Oracle® Communications Converged Application Server

Concepts

Release 8.1

F82336-01

October 2023

Oracle Communications Converged Application Server Concepts, Release 8.1

F82336-01

Contents

About This Guide

This document provides a technical overview of Oracle Communications Converged Application Server, including its features, architecture, standards alignment, and supported platforms. It also describes the service invocation method of the SIP Servlet API.

Table 1   Revision History

Date	Revision
Oct 2023	• Initial release

1

Overview of Converged Application Server Architecture

This chapter introduces the Oracle Communications Converged Application Server.

Before continuing, you should familiarize yourself with the Internet Engineering Task Force (IETF) standards listed in Table 1-1.

Table 1-1 SIP and SDP Standards

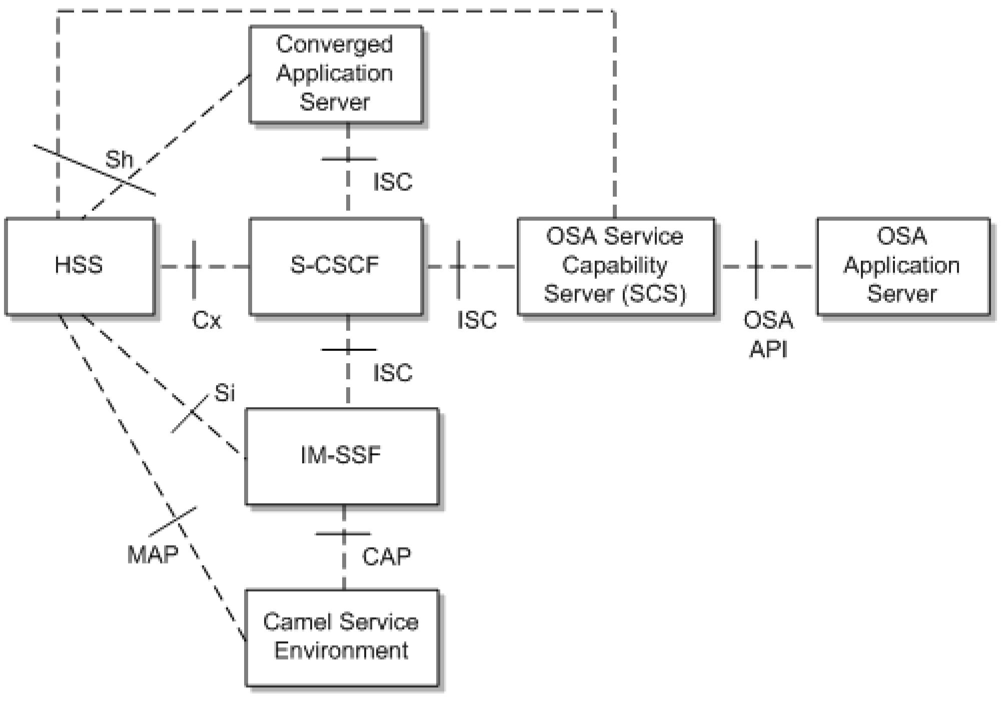
Protocol	Description	URL
Session Initiation Protocol (SIP)	From the IETF document, "[SIP is} an application-layer control (signaling) protocol for creating, modifying, and terminating sessions with one or more participants. These sessions include Internet telephone calls, multimedia distribution, and multimedia conferences."	`https://www.ietf.org/rfc/rfc3261.txt`
Session Description Protocol (SDP)	From the IETF document, "SDP is intended for describing multimedia sessions for the purposes of session announcement, session invitation, and other forms of multimedia session initiation."	`https://www.ietf.org/rfc/rfc4566.txt`

About the Converged Application Server

Converged Application Server is a carrier-class Java Platform, Enterprise Edition (Java EE) application server that has been extended with support for the SIP and a number of operational enhancements that allow it to meet the demanding requirements of next-generation Internet Protocol-based communications networks. The Converged Application Server implementation is based on Oracle's widely deployed and time-tested Java EE-compliant WebLogic Server product.

In a typical IP Multimedia Subsystem (IMS) deployment, Converged Application Server fills the role of the IMS SIP Application Server. Figure 1-1 shows the Converged Application Server in the context of the functional architecture for the provision of service in the IMS, as specified by 3GPP TS 23.002, "Network Architecture."

Figure 1-1   Converged Application Server in the IMS Service Architecture


Converged Application Server
Sh
ISC
HSS
S-CSCF
OSA Service Capability Server (SCS)
OSA Application Server
Cx
ISC
OSA API
ISC
Si
IM-SSF
MAP
CAP
Camel Service Environment

Converged Application Server supports all of the standard Oracle WebLogic Server programming interfaces and facilities, such as Java Transaction API (JTA), Java Activation Framework (JAF), Java Message Service (JMS), Java Naming and Directory Interface (JNDI), Java Database Connectivity (JDBC), and Enterprise JavaBeans (EJB). Converged Application Server also supports the protocols typically associated with a standards-compliant Java EE application server, including Remote Method Invocation (RMI) over Internet Inter-Orb Protocol (IIOP), HTTP 1.1, Lightweight Directory Access Protocol (LDAP), Simple Mail Transfer Protocol (SMTP), Post Office Protocol (POP), Internet Message Access Protocol (IMAP), and SNMPv2.

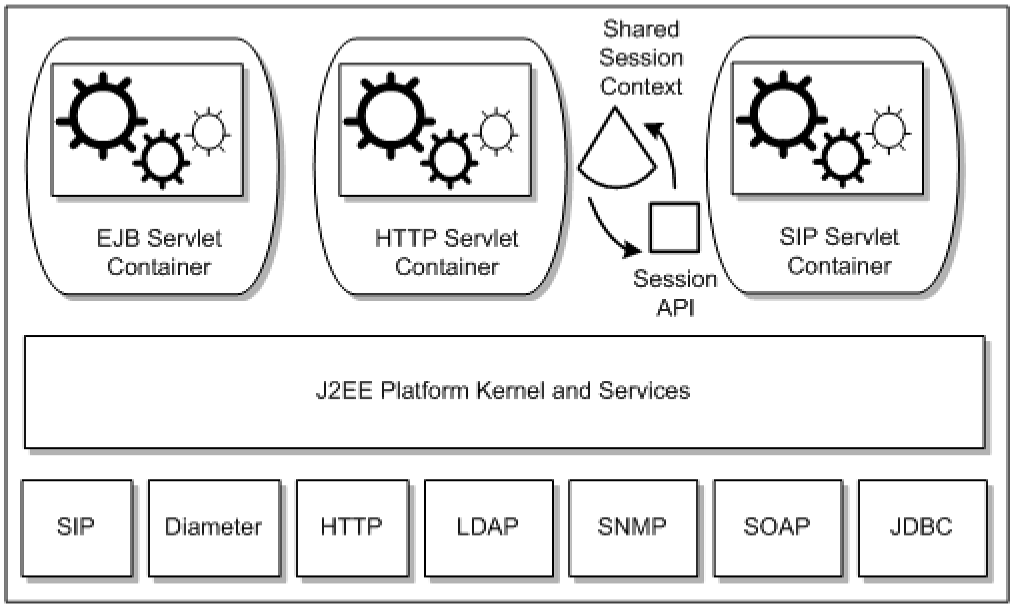
Converged Application Server then builds upon the base Java EE programming model by integrating a SIP Servlet Container that is compliant with the JSR-359 SIP Servlet API specification. This “converged" container provides an execution environment for applications containing both HTTP and SIP protocol handling components, as well as other protocols such as Diameter.

Converged Application Server Architecture

The “SIP Stack" of Converged Application Server is fully integrated into the SIP Servlet container and is substantially more powerful and easier to use than a traditional protocol stack.

As shown in Figure 1-2, Converged Application Server combines the SIP Servlet container with EJB and HTTP Servlet containers, supporting application convergence through session context sharing.

Figure 1-2 Converged Application Server Extended Java EE for Next Generation Networks



The SIP Servlet API defines a higher layer of abstraction than simple protocol stacks provide and frees the developer from concern for the mechanics of the SIP protocol itself. Specifically, the API handles the syntactic validation of received requests, transaction layer timers, generation of non-application-related responses, generation of fully-formed SIP requests from request objects (which involves correct preparation of system headers and generation of syntactically correct SIP messages), and lower-layer transport protocols (such as TCP, UDP or SCTP).

The Servlet container distributes request and response objects to components in a structured way, maintains awareness of the state of the larger, converged SIP and HTTP application session, and manages the end-to-end object lifecycle, including resource, transaction, and session state management. The converged SIP and HTTP container thereby frees the developer from much work (and opportunity for error) and allows deployed applications to inherit the high-availability, performance, and operational features provided by the robust Converged Application Server container implementation.

The SIP Servlet API greatly simplifies the task of implementing SIP User Agents, Proxies and Back-to-Back-User-Agents, and it narrows the developers exposure to operational concerns such as resource management, reliability, manageability and interaction between services. (See "Developing Applications for Converged Application Server" for more information.)

Converged Application Server incorporates a number of architectural features that allow for its deployment as a highly-available, fault tolerant cluster.

Engines processes all signaling traffic and replicates transaction and session state between all engines in a cluster. This clustering capability, combined with a third-party load balancer, transparently provides services with Telco-grade availability, scalability, and fault tolerance (session retention), ensuring that ongoing sessions are not affected by the failure of individual cluster members since a production deployment of Converged Application Server has no single point of failure.

Configuring and Administering the Converged Application Server Deployment

Converged Application Server provides several tools and mechanisms for administration and configuration which include:

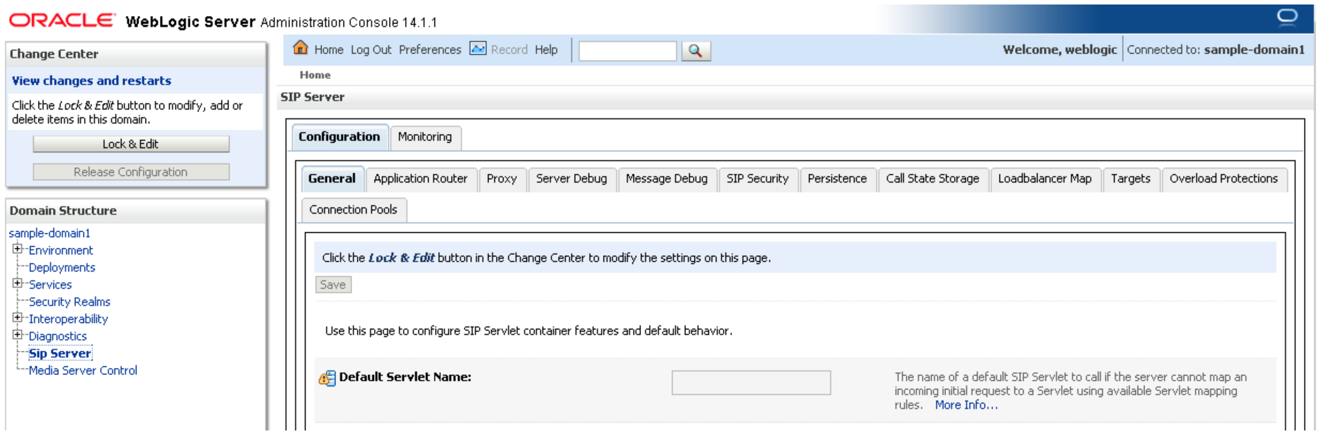
- **Administration Console**: Converged Application Server provides an extensive Web-based GUI that supports all configuration management, including deployment of applications, configuration of connectivity, and other common tasks. This interface offers secure, role-based administration of servers from any terminal that has access to the Administration Server and supports a standard HTML Web browser.
- **Java Management Extensions (JMX)**: Converged Application Server interoperates with standard network element management systems via JMX. Many common network management suites support JMX natively, which is the standard management technology for Java applications.
- **Simple Network Management Protocol (SNMP)**: Converged Application Server interoperates with standard network element management systems via use of SNMP, V2. The Converged Application Server SNMP MIB complies with MIB II. Converged Application Server also enables developers to send SNMP traps from within application code. See the *Oracle Communications Converged Application Server Developer Guide*.
  Converged Application Server also uses the SNMP features available in Oracle WebLogic Server, such as SNMP proxying. See Monitoring Oracle WebLogic Server with SNMP.
- **WebLogic Scripting Tool (WLST)**: Converged Application Server provides a Command Line Interface (CLI) using WLST for manual runtime configuration from any network terminal with secure access to the Administration Server. See Understanding the WebLogic Scripting Tool.

Administration Console

The Converged Application Server Web Administration Console is used for the following tasks:

- Configuring application container and related resource properties
- Configuring security
- Deploying applications or components
- Monitoring resource usage
- Configure debug logging for the servers in the domain
- Displaying log messages
- Starting and stopping servers

Figure 1-3   Converged Application Server Administration Console



Click the *Lock & Edit* button to modify, add or delete items in this domain.

Click the ***Lock & Edit*** button in the Change Center to modify the settings on this page.

Use this page to configure SIP Servlet container features and default behavior.

The name of a default SIP Servlet to call if the server cannot map an incoming initial request to a Servlet using available Servlet mapping rules.

In addition to providing access to the configuration for the implementation, the Converged Application Server Administration console serves as a monitoring and troubleshooting interface. It provides a convenient interface for observing system-wide usage metrics, including SIP traffic activity. See Monitoring and Overload Protection in *Converged Application Server Administrator Guide*.

2

Developing Applications for Converged Application Server

This chapter describes the environment for developing applications with Oracle Communications Converged Application Server.

Overview of Developing Applications for Converged Application Server

Oracle Communications Converged Application Server is a development and runtime platform for implementing and deploying communication services. The Converged Application Server supports capabilities that form the basis for advanced communication services, including those represented by Rich Communication Services (RCS) and Voice over Long-Term Evolution (VoLTE). The Converged Application Server simplifies the development of converged applications that provide voice, IM, rich media, and presence services to end users.

The Converged Application Server provides comprehensive support for the SIP protocol. The Session Initiation Protocol (SIP) protocol, specified by Java Specification Request (JSR) 359: SIP Servlet Specification, version 2.0, extends the basic concept of the Servlet. The SIP Servlet API specification describes not only the programming API but also the Servlet container function. The container is the Server (software) that hosts or “contains" applications written using the API. The SIP Servlet container hosts SIP applications.

The Converged Application Server SIP container performs a number of SIP functions as specified by various Request for Comments (RFCs), thus taking the burden off of the applications themselves. At the same time, the container exposes the application to SIP protocol messages through the SIP Servlet API. In this way, the application can perform various actions based on the SIP messages it receives from the container. Different applications can be coded and deployed to the container in order to provide various telecommunication or multimedia services.

SIP Protocol Support

The SIP Servlet API enables applications to perform a complete set of SIP Signaling functions. The SIP Protocol specification defines different types of high level SIP roles, namely User Agents (UAs) which include UA Clients, UA Servers, and Back-to-back user agents (B2BUAs). The SIP protocol also defines the roles of Proxies, Registrars, and Redirect Servers. The SIP Servlet API is a allows any of these roles to be coded as SIP Servlet application.

SIP is an extensible protocol, which is one of its strengths. Applications can extend the base protocol to add new features as necessary. In fact, there are a number of RFCs that define extensions to the base Internet Engineering Task Force (IETF) RFC 3261 SIP: Session Initiation Protocol. The SIP Servlet API is also designed to allow developers to easily extend functionality. This is accomplished by dividing up the SIP processing between the container functions the applications. Most of the base protocol processing is performed by the container, leaving some of the higher level tasks for the applications to perform. This clever division is what lends a great deal of power and flexibility to the SIP Servlet API.

Simplicity and Ease of Use

The SIP Servlet container handles "non-application-specific" complexity outside of the application code itself. Concerns like network connectivity, protocol transactions, dialog management and route processing are required by virtually all applications, and it would be enormously wasteful and error-prone to require each application to implement this support. With the SIP Servlet API, all of these tasks are managed by the container, leaving applications to provide higher level functions.

As an example, consider a SIP Proxy component:

1. A SIP Servlet within the SIP Servlet container receives a SIP request object and proxies it. A SIP Proxy must add its own *Via* header to the request; the header is required by the base SIP protocol to indicate which entities were traversed by the request. The *Via* header also stores the branch identifier which acts as the transaction identifier.

