WebLogic SIP Server installation you require:

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

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

Upgrading to a New Version of WebLogic SIP Server

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

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

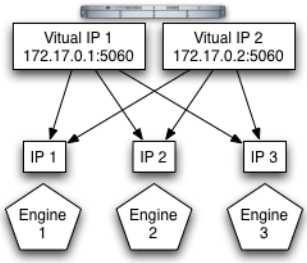
1. Configure the Load Balancer—Define a new, internal Virtual IP address for the new engine tier cluster you will configure.
2. Configure the New Engine Tier Cluster—Create and configure a new, empty engine tier cluster that will host upgraded engine tier servers and your converged applications.
3. Define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

Each procedure is described in the sections that follow.

Configure the Load Balancer

Begin the software upgrade procedure by defining a new internal virtual IP address for the new engine tier cluster you will create in "Configure the New Engine Tier Cluster" on page 8-4. The individual server IP addresses (the pool definition) for the new virtual IP address should be identical to the pool definition of your currently-active engine tier cluster. Figure 8-1 shows a sample configuration for a cluster having three engine tier server instances; both virtual IP addresses define the same servers.

Figure 8-1 Virtual IP Address Configuration for Parallel Clusters



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

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

Configure the New Engine Tier Cluster

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

1. On the Administration Server machine, install the new WebLogic SIP Server software into a new BEA home directory. The steps that follow refer to `c:\beanew` as the BEA home directory in which the new software was installed. `c:\bea` refers to the software implementation that is being upgraded.
2. Log in to the Administration Console for the active WebLogic SIP Server domain.
3. In the Administration Console, create a new, empty engine tier cluster for hosting the upgraded engine tier servers:
   a. In the left pane, select the Clusters node.
   b. Click New to configure a new cluster.
   c. Enter a name for the new cluster. For example, "NewEngineCluster."

d. Enter the multicast address and port number of the cluster.

e. Click OK to create the cluster.

4. Proceed to "Define the Cluster-to-Load Balancer Mapping" on page 8-5.

Define the Cluster-to-Load Balancer Mapping

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

To define the cluster-to-load balancer mapping:

1. Log in to the Administration Console for the active WebLogic SIP Server domain.
2. Click the Lock & Edit button.
3. In the Administration Console, create a new load balancer map entry for the new cluster you created:

   a. In the left pane, select the SipServer node.

   b. Select the Configuration->Loadbalancer Map tab.

   c. Click New to create a new mapping.

   d. Enter the Cluster Name and SIP URI of the new cluster you created in "Configure the New Engine Tier Cluster" on page 8-4. For example, enter "NewEngineCluster" and "sip:172.17.0.2:5060".

   e. Click Finish to create the new mapping.

4. Repeat Step 2 to enter a mapping for the older cluster. For example, create an entry for "EngineCluster" and "sip:172.17.0.1:5060".
5. Click Activate Changes to apply the new configuration.

   Note that you must include a mapping for both the older and the newer engine tier cluster. A mapping consists of the internal virtual IP address of the cluster configured on the load balancer, as well as the cluster name defined in the WebLogic SIP Server domain. Listing 0-3 shows an entry in the `sipserver.xml` file for the sample clusters described earlier.

Listing 0-3 Sample cluster-loadbalancer-map Definition

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

Duplicate the SIP Servlet Configuration

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

```
cp c:\bea\user_projects\domains\mydomain\config\custom\*.xml
c:\beanew\user_projects\domains\mydomain\config\custom
```

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

Upgrade Engine Tier Servers and Target Applications to the New Cluster

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

1. Access the Administration Console for your production domain.
2. Click Lock & Edit.
3. Select the first running engine tier server that you want to upgrade:
   a. Select the Environment->Servers tab in the left pane.

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

4. Select the Control->Start/Stop tab in the right pane.
5. Select Shutdown->When work completes from the table.
6. Select Yes to perform the shutdown.

   The server remains active while clients are still accessing the server, but no new connection requests are accepted. After all existing client connections have ended or timed out, the server shuts down. Other server instances in the engine tier process client requests during the shutdown procedure.

7. Select the Environment->Servers tab in the left pane and verify that the Managed Server has shut down.
8. Change the stopped server's cluster membership so that it is a member of the new engine tier cluster:

   a. Expand the Environment->Clusters tab in the left pane.

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

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

   d. Select the checkbox next to the name of the stopped server, and click Remove.

   e. Click Yes to remove the server from the cluster.

   f. Click Apply.

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

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

   i. Click Add to add a new server.

   j. Select the name of the stopped server from the drop-down list, and click Finish.

9. After adding the first engine tier server to the new cluster, you can now target your own applications to the new cluster.

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

   a. In the left pane, select the Deployments node.

   b. Click Install.

c. Use the links to select an application to deploy, and click Next.

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

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

f. Click Finish.

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

h. Click Start->Servicing all requests.

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

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

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

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

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

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

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

At this point, all running Managed Servers are using the new SIP Container implementation and are hosting your production SIP Servlets with the same SIP Servlet container settings as your old configuration. Data tier servers can now be upgraded using the instructions in "Upgrade Data Tier Servers" on page 8-8.

Upgrade Data Tier Servers

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

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

WARNING: Do not shut down or restart a data tier server instance in a partition unless two additional servers in the same partition are available, and both are in the `ONLINE` state. See Monitoring and Troubleshooting Data Tier Servers in the *Configuration Guide* for information about determining the state of data tier servers.