   Because the maintenance of transactions and their associated state machine is maintained by the container, it is the container that actually inserts the *Via* headers into the SIP Request.

2. The downstream SIP entity which next receives the request sends the response back along the path built up by the SIP entities in the path of the request that have inserted themselves into the *Via* or *Record-Route* headers.

3. The container gets the response, removes the *Via* header it inserted in the original request and then processes the response. The application code does not need to manage the *Via* header, which simplifies application development.

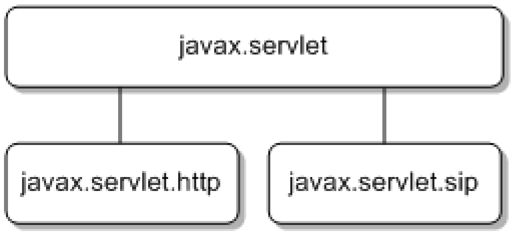
There are many cases in which Converged Application Server handles that sort of mundane, but essential, protocol detail.

Converged Applications

The SIP Servlet API specification is closely aligned with the Java Platform, Enterprise Edition (Java EE) specifications, and it is expected that containers that host SIP Servlet applications also make Java EE features available to developers. The most notable of these features is the HTTP Servlet container. There are many use cases in which a converged application, using both SIP and HTTP functions, is required, from conferencing and click-to-call applications to Presence and User Agent Configuration Management applications. Converged applications can also combine other protocols such as Diameter to perform advanced functions such as modifying subscriber profile data.

Figure 2-1 illustrates that `javax.servlet.http` and `javax.servlet.sip` converge in the SIP Servlet API.

Figure 2-1 HTTP/SIP Convergence in the SIP Servlet API



Application Composition

The SIP Servlet API enables multiple applications to execute on the same request or response, independently of one another. This is another very powerful feature of the SIP Servlet API. The promise is that application developers are able to write applications providing features that are independent of each other, but can be deployed to the same host SIP Servlet container. The applications can be “composed" (or sequenced) to provide a service on a call. This composition is facilitated by the container. See "SIP Servlet API Service Invocation" for more information.

Highly Reliable Implementation

Application data stored in container-managed session objects can benefit from replication and failover. Almost all applications that perform some useful functions require some state between different Requests and Responses. Some state information is mandated by the SIP protocol itself, such as the transaction state machine with its Server and Client Transactions, and the Dialog state machine.

The container also has a notion of message context which encapsulates the SIP level state, and the concept of Sessions, which are the SIP Servlet API constructs. Applications can save their own state in the Session objects maintained by the container. A carrier-grade container will replicate this state such that the call becomes fault tolerant of a container instance, as is done in Converged Application Server.

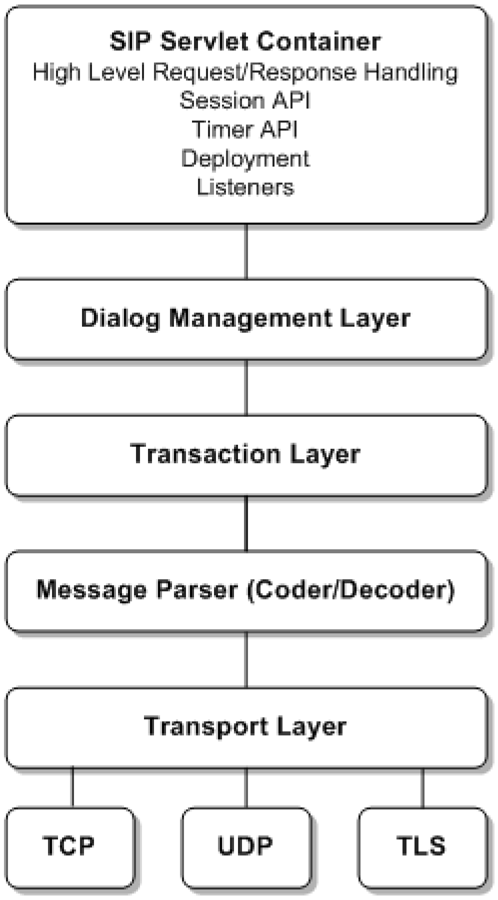
Overview of the SIP Servlet Container

Figure 2-2 shows the logical layers of a Converged Application Server SIP Servlet Container. The five layers shown from the bottom are what are known as the SIP stack, the functionality of which is defined in RFC 3261 and the associated RFCs that extend the base protocol.

SIP, being a transaction-based protocol, has a well-defined transaction layer. SIP requests are always followed by one or more provisional Responses and one final response, with the exception of the ACK which has no response. The transaction machinery is designed to keep track of the provisional and final responses.

Figure 2-2 shows the message processing layers in the Converged Application Server SIP Servlet container which are the following from top to bottom: Dialog Management Layer, Transaction Layer, Message Parser, the Transport Layer, and the bottom layer comprising of Transmission Control Protocol (TCP), User Datagram Protocol (UDP), and Transport Layer Security (TLS).

Figure 2-2 Message Processing Layers in the Converged Application Server SIP Servlet Container



SIP Dialog Handling

A dialog is a point-to-point session between two SIP endpoints that is uniquely identified by a dialog identifier. Not all SIP requests create dialogs. However, the ones that do create dialogs have a well-defined mechanism of establishing and tearing down the dialog (INVITE, SUBSCRIBE/NOTIFY, REFER).

The SIP stack shown in this diagram is not strictly in accordance with RFC 3261. It differs from the specification in that there is a layer called Transaction User (TU) above the Transaction layer, and the dialog management layer is not explicitly a layer in 3261. The “Dialog layer" is a very visible constituent of a SIP Servlet container because the dialogs correspond roughly to the `SipSession` objects. In Figure 2-2, the TU layer is actually split between the Dialog management layer and the big Container block.

The primary purpose of the Container is to host SIP Servlet applications that are written to the container's SIP Servlet API implementation. It exposes objects like `SipServletRequest`, `SipServletResponse`, different types of Sessions, facilities such as Timer, Logging, and so forth.

Although SIP is a human-readable, text-based protocol, and is well-defined in Request for Comment (RFC) 3261, writing SIP applications can be a challenging task. The SIP

Servlet API is designed to simplify SIP application development. While the SIP Servlet API allows access to all the headers present in a SIP Request, it does not require applications to understand or modify all of them for correct protocol behavior. Also, there are some headers that are strictly off limits for applications. The SIP Servlet API defines the so-called "system headers" which are to be managed only by the container. These headers include the *From*, *To*, *Call-ID*, *CSeq*, *Via*, *Route* (except through pushRoute), *Record-Route*, and *Contact* headers. Applications can add attributes to the *Record-Route* header and *Contact* header fields in all request messages, as well as 3xx and 485 responses. Additionally, for containers such as Converged Application Server that implement the reliable provisional responses extension, *RAck* and *RSeq* are also considered to be system headers. The system header management performed by the container offloads a tremendous amount of complexity from applications.

The *From*, *To*, *Call-ID*, and *CSeq* message headers collectively identify a given SIP dialog. The SIP Servlet container keeps track of the dialog state and dialog-related data for the hosted applications. The SIP Servlet container is responsible for managing *Record-Route*, *Contact*, and *Via* headers because the network listen points, failure management, multi-homing, transport switching, and so forth are also handled by the container. Applications can participate in the routing decisions of a Request emanating from the container by explicitly modifying *Request-URI* or adding *Route* headers with **pushRoute**(). As a result, applications have no responsibility for resource management. The SIP Servlet API draws heavily from Java EE standardization and common practices, such as the declarative usage of container features such as security, mapping, and environment resources.

Perhaps the greatest advantage of the SIP Servlet API is the API itself. The SIP Servlet API abstracts a large number of complex SIP tasks behind intuitive constructs. The Proxy interface, representing the proxy functionality in SIP, is an excellent example. A proxy can:

- Be stateful or stateless.
- Recurse automatically (send Requests automatically) upon receipt of a 3xx SIP response code to the Contact address(es) contained in the Response header.
- Use the *Record-Route* header to ensure that subsequent requests also go through it.
- Act as a forking proxy to proxy to multiple destinations, either in parallel or in sequence.

With the SIP Servlet API, all of these options are simple attributes of the Proxy object. The container-managed Proxy deals with all low level details like finding a target set (based on *Request-URI* or *Route* headers), applying RFC rules if a strict router is upstream or downstream, creating multiple client transactions, correlating responses, choosing the best response, and so forth.

Using the SIP Servlet API

This section describes additional important interfaces and constructs of the SIP Servlet API, and includes examples.

Note:

JSR 359 defines a new method for creating SIP servlets using Plain Old Java Objects (POJOs) in conjunction with SIP specific annotations as well as other annotations defined by Java EE Common Dependency Injections (CDIs), which can significantly reduce the amount of code and complexity. JSR 359 is, however, fully backwards compatible with 1.x applications, and those examples are left intact in this section.

The SipServlet Object

The `SipServlet` class extends the `GenericServlet` class in the servlet base package. The service method dispatches the SIP message to either `doRequest()` or `doResponse()`, and in turn the requests are directed to the doXXX methods for Requests such as `doInvite`, `doSubscribe`, and so forth, or to doXXX methods for Responses such as `doSuccessResponse` and `doErrorResponse`.

If you are creating a SIP Servlet using POJOs and annotations, you can use the @SipServlet annotation in conjunction with the following method specific annotations:

- @Invite
- @Ack
- @Options
- @Bye
- @Cancel
- @Register
- @Prack
- @Subscribe
- @Notify
- @Message
- @Info
- @Update
- @Refer
- @Publish

The servlet-mapping element defined in the deployment descriptor can define the rule that MUST be satisfied before invoking a particular Servlet. The mapping rules have a well-defined grammar in JSR 116. Example 2-1 shows a mapping that invokes a Servlet only if the Request is an INVITE and the host part of the Request-URI contains the string “example.com". See "SIP Servlet API Service Invocation" for more information on servlet mapping rules.

Example 2-1 Example Servlet Mapping Rule

```
pattern
  <and>
    <equal>
```

```
        <var>request.method</var>
        <value>INVITE</value>
      </equal>
      <contains ignore-case="true">
        <var>request.from.uri.host</var>
        <value>example.com</value>
      </contains>
    </and>
</pattern>
```

There is normally only one `SipServlet` object accessed by concurrent Requests, so it is not a place to define any call- or session- specific data structure. The doXXX methods in the application generally implement the business logic for a given request. Consider Example 2-2.

Example 2-2   Example SIP Servlet

```
1: package test;
2: import javax.servlet.sip.SipServlet;
3: import javax.servlet.sip.SipServletRequest;
4: import java.io.IOException;
5: public class SimpleUasServlet extends SipServlet {
6:   protected void doInvite(SipServletRequest req)
7:      throws IOException {
8:     req.createResponse(180).send();
9:     req.createResponse(200).send();
10:  }
11:  protected void doBye(SipServletRequest req) throws IOException {
12:    req.createResponse(200).send();
13:    req.getApplicationSession().invalidate();
14:  }
15: }
```

Example 2-2 shows a simple UAS Servlet that is invoked on an incoming INVITE Request (triggered by a rule similar to the one defined in Example 2-1). The container invokes the application by invoking the `doInvite` method. The application chooses to send a 180 Response (line 8) followed by a 200 Response (line 9). The application does nothing with the ACK, which would be sent by the UAC. In this case the container receives the ACK and silently ignores it. If it were a stateful proxy it would have proxied it.

Example 2-3 shows how the same class could be written using an annotated POJO.

Example 2-3   Example SIP Servlet Using an Annotated POJO

```
package test;
import javax.inject.Inject;
import javax.servlet.sip.SipFactory;
import javax.servlet.sip.SipServletRequest;
import javax.servlet.sip.annotation.Bye;
import javax.servlet.sip.annotation.Invite;
import javax.servlet.sip.annotation.SipServlet;

@SipServlet(loadOnStartup = 1)
public class SimpleUasServlet {
  @Inject SipFactory sipFactory;
  @Invite
  public void handleInviteRequest(SipServletRequest req) throws IOException {
    req.createResponse(180).send();
    req.createResponse(200).send();
  }
  @Bye
  public void handleByeRequest(SipServletRequest req) throws IOException {
    req.createResponse(200).send();
    req.getApplicationSession().invalidate();
  }
}
```

SIP Factory

As its name suggests, this class is used to create various SIP Servlet API objects such as `Request`, `SipApplicationSession`, `Addresses`, and so forth. An application acting as a UA can use it to create a new Request. Requests created through the factory have a new Call-ID (with the exception of a particular method for B2BUAs in which the application can chose to re-use the existing Call-ID on the upstream leg) and do not have a tag in the To header. The Factory object can be retrieved using the `javax.servlet.sip.SipFactory` attribute on the ServletContext.

See the "findme" example installed with Converged Application Server for an example of obtaining a factory object using `SipFactory`.

If you are using annotated POJOs, you can use the CDI @Inject annotation to inject the SipFactory into your servlet class:

```
@Inject SipFactory sipFactory;
```

SIP Messages

There are two classes of SIP messages: `SipServletRequest` and `SipServletResponse`. These classes respectively represent SIP Requests (INVITE, ACK, INFO, and so forth) and Responses (1xx, 2xx, and so forth). Messages are delivered to the application through various doXXX methods defined in the `SipServlet` class.

SIP is an asynchronous protocol and therefore it is not obligatory for an application to respond to a Request when the `doRequest`(doXXX) method is invoked. The application may respond to the Request at a later stage, because they have access to the original Request object.

Both the `SipServletRequest` and `SipServletResponse` objects are derived from the base `SipServletMessage` object, which provides some common accessor/mutator methods such as `getHeader()`, `getContent()`, and `setContent()`. The `SipServletRequest` defines many useful methods for Request processing:

- `SipServletRequest.createResponse()` creates an instance of the `SipServletResponse` object. This represents the Response to the Request that created it. Similarly, `SipServletRequest.createCancel()` creates a CANCEL Request to a previously sent Request.

> **Note:**
>
> The CANCEL is sent if the UAC decides to not proceed with the call if it has not received a response to the original request. Sending a CANCEL if you have received a 200 response or not received a 100 response would be wrong protocol behavior, luckily the SIP Servlet API steps up to rescue here too. The UAC application can create and send a CANCEL oblivious to these details. The container ensures that a CANCEL is sent out only if a 1xx response code is received, and any response >200 is not received.

- `SipServletRequest.getProxy()` returns the associated Proxy object to enable an application to perform proxy operations.
- `SipServletRequest.pushRoute(SipURI)` enables a UAC or a proxy to route the request through a server identified by the SipURI. The effect of this method is to add a Route header to the request at the top of the Route header list.

Another method of interest is `SipServletRequest.isInitial()`. It is important to understand the concept of initial and subsequent requests, because an application may treat each one differently. For example, if an application receives a Re-INVITE request, it is delivered to the Servlet's `doInvite()` method, but the `isInitial()` method returns `false`.

Initial requests are usually requests outside of an established dialog, of which the container has no information. Upon receiving an initial Request, the container determines which application should be invoked; this may involve looking up the Servlet-mapping rules. Some Requests create dialogs, so any Request received after a dialog is established falls into the category of a "subsequent" Request. Closely-linked with the dialog construct in SIP is the `SipSession` object (see "SipSession ").

In the `SipServletResponse` object, one particular method of interest is `createAck()`. `createAck()` creates an ACK Request on a 2xx Response received for the INVITE transaction. ACKs for non-2xx responses of the INVITE transaction are created by the container itself.

SipSession

The `SipSession` roughly corresponds to a SIP dialog. For UAs the session maintains the dialog state as specified by the RFC, in order to correctly create a subsequent request in a dialog. If an application is acting as a UA (a UAC or a B2BUA), and after having processed an initial request wants to send out a subsequent request in a dialog (such as a Re-INVITE or BYE), it must use `SipSession.createRequest()` rather than one of `SipFactory` methods. Using a factory method would result in requests being created “out of dialog".

The `SipSession` is also a place for an application to store any session-specific state that it requires. An application can set or unset attributes on the `SipSession` object, and these attributes are made available to the application over multiple invocations.

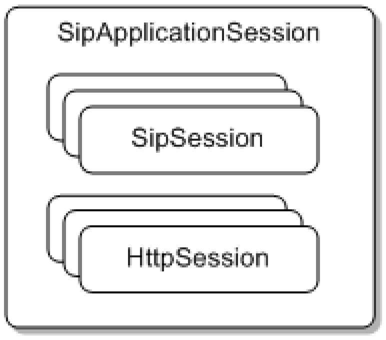
`SipSession` also provides the `SipSession.setHandler(String nameOfAServlet)` method, which assigns a particular Servlet in the application to receive subsequent Requests for that `SipSession`.

SipApplicationSession

The `SipApplicationSession` logically represents an instance of the application itself. An application may have one or more protocol sessions associated with it, and these protocol sessions may be of type `SipSession` or `HttpSession` as of JSR 116. Applications can also store application-wide data as an attribute of the `SipApplicationSession`.

Any attribute set on a `SipApplicationSession` object or its associated `SipSession` is visible only to that particular application. The SIP Servlet API defines a mechanism by which more than one application can be invoked on the same call. This feature is known as application composition. `SipApplicationSession` provides a `getSessions()` method that returns the protocol sessions associated with the application session. The image below shows the containment hierarchy of the different sessions in the SIP Servlet API.

Figure 2-3 Relationship Between Session Object Types



The `encodeUri(URI)` method in the `SipApplicationSession` interface is of particular interest. This method encodes the SipApplication identifier with the URI specified in the argument. If the container receives a new request with this encoded URI, even if on a different call, it associates the encoded `SipApplicationSession` with this Request. This method can link two disparate calls, and it can be used in a variety of other ways. `SipApplicationSession` is also associated with application session timers (see Application Timers).

Application Timers

The SIP Servlet API provides a timer service that applications can use. The `TimerService` interface can be retrieved using a `ServletContext` attribute, and it defines a `createTimer(SipApplicationSession appSession, long delay, boolean isPersistent, java.io.Serializable info)` method to start an application-level timer.

The `SipApplicationSession` is implicitly associated with application-level timers. When a timer fires, the container invokes an application-defined `TimerListener` and passes it the `ServletTimer` object. The listener can use the `ServletTimer` object to retrieve the `SipApplicationSession`, which provides the correct context for the timer's expiry.

SIP Servlet Application Example: Converged SIP and HTTP Application

In terms of the SIP Servlet API, a converged application is one that involves more than one protocol, in this case SIP and HTTP. The example below presents an example of a simple JSP page which can be accessed through an HTTP URL.

Example JSP Showing HTTP and SIP Servlet Interaction

Example 2-4   Example JSP Showing HTTP and SIP Interaction

```
1:<html>
2:<body>
3: <%
4:  if (request.getMethod().equals("POST")) {
5:   javax.servlet.sip.SipFactory factory =
6:     (javax.servlet.sip.SipFactory)
application.getAttribute(javax.servlet.sip.SipServlet.SIP_FACTORY);
7:   javax.servlet.sip.SipApplicationSession appSession =
8:      factory.createApplicationSession();
9:   javax.servlet.sip.Address to =
10:     factory.createAddress("sip:localhost:5080");
11:   javax.servlet.sip.Address from =
12:     factory.createAddress("sip:localhost:5060");
13:   javax.servlet.sip.SipServletRequest invite =
14:      factory.createRequest(appSession, "INVITE", from, to);
15:   javax.servlet.sip.SipSession sess = invite.getSession(true);
16:     sess.setHandler(“sipClickToDial");
17:   //invite.setContent(content, contentType);
18:   invite.send();
19:  }
20:%>
21:<p>
22:Message sent ...
23:</p>
24:</body>
25:</html>
```

The JSP example would need to be packaged in the same application as a SIP Servlet. The entire application is a skeleton of a click-to-dial application (called `sipClickToDial`), where by clicking on a Web page you initiate a SIP call.

The HTTP Servlet creates a SIP Request from a factory and sends it to a SIP URI. When an HTTP POST Request is sent to the HTTP Servlet it obtains the `SipFactory` on line 5-6. Next, it creates an application session (line 7-8). The application session is the center piece for all of the application's SIP and HTTP interactions. The overall purpose is to send out a SIP Request, which is done in lines 13-14, but first the application creates the From and To headers to be used when forming the INVITE request.

On line 16 the application assigns a handler to the `SipSession` that is associated with the INVITE Request that was created, and this ensures that the Response sent by a UAS that receives the request is dispatched to a SIP Servlet for processing.

SIP Servlet Application Example: SUBSCRIBE and NOTIFY

In the example shown below, the application receives a SUBSCRIBE Request and sends out a NOTIFY Request. The application then waits for the notification recipient for three seconds, and if does not receive a success response (a 2xx class response), then it may take some other action (for example, log a message).

Example 2-5 Example of SUBSCRIBE and NOTIFY Handling

```
1:public class Sample_TimerServlet extends SipServlet
2:  implements TimerListener {
3:  private TimerService timerService;
4:  private static String TIMER_ID = "NOTIFY_TIMEOUT_TIMER";
5:  public void init() throws ServletException {
6:    try {
7:      timerService =
8:(TimerService)getServletContext().getAttribute
9:      ("javax.servlet.sip.TimerService");
10:    }
11:    catch(Exception e) {
12:      log ("Exception initializing the servlet "+ e);
13:    }
14:  }
15:  protected void doSubscribe(SipServletRequest req)
16:  throws ServletException, IOException {
17:    req.createResponse(200).send();
18:    req.getSession().createRequest("NOTIFY").send();
19:    ServletTimer notifyTimeoutTimer =
20:      timerService.createTimer(req.getApplicationSession(), 3000,
21:  false, null);
22:    req.getApplicationSession().setAttribute(TIMER_ID,
23:notifyTimeoutTimer);
24:  }
25:  protected void doSuccessResponse(SipServletResponse res)
26:  throws javax.servlet.ServletException, java.io.IOException {
27:    if (res.getMethod().equals("NOTIFY")) {
28:      ServletTimer notifyTimeoutTimer =
29:  (ServletTimer)(res.getApplicationSession().getAttribute(TIMER_ID));
30:      if (notifyTimeoutTimer != null) {
31:        notifyTimeoutTimer.cancel();
32:        res.getApplicationSession().removeAttribute(TIMER_ID);
33:      }
34:    }
35:  }
36:  public void timeout(ServletTimer timer) {
37:    // This indicates that the timer has fired because a 200 to
38:    // NOTIFY was not received. Here you can take any timeout
39:    // action.
40:    // .........
41:    timer.getApplicationSession().removeAttribute
("NOTIFY_TIMEOUT_TIMER");
42:  }
43:}
```

The Servlet itself implements `TimerListener` so that it will be notified of the timeout. The example starts by obtaining the `TimerService` from the `ServletContext` in lines 7-9. The timer is then set for 3000 ms (3 seconds) upon receiving the SUBSCRIBE request on line 20. Note that the timer could be set at any stage. There is also an option to attach an object to the timer. The object could be used as an identifier or an invocable message at a later stage. This sample simply associates the timer with a literal.

After sending the NOTIFY the application creates the timer and saves its reference in the `SipApplicationSession` for later use on line 22.

If the application receives a 200 response to the NOTIFY, it can then extract the timer reference and cancel the timer (line 25). However, if no response is received in 3 seconds, then the timer fires and the container calls the `timeout()` callback method (line 36).

The example above can be rewritten using an annotated POJO as shown in the example below.

Example 2-6 Example of SUBSCRIBE and NOTIFY Handling using an Annotated POJO

```
@SipServlet(loadOnStartup = 1);
public class Sample_TimerServlet {

  // Inject the TimerService using the @Resource annotation...
  @Resource TimerService timerService;
  private static String TIMER_ID = "NOTIFY_TIMEOUT_TIMER";

  @Subscribe
  protected void handleSubscribeRequest(SipServletRequest req) throws IOException {
    req.createResponse(200).send();
    req.getSession().createRequest("NOTIFY").send();
    ServletTimer notifyTimeoutTimer = timerService.createTimer(
    req.getApplicationSession(), 3000, false, null);
    req.getApplicationSession().setAttribute(TIMER_ID, notifyTimeoutTimer);
  }

  @SuccessResponse
  protected void handleSuccessResponse(SipServletResponse res) throws IOException {
    if (res.getMethod().equals("NOTIFY")) {
      ServletTimer notifyTimeoutTimer =
      (ServletTimer)(res.getApplicationSession().getAttribute(TIMER_ID));
      if (notifyTimeoutTimer != null) {
        notifyTimeoutTimer.cancel();
        res.getApplicationSession().removeAttribute(TIMER_ID);
      }
    }
  }

  public void timeout(ServletTimer timer) {
    // This indicates that the timer has fired because a 200 to
    // NOTIFY was not received. Here you can take any timeout
    // action.
    // .........
    timer.getApplicationSession().removeAttribute ("NOTIFY_TIMEOUT_TIMER");
  }
}
```

Converged Application Server Profile API

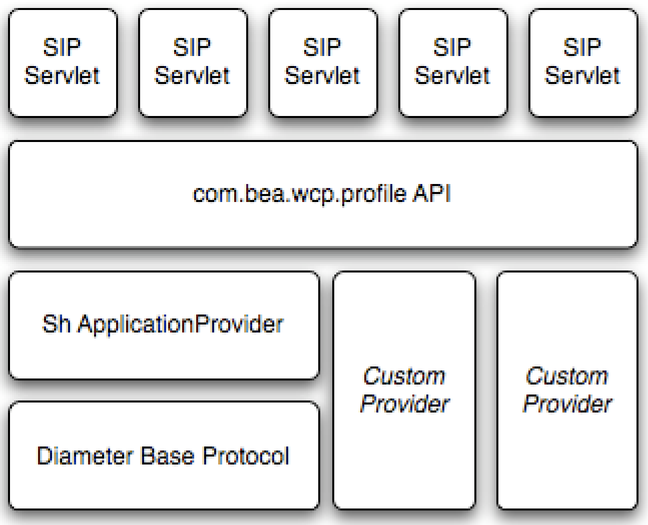
The IMS specification defines the Sh interface as the method of communication between the Application Server (AS) function and the Home Subscriber Server (HSS), or between multiple IMS Application Servers. The AS uses the Sh interface in two basic ways:

- To query or update a user's data stored on the HSS
- To subscribe to and receive notifications when a user's data changes on the HSS

The user data available to an AS may be defined by a service running on the AS (repository data), or it may be a subset of the user's IMS profile data hosted on the HSS. The Sh interface specification, 3GPP TS 29.328 V5.11.0, defines the IMS profile data that can be queried and updated via Sh. All user data accessible via the Sh interface is presented as an XML document with the schema defined in 3GPP TS 29.328.

The IMS Sh interface is implemented as a provider to the base Diameter protocol support in Converged Application Server. The provider transparently generates and responds to the Diameter command codes defined in the Sh application specification. A higher-level Profile Service API enables SIP Servlets to manage user profile data as an XML document using XML Document Object Model (DOM). Subscriptions and notifications for changed profile data are managed by implementing a profile listener interface in a SIP Servlet.

Figure 2-4 Profile Service API and Sh Provider Implementation



Converged Application Server includes only a single provider for the Sh interface. Future versions of Converged Application Server may include new providers to support additional interfaces defined in the IMS specification. Applications using the profile service API will be able to use additional providers as they are made available.

Using Document Keys for Application-Managed Profile Data

Servlets that manage profile data can explicitly obtain an Sh XML document from a factory using a key, and then work with the document using DOM.

The document selector key identifies the XML document to be retrieved by a Diameter interface, and uses the format *protocol*`://`*uri*`/reference_type[/`*access_key*`]`.

The example below summarizes the required document selector elements for each type of Sh data reference request.

Table 2-1 Summary of Document Selector Elements for Sh Data Reference Requests

Data Reference Type	Required Document Selector Elements	Example Document Selector
RepositoryData	sh://uri/reference_type/Service-Indication	sh://sip:user@oracle.com/RepositoryData/Call Screening/
IMSPublicIdentity	sh://uri/reference_type/[Identity-Set] where Identity-Set is one of: • All-Identities • Registered-Identities • Implicit-Identities	sh://sip:user@oracle.com/IMSPublicIdentity/Registered-Identities
IMSUserState	sh://uri/reference_type	sh://sip:user@oracle.com/IMSUserState/
S-CSCFName	sh://uri/reference_type	sh://sip:user@oracle.com/S-CSCFName/
InitialFilterCriteria	sh://uri/reference_type/Server-Name	sh://sip:user@oracle.com/InitialFilterCriteria/www.oracle.com/
LocationInformation	sh://uri/reference_type/(CS-Domain | PS-Domain)	sh://sip:user@oracle.com/LocationInformation/CS-Domain/
UserState	sh://uri/reference_type/(CS-Domain | PS-Domain)	sh://sip:user@oracle.com/UserState/PS-Domain/
Charging information	sh://uri/reference_type	sh://sip:user@oracle.com/Charging information/
MSISDN	sh://uri/reference_type	sh://sip:user@oracle.com/MSISDN/

Converged Application Server provides a helper class, `com.bea.wcp.profile.ProfileService`, to help you easily retrieve a profile data document. The `getDocument()` method takes a constructed document key, and returns a read-only `org.w3c.dom.Document` object. To modify the document, you make and edit a copy, then send the modified document and key as arguments to the `putDocument()` method.

See "Using the Profile Service API (Diameter Sh Interface)" in *Converged Application Server Diameter Application Development Guide* for more information.

Monitoring Profile Data

The IMS Sh interface enables applications to receive automatic notifications when a subscriber's profile data changes. Converged Application Server provides an easy-to-use API for managing profile data subscriptions. A SIP Servlet registers to receive notifications by implementing the `com.bea.wcp.profile.ProfileListener` interface, which consists of a single update method that is automatically invoked when a change occurs to profile to which the Servlet is subscribed. Notifications are not sent if that same Servlet modifies the profile information (for example, if a user modifies their own profile data).

Actual subscriptions are managed using the subscribe method of the **`com.bea.wcp.profile.ProfileService`** helper class. The subscribe method requires that you supply the current **`SipApplicationSession`** and the key for the profile data document you want to monitor. See "Using Document Keys for Application-Managed Profile Data" for more information.

Applications can cancel subscriptions by calling `ProfileSubscription.cancel()`. Also, pending subscriptions for an application are automatically cancelled if the application session is terminated.

Example 2-7 shows sample code for a Servlet that implements the `ProfileListener` interface.

Example 2-7 Sample Servlet Implementing ProfileListener Interface

```
package demo;
    import com.bea.wcp.profile.*;
    import javax.servlet.sip.SipServletRequest;
    import javax.servlet.sip.SipServlet;
    import org.w3c.dom.Document;
    import java.io.IOException;
    public class MyServlet extends SipServlet implements ProfileListener {
      private ProfileService psvc;
      public void init() {
        psvc = (ProfileService)
getServletContext().getAttribute(ProfileService.PROFILE_SERVICE);
      }
      protected void doInvite(SipServletRequest req) throws IOException {
        String docSel = "sh://" + req.getTo() + "/IMSUserState/";
        // Subscribe to profile data.
        psvc.subscribe(req.getApplicationSession(), docSel, null);
}
      public void update(ProfileSubscription ps, Document document) {
        System.out.println("IMSUserState updated: " + ps.getDocumentSelector());
      }
    }
```

The `ProfileListener` interface is handled similar to the `TimerService` provided by JSR 116 for application timers. Multiple Servlets in an application may implement the `ProfileListener` interface, but only one Servlet may act as a listener. The SIP deployment descriptor for the application must designate the profile listener class in the set of listeners as shown in Example 2-8.

Example 2-8 Declaring a ProfileListener

```
<listener>
<listener-class>com.foo.MyProfileListener</listener-class>
Declaring a ProfileListener
</listener>
<listener>
  <listener-class>com.foo.MyProfileListener</listener-class>
</listener>
```

Example 2-7 can be rewritten using an annotated POJO as shown in Example 2-9.

Example 2-9 Sample Servlet Implementing ProfileListener Interface Using an Annotated POJO

```
package demo;
// Includes excluded for brevity...
@SipServlet(loadOnStartup = 1);
public class MyServlet implements ProfileListener {
  private ProfileService psvc;
  @Inject SipFactory sipFactory;
  public void init() {
    psvc = (ProfileService)
getServletContext().getAttribute(ProfileService.PROFILE_SERVICE);
```

```
  }
  @Invite
  protected void handleInvite(SipServletRequest req) throws IOException {
    String docSel = "sh://" + req.getTo() + "/IMSUserState/";
    // Subscribe to profile data.
    psvc.subscribe(req.getApplicationSession(), docSel, null);
  }
  public void update(ProfileSubscription ps, Document document) {
    System.out.println("IMSUserState updated: " + ps.getDocumentSelector());
  }
}
```

Developing "Zero Downtime" Upgradable Applications

With Converged Application Server, you can upgrade a deployed SIP application to a newer version without losing existing calls being processed by the application. This type of application upgrade is accomplished by deploying the newer application version alongside the older version. Converged Application 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 un-deploy it.

Converged Application Server's upgrade feature ensures that no calls are dropped while during the upgrade of a production application. The upgrade process also enables you to revert or rollback the process of upgrading an application. If, for example, you determine that there is a problem with the newer version of the deployed application, you can simply un-deploy the newer version. Converged Application Server then automatically directs all new requests to the older application version.

Requirements and Restrictions for Upgrading Deployed Applications

To use the application upgrade functionality of Converged Application Server:

- You must assign version information to your updated application in order to distinguish it from the older application version. Note that only the newer version of a deployed application requires version information; if the currently-deployed application contains no version designation, Converged Application Server automatically treats this application as the “older" version.
- Both the deployed application and the updated application must provide only SIP protocol functionality. You cannot upgrade converged HTTP/SIP applications using these procedures.
- 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. Converged Application Server provides SipApplicationRuntimeMBean methods for obtaining the base application name and version identifier, as well as determining whether the current application version is active or retiring.
- When applications take part in a composed application (using application composition techniques), Converged Application Server always uses the latest version of an application when only the base name is supplied.

Developing IR.92 Supplementary Services

Oracle Communication Converged Application Server provides support for GSM Association's (GSMA) IR.92 specifications for delivering Voice over LTE (VoLTE). You can implement these services using either the SIP Servlet 2.0 (JSR 359) or the Service Foundation Toolkit (SFT).

SFT provides enhanced APIs that you can use to quickly and easily implement applications for delivering IR.92-compliant supplementary services over VoLTE. The APIs provide support for supplementary services such as Call Forwarding, Incoming and Outgoing Call Barring, ID Presentation and Restriction, Multi-Party Conferencing, and Message Waiting Indication (MWI).

About Converged Application Server and VoLTE

GSM Association's (GSMA) IR.92 specifications defines the IP Multimedia Subsystem (IMS) profile for delivering Voice over LTE (VoLTE). The GSMA VoLTE initiative has defined IMS as the common way to deliver voice and messaging services over mobile broadband all-IP networks.

In September 2010, GSMA published the IREG Permanent Reference Document IR.92, which outlines the specifications for migrating 2G and 3G mobile voice, video and messaging services to 4G mobile broadband networks, such as LTE.

This chapter describes the following IR.92 call control services, and their implementation in Converged Application Server:

- Communication Diversion
- Communication Barring
- Communication Hold
- Originating Identification Presentation and Restriction
- Communication Waiting
- Message Waiting Indication
- Announcement Support

Communication Diversion

Communication Diversion (also referred to as Call Diversion or Call Forwarding) lets users of the service (the called party, or callee) forward incoming calls to another phone number (the third party). The third party may be a mobile telephone, voice-mail box, or other telephone number where the desired called party is situated.

With Communication Diversion activated:

- Users can continue to make outgoing calls from their telephone while incoming calls are forwarded.
- When the telephone number the user's calls are being forwarded to is busy, callers to the forwarded number will receive a busy signal.

Converged Application Server supports the following IR.92 defined Communication Diversion modes for VoLTE:

- Communication Forwarding Unconditional 3GPP TS 24.604
- Communication Forwarding on No Reply 3GPP TS 24.604
- Communication Forwarding on not Logged in 3GPP TS 24.604
- Communication Forwarding on Busy 3GPP TS 24.604
- Communication Forwarding on not Reachable 3GPP TS 24.604

Communication Diversion applications forward calls by removing the callee (the person to whom the call is being made), and adding a new participant (the third party) in the calle's place.

Communication Barring

Communication Barring lets users bar (or restrict) certain or all types of calls to and from their phone. For example, a user can restrict outgoing calls, outgoing international calls, or incoming calls from undesirable callers.

Converged Application Server supports the following IR.92 defined Call Barring modes for VoLTE:

- Barring of All Incoming Calls 3GPP TS 24.611
- Barring of All Outgoing Calls 3GPP TS 24.611
- Barring of Outgoing International Calls 3GPP TS 24.611
- Barring of Outgoing International Calls—ex Home Country 3GPP TS 24.611
- Barring of Incoming Calls When Roaming 3GPP TS 24.611

To implement international call barring, the Communication Barring application must have access to the phone number of the participant. The application can access this information from various sources, such as the Registrar, to determine roaming status. Similarly, the profile of the user—such as country of origin—can be obtained by the application using other interfaces (for example, the Diameter interface).

Communication Hold

Communication Hold (also referred to Call Hold) allows a user to suspend a communication session—the reception of media stream(s) from an established IP multimedia session—and resume the media stream(s) at a later time. Placing a Communication Hold on an ongoing session is achieved by sending a Session Description Protocol (SDP) offer where each of the communications (media streams) to be held are marked with the `sendonly` attribute if they were previously bidirectional media streams. To resume the session, a new SDP offer is issued in which each of the held media streams is marked with the default `sendrecv` attribute.

Communication Hold also allows an AS to play music or an announcement to the held party. This is achieved using an AS that acts as a third-party call controller (3PCC), and replaces the existing session of one of the users with a session originating from an application server that plays the announcement or music until the user's session is resumed. See the 3GPP TS 24.628 specification for more information on the playing of announcements during Communication Hold.

Setting the Communication Hold Bandwidth

The 3GPP TS 24.610 specification requires that the AS of the User Equipment (UE) invoking a media stream whose SDP session attribute is `recvonly` use a lower bandwidth. The SDP specifies a lower bandwidth by setting the bandwidth (the `b=` line in the SDP) to a lower value. The `b=` line contains two elements:

- The bandwidth value expressed in kilo bits per second (kbps).
- An alphanumeric modifier that indicates the communication session or media stream to which to apply the specified bandwidth value.

The modifiers whose bandwidth values are specified by SFT are:

- AS: Application Specific Maximum, which specifies the total bandwidth for a single media stream from one source.
- RS: RTCP bandwidth allocated to active data senders.
- RR: RTCP bandwidth allocated to other participants (receivers) in the RTP session.

When the bandwidth setting is enabled, SFT sets the default value for the AS bandwidth to zero (`b=AS:0`). The `b=RR:` and `b=RS:` parameters are set to a value of 800 kbps, which is high enough to allow the continuation of the RTCP flow: `b=RR:800` and `b=RS:800`

Example 2-10 Bandwidth Line in the Session Description Protocol

```
v=0
o=alice 2890844526 2890842807 IN IP4 126.16.64.4
s=SDP Seminar
c=IN IP4 224.2.17.12/127
t=2873397496 2873404696
m=audio 49170 RTP/AVP 0
b=AS:0
b=RR:800
b=RS:800
```

Note:

The 3GPP TS 24.610 specification recommends that the AS modify the bandwidth for media streams whose SDP session attribute is `recvonly`. Media streams whose SDP session attribute are `inactive`, `sendonly`, and `sendrecv` are not affected.

While the 3GPP TS 24.610 specification recommends these values to preserve network bandwidth when a communication is placed on hold, you may need to adjust the bandwidth to better suit the requirements of the Communication Hold application.

Originating Identification Presentation and Restriction

Converged Application Server supports the following Identity Presentation and Restriction services:

- **Originating Identification Presentation (OIP)**: Enables the called party to receive a network generated and trusted identity of the calling user on the screen of the mobile device. The originating user may also present a custom identity to be seen at the called party. The user generated identity is usually screened by the network of the originating user.

- **Originating Identification Restriction (OIR)**: Invoked when the calling user does not want their identity to be shown to the called party. In such cases, the network of the originating user signals to the network of the called user, to withhold the identity of the calling user.
- **Terminating Identification Presentation (TIP)**: Enables the calling party to receive the identification information of the remote party from the network. This information is provided to the originating party once the IMS communication has been accepted between the endpoints. The information is delivered regardless of the capability of the handset to process such information at the originating end.
- **Terminating Identification Restriction (TIR)**: Provides the terminating party with an option to restrict the identity to be presented to the originating party of the IMS communication. Logically speaking, TIR is the opposite of TIP.

To support the Identity Presentation and Restriction services listed above:

- UE and IMS core network must support the SIP procedures described in the 3GPP TS 24.607 specification. Service configuration, as described in Section 4.10 of 3GPP TS 24.607, is optional.
- UE and IMS core network must support the SIP procedures in the 3GPP TS 24.608 specification. Service configuration, as described in section 4.9 of 3GPP TS 24.608, is optional.

Privacy Service Behavior

The privacy service role is instantiated by a network intermediary. Typically this function is performed by entities that act as SIP proxy servers. The privacy service is designed to provide privacy functions for SIP messages that cannot otherwise be provided by the UAs themselves. Table 2-2 lists the semantics of each `priv-value`, and the RFC that defines them.

Table 2-2 Types of Privacy Service Behaviors

Privacy Type	Description
`user`	Request that privacy services provide a user-level privacy function. See RFC 3323, "A Privacy Mechanism for the Session Initiation Protocol (SIP)" for more information.
`header`	Request that privacy services modify headers that cannot be set arbitrarily by the user. For example, the Contact and Via headers. See RFC 3323, "A Privacy Mechanism for the Session Initiation Protocol (SIP)" for more information.
`session`	Request that privacy services provide privacy for session media. See RFC 3323, "A Privacy Mechanism for the Session Initiation Protocol (SIP)" for more information.
`none`	Privacy services must not perform any privacy function. See RFC 3323, "A Privacy Mechanism for the Session Initiation Protocol (SIP)" for more information.
`critical`	Privacy service must perform the specified services or fail the request. See RFC 3323, "A Privacy Mechanism for the Session Initiation Protocol (SIP)" for more information.

Table 2-2 (Cont.) Types of Privacy Service Behaviors

Privacy Type	Description
`id`	Privacy requested for Third-Party Asserted Identity. See RFC 3325, "Private Extensions to the Session Initiation Protocol (SIP) for Asserted Identity within Trusted Networks" for more information.
`history`	Privacy requested for History-Info headers. See RFC 4244, "An Extension to the Session Initiation Protocol (SIP) for Request History Information" for more information.

RFC 5379 describes privacy considerations and the recommended treatment of each SIP header that may reveal user-privacy information. Section 5, "Recommended Treatment of User-Privacy-Sensitive Information" of RFC 5379 describes how each header affects privacy, the desired treatment of the value by the user agent and privacy service, and other details needed to ensure privacy.Table 2-3 lists the recommended treatment for each `priv-value` contained in the SIP header. See "Section 5" of RFC 5379 "Guidelines for Using the Privacy Mechanism for SIP" for more information.

Table 2-3 Treatment of User-Privacy Information in Target SIP Headers

Target SIP Headers	Where	User	Header	Session	ID	History
Call-ID	R	Anonymize				
Call-Info	Rr	Delete	Not added			
Contact	R		Anonymize			
From	R	Anonymize				
History-Info	Rr		Delete	Delete		Delete
In-Reply-To	R	Delete				
Organization	Rr	Delete	Not added			
P-Asserted-Identity	Rr		Delete		Delete	
Record-Route	Rr		Anonymize			
Referred-By	R	Anonymize				
Reply-To	Rr	Delete				
Server	r	Delete	Not added			
Subject	R	Delete				
User-Agent	R	Delete				
Via	R		Anonymize			
Warning	r	Anonymize				

Providing Privacy for the History-Info Header

The History-Info header (defined in RFC 4244) provides a way of capturing any redirection information that may have occurred during a particular call. Without the History-Info header the redirecting information would be lost. The History-Info header is generated when a SIP request is re-directed, and can appear in any SIP request not associated with a dialog. The History-Info header is generated by a User Agent or proxy and is passed from one entity to another through requests and responses.

Communication Waiting

Communication Waiting (also referred to as Call Waiting) informs a user (or the user equipment) that limited resources are available for incoming calls. Typically this means that the callee is involved in a communication session with another caller, and is not able to answer the incoming call from the second caller. Communication Waiting provides a mechanism by which you can create an application to inform a user that there is a second incoming call. The user then has the choice of accepting, rejecting, or ignoring the waiting call. Converged Application Server supports the 3GPP TS 24.615 and the GSMA IR.92 specifications.

Supporting Network- and Terminal-based Communication Waiting

When using SFT to develop Communication Waiting services, Converged Application Server supports both network- and terminal-based Communication Waiting.

About Network-based Communication Waiting

Network-based Communication Waiting occurs when the AS determines that one of the following conditions has occurred:

- The SIP INVITE request fulfills the Network Determined User Busy (NDUB) condition for the callee.
- The caller receives a SIP message 486 Busy Here (indicating that the callee is busy) with a 370 Warning in the SIP header field indicating that there is insufficient bandwidth for the call to complete.

To support network-based Communication Waiting, the AS performs the following functions in response to receiving an appropriate Communication Waiting condition:

1. Modifies the SIP INVITE request and forwards or re-sends it to User B.
2. Provides an announcement to User C upon receipt of a 180 Ringing response from User B.
3. Inserts an Alert-Info header field set to `urn:alert:service:call-waiting` in the 180 Ringing response and forwards it to User C.
4. Rejects the communication by sending a 486 Busy Here response to User C upon receipt of a 415 Unsupported Media Type response.

About Terminal-based Communication Waiting

Terminal-based Communication Waiting occurs when the AS receives the SIP message 180 Ringing with the Alert-Info header URN Indication Values set to `urn:alert:service:call-waiting`.

To support terminal-based Communication Waiting, the application server performs the following functions in response to receiving an appropriate Communication Waiting condition:

1. Sends an announcement to the calling user (the caller).
2. Sends a 180 Ringing response to the caller.
3. Initiates the Telephony Application Server-Communication Waiting (TAS-CW) timer. This optional timer specifies the amount of time the network will wait for a response from User

B, in response to the communication from User C. The value of the timer is between 0.5 and 2 minutes.

If the TAS-CW timer expires, the AS sends a CANCEL request to User B with a Reason header field set to "SIP," and the cause set to 408 Request Timeout, indicating that the user could not be found in the allotted time. A 480 Temporarily Unavailable response is sent to User C, including a Reason header field set to ISUP Cause Code 19, indicating that there was no answer from the callee.

Message Waiting Indication

Message Waiting Indication (MWI) is a service that informs a user about the status of recorded messages. To use the notification feature, the user must subscribe to a notification service that makes use of the Message Waiting Indication status messages. With the Message Waiting Indication feature you can:

- Send notification when a new subscription arrives.
- Specify when notifications are sent in response to subscriptions.
- Reject subscriptions.
- Terminate subscriptions.

> **Note:**
>
> Typically a voice-mail server manages Message Waiting Indication accounts. When a new message arrives, the voice-mail server typically provides a listener or API that you can resister with to receive notification of new messages. How the application manages the message account is beyond the scope of the SFT Message Waiting Indication APIs.

Message Waiting Indication lets the AS notify a subscriber that there is a message waiting for them. The indication is delivered to the subscriber's UE after they have successfully subscribed to the Message Waiting Indication service. Message Waiting Indication is defined in the 3GPP TS 24.606 specification.

When Converged Application Server receives a SUBSCRIBE message, SFT notifies the MWI application via a SUBSCRIPTION event. RFC 3842 specifies that a NOTIFY message must be sent when accepting new subscriptions, the subscription has expired, or an unsubscribe event occurs. Converged Application Server's Event Notification Service sends these NOTIFY messages automatically.

Announcement Support

Announcements are service-related messages played to a recipient to inform them about the state of a call. Announcements can be provided using either audio or video content. Converged Application Server supports the playing of announcements as defined in the 3GPP TS 24.628 specification.

Converged Application Server supports the following approaches to playing announcements:

- Send the media stream to the recipient of the announcement for playback.

This approach uses a media server and Media Resource Function Processor (MRFP). The media is streamed to the recipient using the Real-time Transport Protocol (RTP) after establishing a media session with the media server. Based on the point-in-time at which the media session is initiated, an early- or non-early media session can be used.

SFT reserves a media resource using the JSR 309 API (the JSR 309 driver used by the media server). The underlying mechanism between the JSR309 driver and MRFP is protocol agnostic.

- Send information about the media content that lets the recipient retrieve and playback the announcement.

  This approach sends a URI identifying the media to the recipient, allowing them to determine whether or not to play the announcement.

Developing Services Using XCAP

Converged Application Server lets you access an XML Document Management Server (XDMS). The XDMS handles the management of user-generated XML documents stored on the network, such as authorization rules and contact and group lists (also referred to as resource lists).

The XML Configuration Access Protocol Server (XCAP server), provides an interface that allows for the manipulation of service-related data stored as XML documents within the XDMS. The XCAP specification defines how an HTTP address (or URI) can identify the way XML documents are stored on an XCAP server. It also defines how the URI can be used to identify entire XML documents, individual elements, or XML attributes that can be retrieved, updated, or deleted.

- An Application Unique ID (AUID), which uniquely identifies the application usage, must be created.
- The XML schema must be defined.
- The default namespace binding, which maps the namespace prefixes to the namespace URIs, must be set.
- The MIME type of the document must be defined.
- The XCAP server must be able to validate the content of each XCAP document that is being modified.
- The data in the XML document must have a well defined semantic.
- Naming conventions for XCAP client URIs must be set.
- Resource interdependencies, how changes to one resource will effect other resources, has to be determined.

The following operations are supported using XCAP:

- Retrieving an item
- Deleting an item
- Modifying an item
- Adding an item

The XCAP addressing mechanism is based on XML Path Language (XPath), a query language for selecting nodes in XML documents. The operations above can be executed on XML documents and elements. Operations to XML attributes are not supported, however, attributes can be handled indirectly by modifying the elements that contain them.

About XCAP and VoLTE

Converged Application Server provides two levels of XCAP support: Access to the XDMS using a base XCAP API that is not specific to any schema, and a high level API providing support for GSMA IR.92 supplementary services using VoLTE as supported by the Service Foundation Toolkit (SFT). The VoLTE version of the XCAP API, supports the following supplementary services:

- 3GPP TS 24.611 Communications Diversion
- 3GPP TS 24.604 Communication Barring
- 3GPP TS 24.607 Originating Identification Presentation and Originating Identification Restriction
- 3GPP TS 24.608 Terminating Identification Presentation and Terminating Identification Restriction
- 3GPP TS 24.615 Communication Waiting

The supported VoLTE functions are:

- Partial operations (adding and removing XML elements)
- Data validation
- Support for 409 XCAP error responses as defined in Section 11 of RFC 4825

3

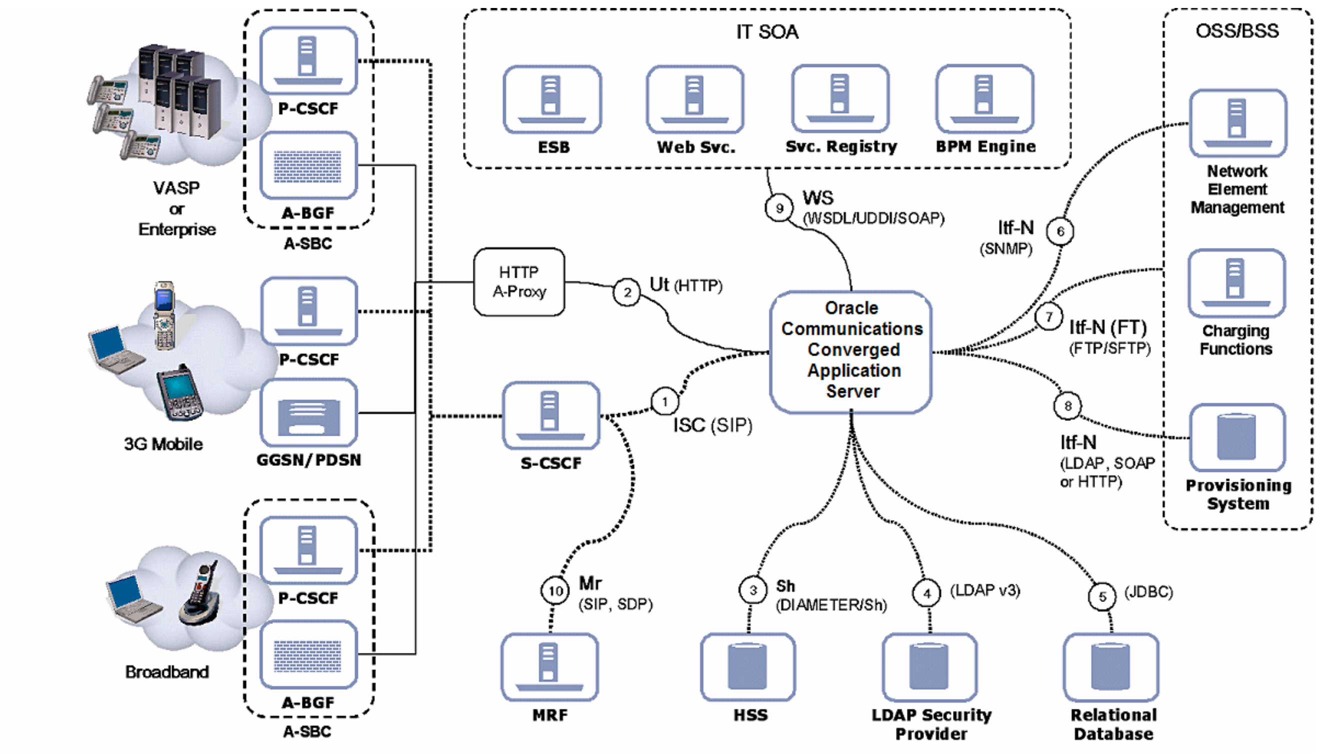
Converged Application Server in the Network

This chapter describes how Oracle Communications Converged Application Server functions in a service provider network.

Converged Application Server in a Typical Service Provider Network

Converged Application Server can be deployed in 3rd Generation Partnership Project (3GPP) R6 compliant IP Multimedia Subsystem (IMS) networks as well as in non-IMS networks. Converged Application Server can interoperate with a number of network functions regardless of which applications or functions it hosts.

Figure 3-1 Converged Application Server Deployed in a Typical Service Provider Network



The following sections provide more information on the role of the Converged Application in the network.

Note:

3GPP R12 Specification Conformance describes Converged Application Server conformance to the requirements introduced in the 3GPP Release 6 specifications.

SIP and IMS Service Control

The Session Initiation Protocol (SIP) interface between the Serving Call Session Control Function (CSCF) and the IMS SIP Application Server (AS) is defined as the IMS Service Control (ISC) reference point. Although ISC is generally compliant with the SIP protocol as defined by the Internet Engineering Task Force (IETF), it introduces several specific procedures and transport layer requirements. SIP usage is often described as the "3GPP SIP Profile."

The ISC reference point does not require that the AS or Serving CSCF add any particular attribute or value to a request or response beyond the standard behavior of a SIP protocol entity. There are, however, a number of SIP methods and headers that are relevant to many of the services that are deployed on the IMS (SIP) AS. In order for the IMS SIP AS to "fully" comply with all of the 3GPP R5 and R6 specifications, many IETF RFCs and drafts would have to be supported. However, it is not reasonable to characterize this as "ISC compliance" because ISC specifically addresses the relationship between the IMS (SIP) AS and the Serving CSCF. From this perspective, ISC compliance is relatively straightforward and is minimally reflected in "Procedures at the AS" defined in 3GPP TS 24.229: "IP Multimedia Call Control Protocol based on Session Initiation Protocol (SIP) and Session Description Protocol (SDP); Stage 3 (Release 6)."

From the perspective of the Converged Application Server, the Serving CSCF is a SIP Proxy and/or User Agent (in the case of the Registration Event Package and third-party registration messages) and is the SIP Application Server's default gateway for SIP requests when the AS instantiates a User Agent Client.

ISC and the 3GPP SIP Profile

The 3GPP requires SIP to be used in a more restricted manner than the IETF specs allow, and also requires a number of additional SIP headers. This use of SIP is often referred to as the "3GPP SIP Profile."

Converged Application Server's SIP Servlet Container provides automated management of session objects, Servlet lifecycle, security, OAM and other functions that are not clearly within the scope of an application's business logic. The SIP Servlet Container allows applications to handle (send/receive) SIP messages with non-standard methods or headers—the container is concerned only with the validation of message syntax, and with the protocol transaction layer.

Converged Application Server uses certain p-headers directly. For example, p-asserted-identity is used as an assertion of identity within the Converged Application Server security framework. Other headers, like the 3GPP p-charging-vector or p-charging-function-address, are relevant only within the scope of the application and have no container-level implications.

Converged Application Server does not programmatically force applications to be compliant with the 3GPP SIP Profile, although applications deployed on Converged Application Server may comply with the SIP Profile as necessary.

AS Session Case Determination Requirement of ISC

When requests are sent to an IMS SIP Application Server by the S-CSCF, the SIP AS is generally required to determine the session case (originating, terminating, or terminating unregistered) of the request, either implicitly or explicitly.

Converged Application Server provides several ways of determining the session case for the request. There are three mechanisms described in the 3GPP standardization that an IMS (SIP) AS may use to make this determination:

- Session Case Specific Addresses (for example, sip:sessioncase_as01.operator.net or sip:as01.operator.net:49494)
- Tokens in the “User Part" of the Request URI (for example, sip:token@as01.operator.net)
- Request URI Parameters (for example, sip:as01.operator.net;parameter)

See "3GPP TS 24.229: IP Multimedia Call Control Protocol based on Session Initiation Protocol (SIP) and Session Description Protocol (SDP); Stage 3 (Release 6)" for more information.

The choice of which mechanism to use is at the discretion of both the Communications Service Provider and the SIP Servlet application deployer. The SIP Servlet API relies on a deployment descriptor file that is packaged with the SIP Servlet Application archive file when it is created. The descriptor explicitly indicates the Service Trigger Points that will be used by the SIP Servlet Container to determine which SIP Servlets to invoke. These Service Trigger Points are sufficient to support any of the methods described above for determining the session case of the request.

See SIP Servlet API Service Invocation for a more detailed description of the Service Trigger Points supported by Converged Application Server.

Transport Layer Issues Related to ISC

The 3GPP Release 6 specifications mandate the use of IPv6 (see IETF RFC 2460: Internet Protocol, Version 6 (IPv6) Specification) for all interfaces, including ISC. Converged Application Server also supports IPv6.

When using TCP, Converged Application Server does not arbitrarily create new connections for each SIP Transaction or Dialog. By default, responses to SIP requests are returned using the connection on which the request was received. If a TCP connection fails, Converged Application Server establishes a new TCP connection to the target host. This may mean that responses to SIP requests are returned using TCP connections that are different from the connection over which the request was sent. Although this conforms to the current best practice and to "IETF RFC 3261: SIP: Session Initiation Protocol," Oracle has discovered that many SIP products on the market demonstrate non-compliant behaviors with regard to handling OSI layer 3 protocols.

Although it is not normally the case that Converged Application Server is deployed directly facing end-user SIP devices, it is important to understand the impact this behavior might have in such cases. When interacting with SIP endpoints on the public Internet, TCP connections are often kept alive indefinitely as a means of overcoming Network Address Translation (NAT) limitations in many typical broadband routers and residential gateways.

Converged Application Server does not provide an Application Layer Gateway (ALG) capability, and it is presumed that such capabilities are provided by a standard Session Border Control function.

HTTP User Interface

The 3GPP reference point associated with the HTTP interface provided by Converged Application Server is “Ut". This interface is primarily used for three purposes:

- As a Web-based User Interface for customer self-care and service configuration, potentially using HTML, XHTML or other presentation technologies.
- To support content indirection.
- To support XML Configuration Access Protocol (XCAP), required by Presence and Conference Control Protocol.

Converged Application Server provides HTTP support through its HTTP Servlet Container. Application developers may implement applications or components that support any or all of the above use cases for the “Ut" reference point.

Service and Subscriber Data and Authentication

Converged Application Server supports the Sh reference point used to interact with the Home Subscriber Server (HSS) as the principal provider of IMS Profile data associated with the Public Identity of the network user or subscriber. In many cases, standard LDAP directory servers or relational databases are also used as supplementary resources for service or subscriber data. These may also be accessed via standard interfaces supported by Converged Application Server.

In many deployments, and for certain types of services such as Presence or media repositories, subscriber and service data can be accessed using other means. These include LDAP, HTTP, or access to relational databases.

In non-IMS deployments, the security provider may also be a standard directory accessed via Lightweight Directory access Protocol (LDAP) or access to a relational database using a database-specific interface. Most major commercial relational databases provide Java Database Connectivity (JDBC). A number of high-performance and fault-tolerant JDBC drivers are available commercially for use with Converged Application Server.

Proxy Registrar

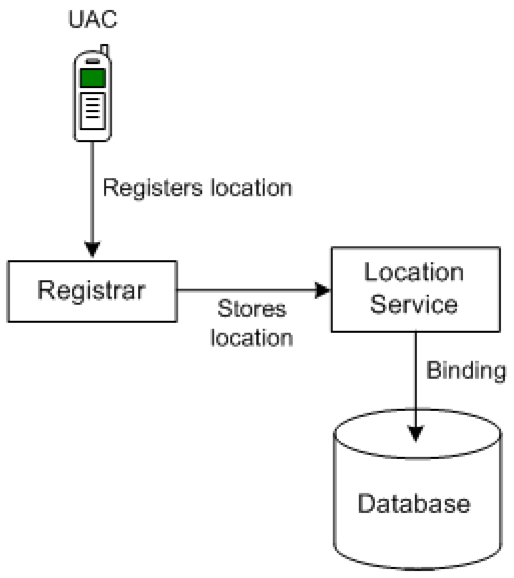
WARNING:

The proxy registrar feature has been deprecated in release 8.0.

The Converged Application Server Proxy Registrar implements the proxy and registrar functions described in RFC 3261. The Proxy Registrar combines the functionality of a SIP proxy server and registrar. Its main tasks include registering subscribers, looking up subscriber locations, and proxying requests onward. The Proxy Registrar is an optional component.

The Proxy Registrar's registrar function processes the REGISTER requests from user agent clients (UACs) and uses a location service to store a binding (that is, an association) between a user's address of record (AOR) and one or more contact addresses, typically the IP addresses of the UACs. The To header field of the REGISTER SIP message sent by a UA contains the address of record whose registration is to be created, queried, or modified and the CONTACT header field contains the corresponding contact addresses. The bindings between the AOR and the contact addresses are persistently stored in a database.

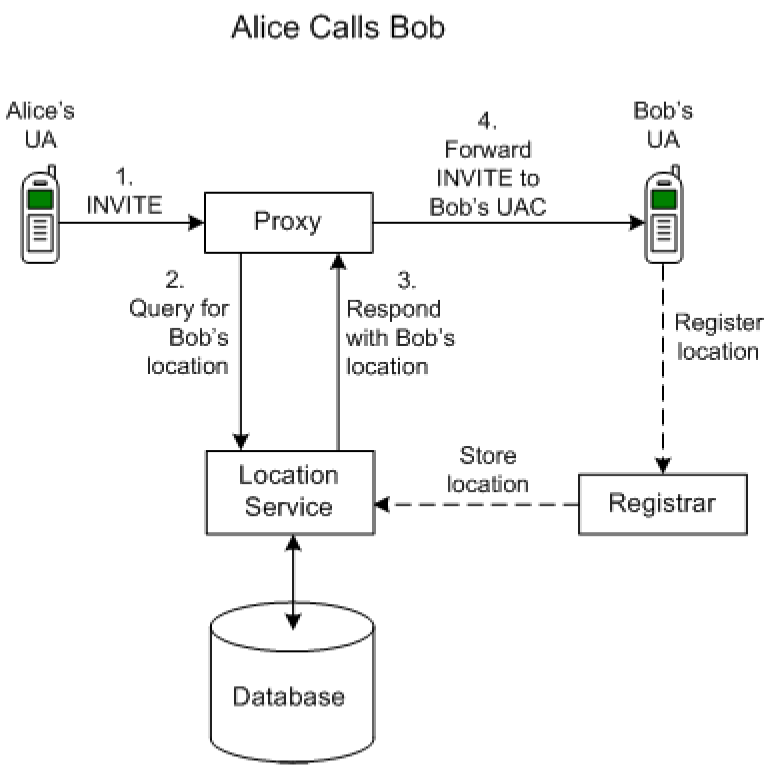
Figure 3-2 Registration Flow



Upon receiving requests to the AOR, the proxy function locates the mapped URIs through a Location Service lookup and then proxies the request using the location information retrieved by this lookup.

Figure 3-3 illustrates a simplified view of the interaction between UAs when a subscriber, Alice, calls another subscriber, Bob, who is located in the same domain.

Figure 3-3 Interaction of UA Elements During a Call

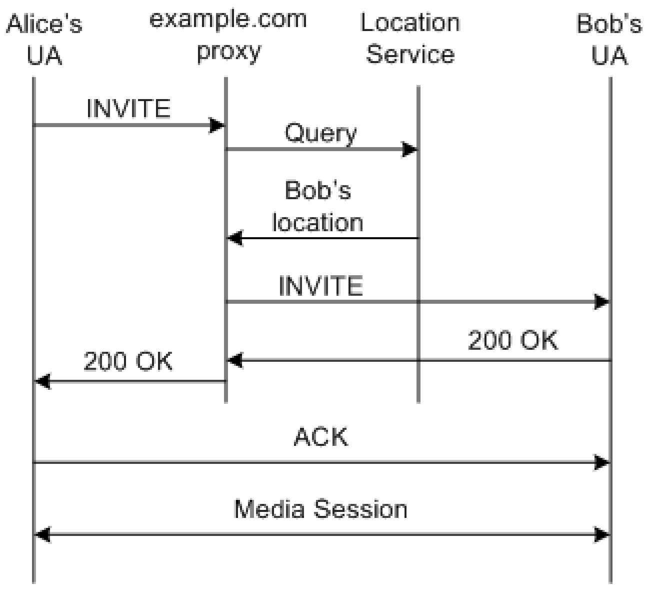


Bob may be registered from multiple user agents, such as personal phone, work phone, and computer. In this case, the query for Bob's location will return multiple bindings to the Proxy. The Proxy will then fork the call, either in parallel or sequentially, to the user agents that Bob is logged in to.

The Proxy is capable of proxying not only INVITE request, but other non-REGISTER requests such as MESSAGE, PUBLISH, SUBSCRIBE and so on.

When a caller and callee are in the same domain, the callee's location can be obtained by the outgoing proxy through the location service. A simplified example of the call flow for this scenario is shown in Figure 3-4. Note that this example does not include `100 Trying` and `180 Ringing` responses.

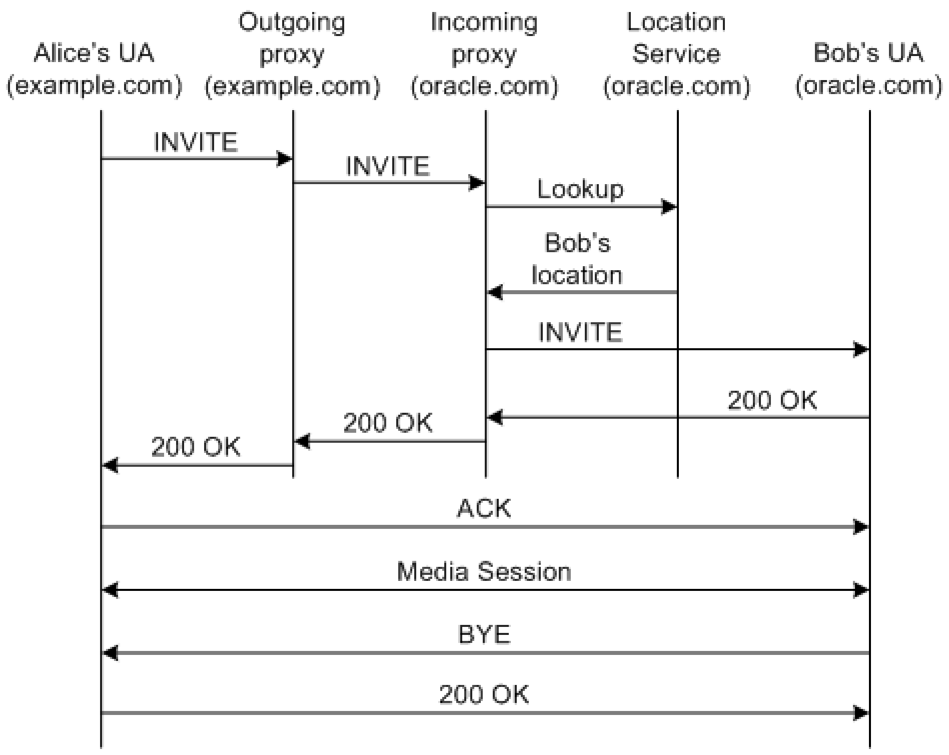
Figure 3-4 Call Establishment Flow Between Subscribers in a Single Domain



After the call is established, Alice and Bob's UAs can communicate directly, without using the Proxy. However, you can configure to route all subsequent traffic through the Proxy as well. This is the default and is useful if you want the ability to add additional services during the session.

If the caller and callee are in different domains, the outgoing proxy forwards the INVITE request to the callee's domain. The incoming proxy in the callee's domain performs the lookup and returns the callee's location, as illustrated in Figure 3-5.

Figure 3-5 Example of a Call Flow Between Two Domains



The Proxy can use ENUM lookup to resolve TEL URLs. The backend for the ENUM service is a DNS, which stores a mapping between TEL URLs and SIP URIs.

You configure the Proxy Registrar primarily through the Administration Console. You configure authentication for the Proxy Registrar by editing the `sip.xml` deployment descriptor packaged in the Proxy Registrar application. You can also edit advanced parameters by using the WebLogic Scripting Tool.

Media Server Control

Converged Application Server enables control of media servers using the Media Server Control API based on JSR309, a standard Java interface. JSR309 (also referred to as JSR 309 and the JSR 309 API) defines an abstract Java interface for the manipulation of audio and video streams and conferences. Vendors of IP media servers provide JSR 309 based driver implementations that work with their IP media servers.

The JSR309 architecture assumes a distributed or IMS-like model where the Converged Application Server and media server reside on separate physical servers. User Agents (such as soft phones) interact with the applications deployed on Converged Application Server using the SIP protocol.

Converged Application Server supports media server control by providing built-in JSR309 support. The support includes the Media Server Control API, which provides a standard API for developing and deploying media rich, JSR-based applications for the Java platform without having prior knowledge of the underlying Media Server Control protocols.

A Java application that uses the Media Server Control API can use any JSR309-based implementation with any compatible media server. However, Converged Application Server provides a built-in JSR309 media driver, JVoiceBridge.

Whether using JVoiceBridge or the driver for another media server, the CAFE and Media Server Control APIs offer interfaces that ease the task of developing media-rich applications, such as Conferencing, Ring-back tone, or IVR applications easily using the JSR309 API.

For developers, the Media Server Control API provides a standard API for developing and deploying media rich, JSR-based applications for the Java platform without having prior knowledge of the underlying Media Server Control protocols. Moreover, a Java application that uses the Media Server Control API can use any JSR309-based implementation with any compatible media server.

For more information, see *JSR 309: Media Server Control API*, `https://jcp.org/en/jsr/detail?id=309`.

Charging and Billing

Converged Application Server provides both a Diameter Rf interface application and a Diameter Ro interface application to facilitate offline and online charging in IMS networks. See "Using the Diameter Rf Interface Application for Offline Charging" and "Using the Diameter Ro Interface Application for Online Charging" in *Converged Application Server Diameter Application Development Guide* for information about how to access and use these Diameter applications in your own SIP Servlets.

Security

Converged Application Server users must be authenticated when they request access to a protected resource, such as a protected method in a deployed SIP Servlet. Converged Application Server enables you to perform SIP Servlet authentication using any of the following techniques:

- **DIGEST authentication** uses a simple challenge-response mechanism to verify the identity of a user over SIP or HTTP. See "Configuring Digest Authentication" in *Converged Application Server Administrator's Guide*.
- **CLIENT-CERT authentication** uses an X509 certificate chain passed to the SIP application to authenticate a user. The X509 certificate chain can be provided in a number of different ways. In the most common case, two-way SSL handshake is performed before transmitting the chain to ensure secure communication between the client and server.
- **BASIC authentication** uses the Authorization SIP header to transmit the username and password to SIP Servlets. BASIC authentication is not recommended for production systems unless you can ensure that all connections between clients and the Converged Application Server instance are secure.

Different SIP Servlets deployed on Converged Application Server can use different authentication mechanisms as necessary. The required authentication mechanism is specified in the auth-method element of the SIP Servlet Application's deployment descriptor. The deployment descriptor may also define resources that are to be protected, listing the specific role names that are required for access.

Authentication Providers

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

Converged Application Server may be configured to use multiple authentication providers via different authentication methods. For example, when using Digest authentication an administrator may configure both a Digest Identity Asserter provider to assert the validity of a digest, and a second LDAP or RDBMS authentication provider that determines the group membership of a validated user.

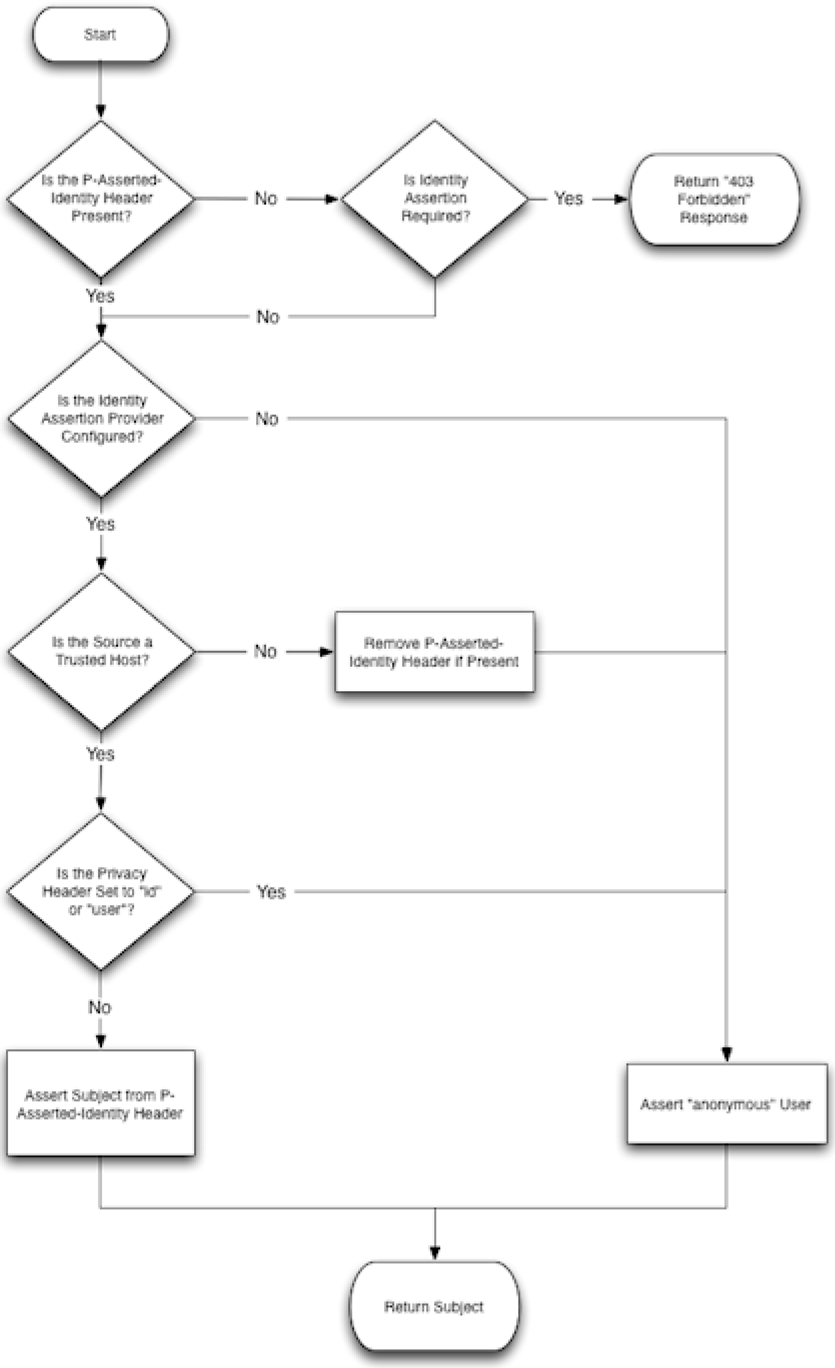
Trusted Host Authentication

Converged Application Server is designed for deployment scenarios where it is adjacent to trusted hosts and it is not required to fulfill the role of an application layer security boundary between the trusted and untrusted domains.

Converged Application Server enables administrators to designate network hosts that are considered to be “trusted." Trusted hosts are hosts for which Converged Application Server performs no authentication. If the server receives a SIP message having a destination address that matches a configured trusted hostname, the message is delivered without Authentication.

Converged Application Server supports the P-Asserted-Identity SIP header as described in IETF RFC 3325: Private Extensions to the Session Initiation Protocol (SIP) for Asserted Identity within Trusted Networks. This functionality automatically logs in using credentials specified in the P-Asserted-Identity header when they are received from a trusted host. When combined with the privacy header, P-Asserted-Identity also determines whether the message can be forwarded to trusted and non-trusted hosts.

Figure 3-6 Asserted Identity Handling in Converged Application Server


Start
Is the P-Asserted-Identity Header Present?
No
Is Identity Assertion Required?
Yes
Return "403 Forbidden" Response
Yes
No
Is the Identity Assertion Provider Configured?
No
Yes
Is the Source a Trusted Host?
No
Remove P-Asserted-Identity Header if Present
Yes
Is the Privacy Header Set to "id" or "user"?
Yes
No
Assert Subject from P-Asserted-Identity Header
Assert "anonymous" User
Return Subject

It is also possible to use Converged Application Server in scenarios that do not involve trusted hosts. See "Standards Alignment" for a more detailed description of Converged Application Server standards compliance.

Declarative Security

The SIP Servlet API specification defines a set of deployment descriptor elements that can be used for providing declarative and programmatic security for SIP Servlets. The primary method for declaring security constraints is to define one or more security-constraint elements and role definitions in the sip.xml deployment descriptor. Converged Application Server adds additional deployment descriptor elements to help developers easily map SIP Servlet roles to actual principals and/or roles configured by the Converged Application Server administrator.

Protecting the Converged Application Server Domain with a Session Border Controller

A Session Border Controller (SBC) is a device used in VoIP networks to exert control over the signaling (and usually also the media streams) involved in setting up, conducting, and tearing down interactive media communications. SBCs are typically used to secure and protect the network and other devices in the operator's network from denial of service (DOS) attacks. Besides security, SBCs also perform functions such as QoS guarantees, regulatory compliances (lawful intercept), statistics collection, and so on. Services developed and deployed on Converged Application Server are most commonly hosted inside trusted networks. It is recommended to protect the network which hosts such services deployed on Converged Application Server with a Session Border Controller.

4

Converged Application Server Cluster Architecture

This chapter describes the Oracle Communications Converged Application Server cluster architecture.

Overview of Converged Application Server Clusters

A Converged Application Server cluster consists of multiple Converged Application Server server instances running simultaneously and working together to provide increased scalability and reliability. A cluster appears to clients to be a single Converged Application Server instance. The server instances that constitute a cluster can run on the same machine, or be located on different machines. You can increase a cluster's capacity by adding additional server instances to the cluster on an existing machine, or you can add machines to the cluster to host the incremental server instances. Each server instance in a cluster must run the same version of Converged Application Server.

Relationship Between Clusters and Domains

A cluster is part of a particular Converged Application Server domain.

A domain is an interrelated set of Converged Application Server resources that are managed as a unit. A domain includes one or more Converged Application Server instances, which can be clustered, non-clustered, or a combination of clustered and non-clustered instances. A domain can include multiple clusters. A domain also contains the application components deployed in the domain, and the resources and services required by those application components and the server instances in the domain. Examples of the resources and services used by applications and server instances include machine definitions, optional network channels, connectors, and startup classes.

You can use a variety of criteria for organizing Converged Application Server instances into domains. For instance, you might choose to allocate resources to multiple domains based on logical divisions of the hosted application, geographical considerations, or the number or complexity of the resources under management. For additional information about domains see Understanding Domain Configuration for Oracle WebLogic Server.

In each domain, one Converged Application Server instance acts as the Administration Server: the server instance which configures, manages, and monitors all other server instances and resources in the domain. Each Administration Server manages one domain only. If a domain contains multiple clusters, each cluster in the domain has the same Administration Server. All server instances in a cluster must reside in the same domain; you cannot "split" a cluster over multiple domains. Similarly, you cannot share a configured resource or subsystem between domains.

Relationship Between Coherence and WebLogic Server Clusters

Coherence clusters consist of multiple managed Coherence server instances that work together to distribute data in-memory to increase application scalability, availability, and performance. A client interacts with the data in a local cache and the distribution and backup of the data is automatically performed across cluster members.

Coherence clusters are different than Converged Application Server clusters. They use different clustering protocols and are configured separately. A Converged Application Server domain can contain a single Coherence cluster. Multiple Converged Application Server clusters can be associated with a Coherence cluster.

For details on configuring and managing Coherence clusters, see Administering Clusters for Oracle WebLogic Server.

Objects That Can Be Clustered

A clustered application or application component is one that is available on multiple Converged Application Server instances in a cluster. If an object is clustered, failover and load balancing for that object is available. Deploy objects homogeneously—to every server instance in your cluster—to simplify cluster administration, maintenance, and troubleshooting.

Web applications can consist of different types of objects, including Enterprise Java Beans (EJBs), servlets, and Java Server Pages (JSPs). Each object type has a unique set of behaviors related to control, invocation, and how it functions within an application. For this reason, the methods that Converged Application Server uses to support clustering—and hence to provide load balancing and failover—can vary for different types of objects. The following types of objects can be clustered in a Converged Application Server deployment:

- Servlets
- JSPs
- EJBs
- Remote Method Invocation (RMI) objects
- Java Messaging Service (JMS) destinations
- Coherence cluster and managed Coherence servers
- Timer services

Objects That Cannot Be Clustered

The following APIs and internal services cannot be clustered in Converged Application Server:

- File services including file shares

Overview of the Cluster Architecture

Converged Application Server itself provides a multi-tier cluster architecture using Oracle Coherence. Coherence clusters consist of multiple managed Coherence server instances that distribute data in-memory to increase application scalability, availability, and performance. An application interacts with the data in a local cache and the distribution and backup of the data is automatically performed across cluster members.

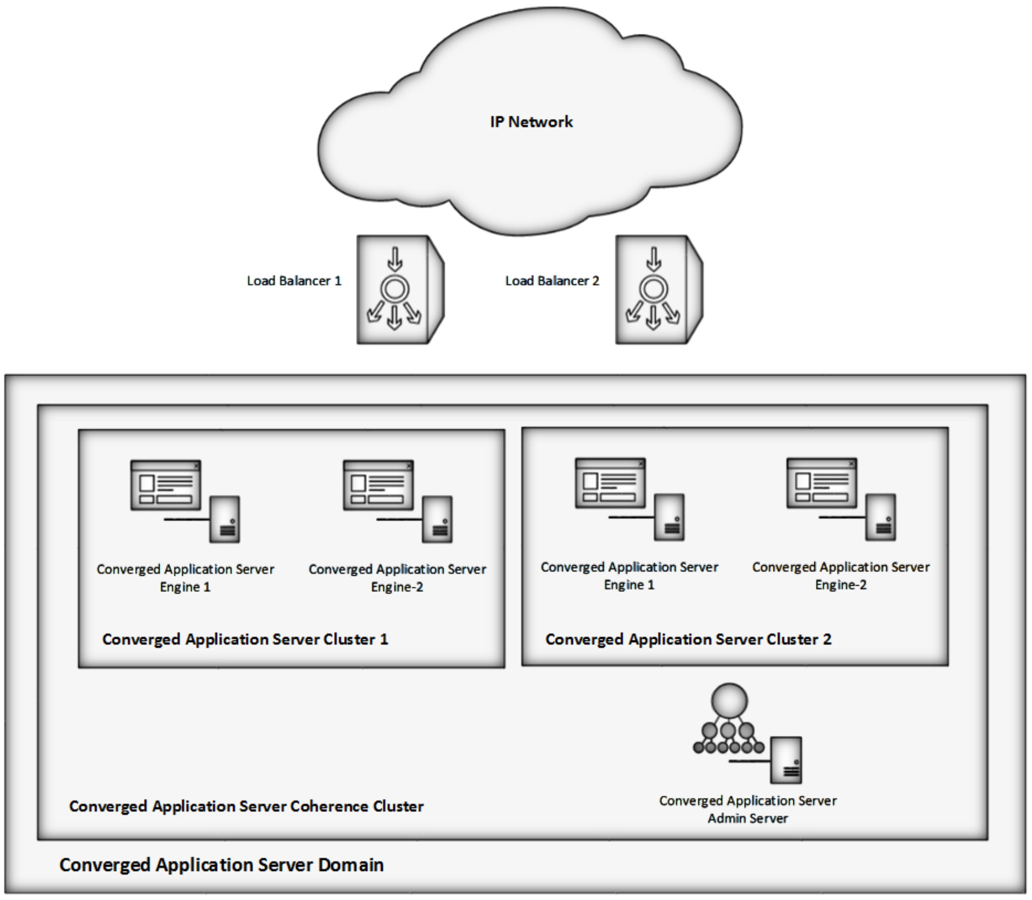
Coherence integration aligns the lifecycle of a Coherence cluster member with the lifecycle of a managed server: starting or stopping a server Java Virtual Machine (JVM) starts and stops a Coherence cluster member. The first member of the cluster starts the cluster service and is the senior member.

Converged Application managed servers that are associated with a Coherence cluster are referred to as managed Coherence servers. Managed Coherence servers in each tier can be individually managed but are typically associated with respective Converged Application Server clusters.

A standard load balancing appliance is used to distribute traffic across the engines in the cluster. It is not necessary that the load balancer be SIP-aware; there is no requirement that the load balancer support affinity between Engines and SIP dialogs or transactions. However, SIP-aware load balancers can provide higher performance by maintaining a client's affinity to a particular engine.

Figure 4-1 shows an example Converged Application Server cluster in which traffic from the IP Network is routed through two load balancers, Load Balancer 1 and Load Balancer 2, which distribute requests to the four Converged Application Server engines in the clusters, Cluster 1 and Cluster 2. Within the same Coherence cluster a single Administration Server handles administration tasks for all of the Converged Application Server engines.

Figure 4-1 Example Converged Application Cluster


IP Network
Load Balancer 1
Load Balancer 2
Converged Application Server Engine 1
Converged Application Server Engine-2
Converged Application Server Cluster 1
Converged Application Server Engine 1
Converged Application Server Engine-2
Converged Application Server Cluster 2
Converged Application Server Coherence Cluster
Converged Application Server Admin Server
Converged Application Server Domain

> **Note:**
>
> There is no arbitrary limit to the number of engines or physical servers within a Coherence cluster.

Geographically-Redundant Installations

Converged Application Server can be installed in a geographically-redundant configuration for implementations that employ distributed data centers, and require continuing operation even after a catastrophic site failure.

The geographically-redundant configuration enables multiple Converged Application Server installations to replicate call state transactions between one another. If a particular site's installation were to suffer a critical failure, the administrator could choose to redirect all network traffic to the secondary, replicated site to minimize lost calls.

For information on configuring geographical redundancy, see "Configuring Geographically-Redundant Installations" in *Converged Application Server Administrator's Guide*.

Administration Server

You manage a Converged Application Server domain using an Administration Server. The Administration Server hosts the Administration Console interface, which you use to configure, deploy, and monitor the Converged Application Server installation.

Oracle recommends the following best practices for configuring Administration Server and Managed Server instances in your Converged Application 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 central processing unit (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 unavailable. For more information, see Administering Server Startup and Shutdown for Oracle WebLogic Server for more information.
- Configure the Node Manager utility to automatically restart all Managed Servers in the Converged Application Server domain. See Understanding the WebLogic Scripting Tool for more information.

If an Administration Server fails, only configuration, deployment, and monitoring features are affected, but Managed Servers continue to operate and process client requests. See Monitoring, Tuning, and Troubleshooting in *Oracle Communications Converged Application Server Administrator's Guide*.

Engines

Converged Application Server engines reside in clusters and host the SIP Servlets and other applications that provide features to SIP clients.

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

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

Note:

Converged Application Server start scripts use default values for many JVM parameters that affect performance. For example, JVM garbage collection and heap size parameters may be omitted, or may use values that are acceptable only for evaluation or development purposes. In a production system, you must rigorously profile your applications with different heap size and garbage collection settings in order to realize adequate performance. See "Monitoring, Tuning, and Troubleshooting the JVM" in *Converged Application Administrator's Guide* for suggestions about maximizing JVM performance in a production domain.

Diameter Support

Converged Application Server supports the Diameter base protocol. It supports the IMS Sh interface provider on engines, which then act as Diameter Sh client nodes. SIP Servlets deployed on the engines can use the profile service API to initiate requests for user profile data, or to subscribe to and receive notification of profile data changes. The Sh interface is also used to communicate between multiple IP Multimedia Subsystem (IMS) Application Servers.

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

The relay agents included in Converged Application Server perform only stateless proxying of Diameter messages; messages are not cached or otherwise processed before delivery to the HSS.

Note:

In order to support multiple HSSs, the 3rd Generation Partnership Project (3GPP) defines the Dh interface to look up the correct HSS. Converged Application Server does not provide a Dh interface application, and can be configured only with a single HSS.

Note that relay agent servers do not function as engines: they should not host applications, store call state data, maintain SIP timers, or even use SIP protocol network resources (sip or sips network channels).

In summary, Converged Application Server supports the following Diameter functions:

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

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

For information on developing Diameter applications for Converged Application Server, see the Converged Application Server Diameter Application Development Guide.

5

Deployment Scenarios

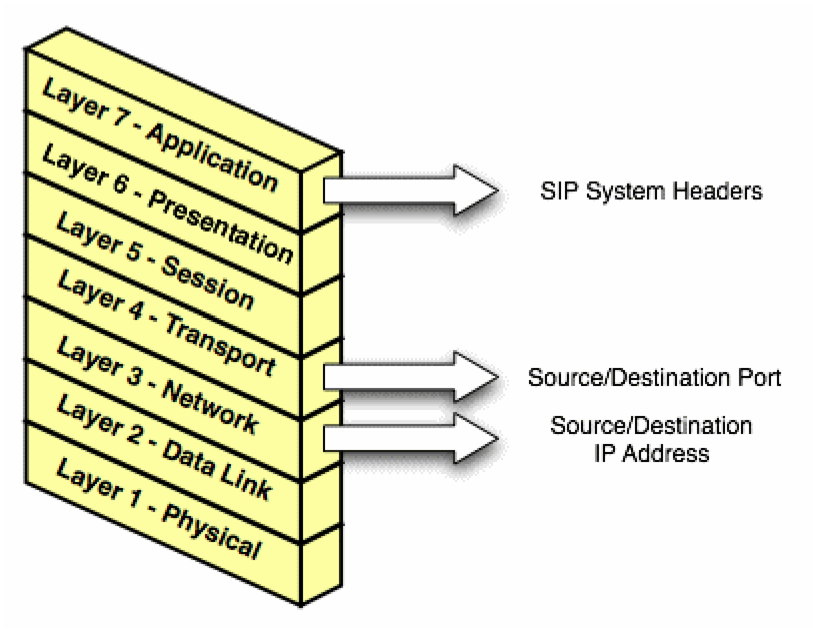
This chapter describes the Oracle Communications Converged Application Server deployment considerations architecture.

Overview of Deployment Scenarios

This section describes common Converged Application Server network architectures and network configuration considerations for each architecture, particularly in relation to the Open Systems Interconnect (OSI) model. See *Converged Application Server Administrator's Guide* for detailed configuration steps described in this section.

Figure 5-1 shows the OSI model layers that are typically affected by the network configuration requirements for the Converged Application Server deployment.

Figure 5-1   OSI Layers Relevant to Converged Application Server Configuration



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

Single-NIC Configurations with TCP and UDP Channels

In a simple network configuration for a server having a single network interface controller (NIC), one or more network channels may be created to support SIP messages over User Datagram Protocol (UDP) and Transmission Control Protocol (TCP), or Session Initiation

Protocol (SIP) over Transport Layer Security (TLS). It is helpful to understand how this simple configuration affects information in the OSI model, so that you can understand how more complex configurations involving multihomed hardware and load balancers affect the same information.

Figure 5-2 Single-NIC Network Channel

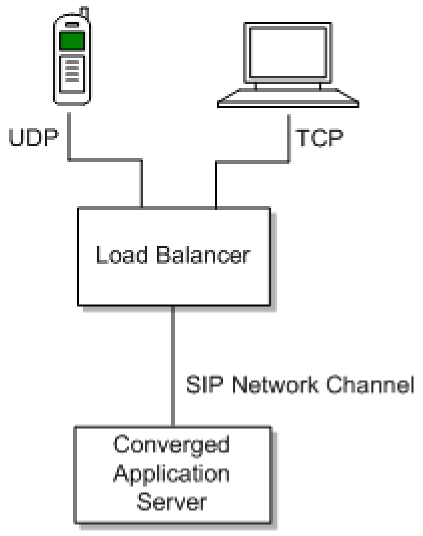


Figure 5-2 shows a single engine tier server instance with a single NIC. The server is configured with one network channel supporting SIP over UDP and TCP. (SIP channels always support both UDP and TCP transports; you cannot support only one of the two.) The scenario also shows two clients communicating with the server, one over UDP and one over TCP.

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

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

Incoming TCP datagrams (delivered from the UA to Converged Application Server) contain the following information:

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

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

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

By default Converged Application Server allocates ports from the ephemeral port number range of the underlying operating system for outgoing UDP datagrams. Converged Application Server allows external connections to use an ephemeral point as the destination port number, in addition to the port number configured in the network channel. In other words, Converged Application Server automatically listens on all ephemeral ports that the server allocates. You can optionally disable Converged Application Server's use of ephemeral port numbers and setting a static port, as described in *Oracle Communications Converged Application Server Administrator's Guide*.

Multihomed Server Configurations Overview

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

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

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

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

Multihomed Servers Listening On All Addresses (IP_ANY)

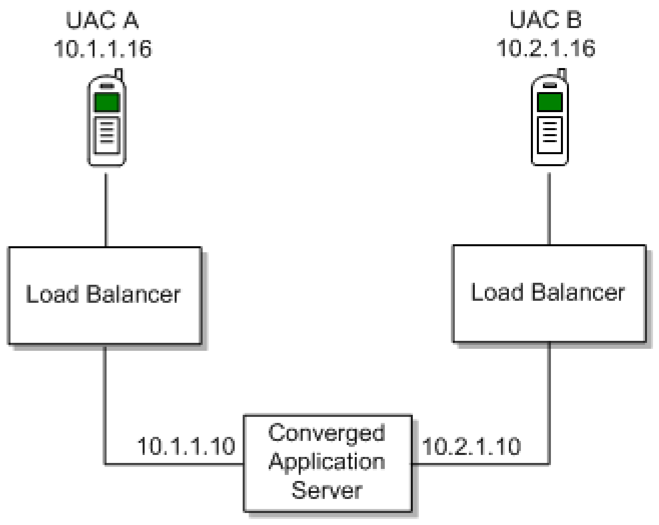
The simplest multihome configuration enables a Converged Application Server instance to listen on all available NICs (physical NICs as well as logical NICs), sometimes described as IP_ANY. To accomplish this, you configure a single network channel and specify a channel listen address of 0.0.0.0.

See information about configuring engine servers to listen on any IP interface in the *Converged Application Server Administrator's Guide*.

Multihomed Servers Listening on Multiple Subnets

Multiple NICs can also be used in engine tier servers to listen on multiple subnets. The most common configuration uses Converged Application Server to proxy SIP traffic from one subnet to another where no direct access between subnets is permitted. Figure 5-3 shows this configuration.

Figure 5-3 Multihomed Configuration for Proxying between Subnets



To configure the Converged Application Server instance in this scenario, you must define a separate network channel for each NIC used on the server machine. Example 5-1 shows the `config.xml` entries that define channels for the sample configuration.

Example 5-1 Sample Network Channel Configuration for NICs on Multiple Subnets

```
<NetworkAccessPoint ListenAddress="10.1.1.10" ListenPort="5060"
Name="sipchannelA" Protocol="sip"/>
<NetworkAccessPoint ListenAddress="10.2.1.10" ListenPort="5060"
Name="sipchannelB" Protocol="sip"/>
```

Understanding the Route Resolver

When Converged Application Server is configured to listen on multiple subnets, a feature called the *route resolver* is responsible for the following activities:

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

For example, in the configuration shown in Figure 5-3, Converged Application Server must add the correct subnet address to SIP system headers and transport datagrams in order for each UA to continue processing SIP transactions. If the wrong subnet is used, replies cannot be delivered because neither UA can directly access the other UA's subnet.

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

For example, in the configuration shown in Figure 5-3, an INVITE message sent from Converged Application Server to UAC B would have a destination address of

10.2.1.16. The operating system would transmit this message using NIC B, which is configured for the corresponding subnet. The route resolver associates NIC B with the configured `sipchannelB` and embeds the channel's IP address (10.2.1.10) in the VIA header of the SIP message. UAC B then uses the Via header to direct subsequent messages to the server using the correct IP address. A similar process is used for UAC A, to ensure that messages are delivered only on the correct subnet.

IP Aliasing with Multihomed Hardware

IP aliasing assigns multiple logical IP addresses to a single NIC, and is configured in the underlying server operating system. If you configure IP aliasing and all logical IP addresses are within the same subnet, you configure Converged Application Server to listen on all addresses.

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

Load Distribution Considerations

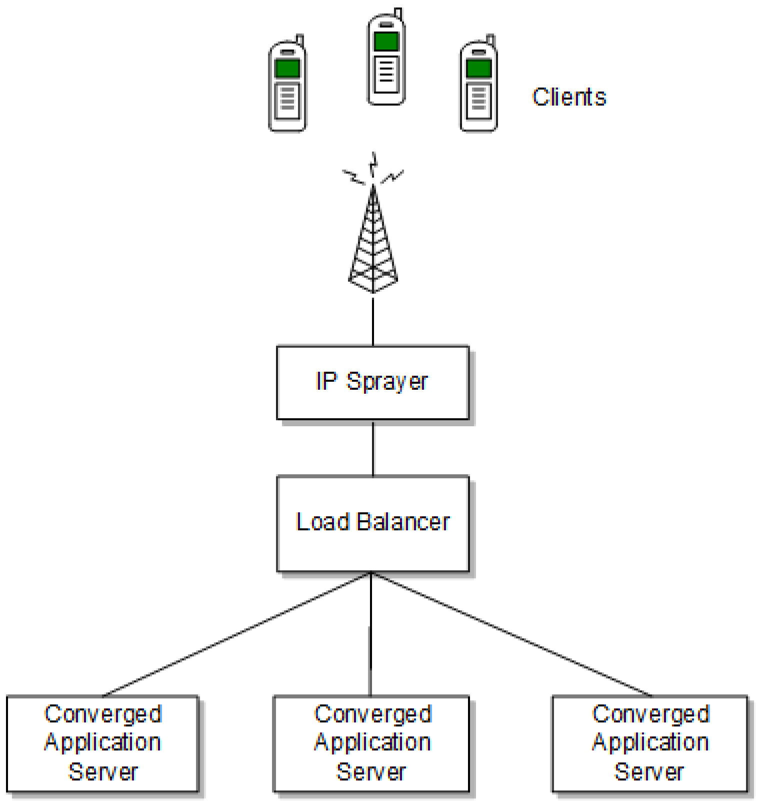
A load balancer improves the reliability and scalability of your Converged Application Server deployment. A load balancer distributes requests among the servers in your deployment and monitors their availability.

The following sections describe considerations related to exposing an external virtual IP (VIP) for the Converged Application Server deployment. These sections assume the use of an IP sprayer. However, your deployment may use other technology to perform the network dispatching function, such as the DNS Resource Records (RR) feature in the Linux operating system.

Single VIP Topology

In the simplest scenario, a single IP sprayer gates access to a Load Balancer which distributes messages to a cluster of engines, as shown in Figure 5-4.

Figure 5-4 SIngle Load Balancer Configuration


Clients
IP Sprayer
Load Balancer
Converged Application Server
Converged Application Server
Converged Application Server

To configure Converged Application Server for use with a single IP sprayer, configure one or more network channels for each server, and configure the external address setting of each channel with the virtual IP address of the IP sprayer. The virtual IP address is the address exposed for the Converged Application Server installation to external clients.

In this configuration, Converged Application Server embeds the external address (or VIP) in SIP message system headers to ensure that clients can reach the cluster for subsequent replies.

Multiple VIP Topology

Multiple IP sprayers (or a multihomed node that functions as an IP sprayer) can be configured to provide several virtual IP addresses for a single Converged Application Server cluster.

To configure Converged Application Server for multiple VIPs, create a dedicated network channel for each IP sprayer or local server NIC. You then set the channel's external address to the virtual IP address of the IP sprayer to which the channel connects.

In this configuration, the route resolver associates a configured channel with the NIC used for originating SIP messages. The public address of the selected channel is then used for populating SIP system messages. See "Understanding the Route Resolver".

Network Address Translation Options

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

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

IP Masquerading Alternative to Source NAT

If you choose not to enable source NAT on your load balancer, Converged Application Server provides limited IP masquerading functionality. To use this functionality, configure a logical address on each engine tier server using the public IP address of the load balancer for the cluster. (Duplicate the same logical IP address on each engine tier server machine). When a local server interface matches the IP address of a configured load balancer (defined in the public address of a network channel), Converged Application Server uses that interface to originate SIP UDP messages, and the Layer 3 address contains a public address.

Caution:

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

You can disable IP masquerading functionality by using the startup option:

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

See the *Converged Application Server Administrator's Guide* for more information on startup options.

Example Scenarios

This section describes Converged Application Server deployment scenarios. In particular, it describes considerations related to SIP, load balancer, and Network Address Translation (NAT), and shows the message flows in such scenarios.

Note:

For more information about implementation-dependent issues surrounding NAT, see the Internet Engineering Task Force (IETF) document *NAT Behavioral Requirements for Unicast UDP* at the IETF website:

`http://www.ietf.org/rfc/rfc4787.txt`

When deployed with multiple SIP-aware load balancers, the Converged Application Server deployment also typically includes an IP sprayer or network-level load balancer to perform IP-level message distribution.

You can also deploy the Converged Application Server with load balancers that are not SIP-aware, meaning that they do not consider existing SIP dialogues when routing requests to servers. The following sections describe the scenario given a non-SIP aware load balancer.

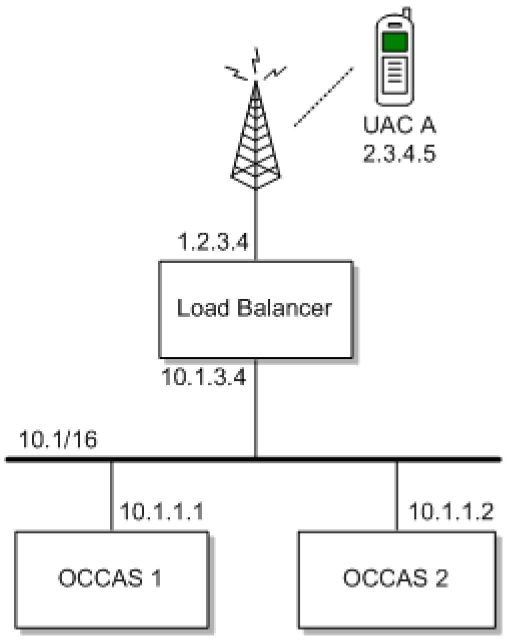
Example Deployment with a Non-SIP Aware Load Balancer

Figure 5-5 shows the sample network topology described in this section. A Converged Application Server cluster, consisting of engines WLSS 1 and WLSS 2, is configured on private IP network 10.1/16 (an internal 16-bit subnet). The cluster's public IP address is 1.2.3.4, which is the virtual IP address configured on the load balancer.

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

The sections that follow discuss configuring the Converged Application Server cluster and load balancer for the example system.

Figure 5-5 Example Network Topology



Converged Application Server Configuration

The Converged Application Server cluster configuration specifies the public address as 1.2.3.4, and the public port as 5060 for each engine. The default route on both Converged Application Server engines points to the load balancer's 10.1/16 network interface: 10.1.3.4. The Converged Application Server (servers WLSS 1 and WLSS 2) routing table is shown in Example 5-2.

Example 5-2   Converged Application Server Routing Table

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

Load Balancer Configuration

The load balancer is configured with a virtual IP address of 1.2.3.4, and two real servers, WLSS 1 and WLSS 2, having addresses 10.1.1.1 and 10.1.1.2, respectively. The load balancer also has an internal IP address of 10.1.3.4 configured on the 10.1/16 network. The UAC address, 2.3.4.5, is reachable from the load balancer by static route configuration on the load balancer. The load balancer routing table is shown in Example 5-3.

Example 5-3   Load Balancer Routing Table

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

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

- NAT-based Configuration
- maddr-based Configuration
- rport-based Configuration

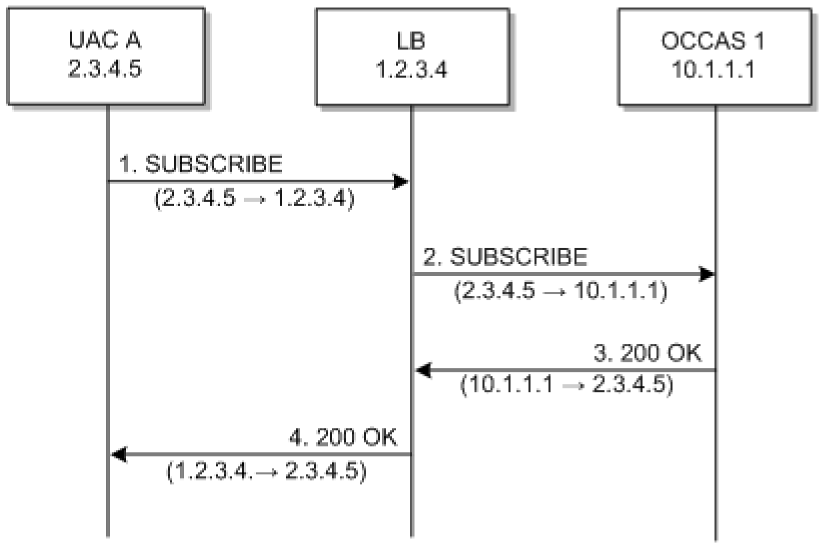
The sections that follow describe each approach.

NAT-based Configuration

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

Figure 5-6 illustrates the UDP packet flow for a SUBSCRIBE/200 OK transaction.

Figure 5-6 SUBSCRIBE Sequence



In the figure, the source and destination IP addresses of the UDP packets are shown under the message path arrow. In the UAC-to-Converged Application Server direction, the load balancer translates the destination IP address but not the source IP address. In the Converged Application Server-to-UAC direction, the load balancer translates the source IP address but not the destination IP address.

Example 5-4 shows the complete message trace (including IP and UDP headers, as well as the SIP payload) for the sequence from Figure 5-6.

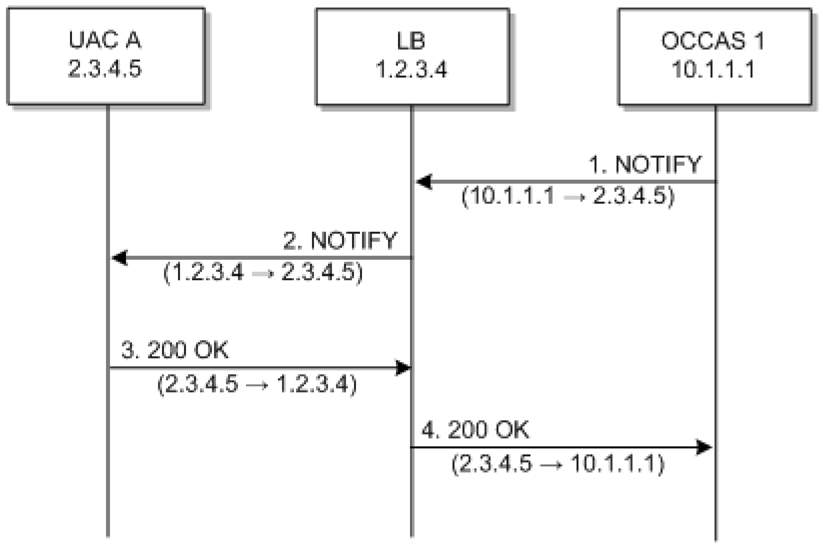
Example 5-4 Complete SUBSCRIBE Message Trace

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

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

No.     Time         Source                 Destination
Protocol Info
```

```
      2 2.426250    2.3.4.5           10.1.1.1          SIP      Request:
SUBSCRIBE sip:subscribe@1.2.3.4:5060

Internet Protocol, Src: 2.3.4.5 (2.3.4.5), Dst: 10.1.1.1 (10.1.1.1)
User Datagram Protocol, Src Port: 9999 (9999), Dst Port: sip (5060)
Session Initiation Protocol
    Request-Line: SUBSCRIBE sip:subscribe@1.2.3.4:5060 SIP/2.0
    Message Header
        Via: SIP/2.0/UDP 2.3.4.5:9999;branch=1
        From: sipp <sip:sipp@2.3.4.5>;tag=1
        To: sut <sip:subscribe@1.2.3.4:5060>
        Call-ID: 1-25923@2.3.4.5
        Cseq: 1 SUBSCRIBE
        Contact: sip:sipp@2.3.4.5:9999
        Max-Forwards: 70
        Event: ua-profile
        Expires: 10
        Content-Length: 0

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

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

Figure 5-7 shows the message sequence that results from the Converged Application Server subsequently sending a NOTIFY request to the UAC:

Figure 5-7 NOTIFY Sequence



As in the previous sequence, the IP address translation takes place in the Converged Application Server to UAC direction for the source IP address, and UAC to Converged Application Server direction for the destination IP address.

Note that this setup does not require the load balancer to maintain session state information or to be SIP-aware. The complete message trace from is shown in Example 5-5 below.

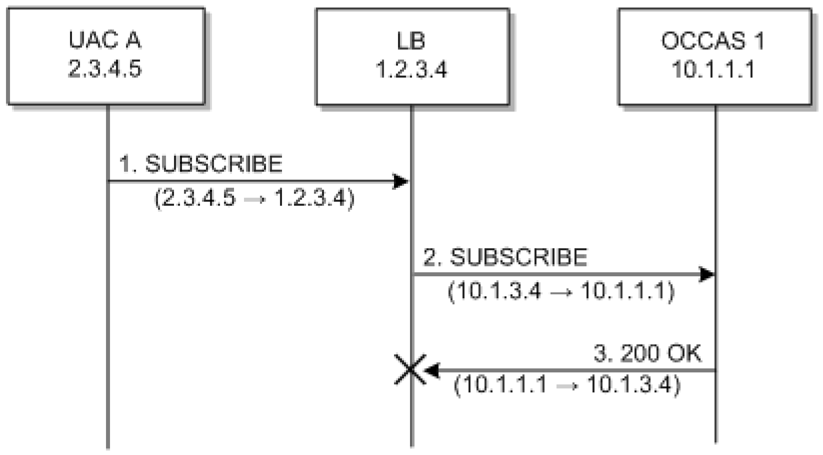
Example 5-5 Complete NOTIFY Message Trace

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

Internet Protocol, Src: 10.1.1.1 (10.1.1.1), Dst: 2.3.4.5 (2.3.4.5)
User Datagram Protocol, Src Port: 42316 (42316), Dst Port: 9999 (9999)
Session Initiation Protocol
    Request-Line: NOTIFY sip:sipp@2.3.4.5:9999 SIP/2.0
    Message Header
        To: sipp <sip:sipp@2.3.4.5>;tag=1
        Content-Length: 0
        Contact:
<sip:app-12eomtm5h5f77@1.2.3.4:5060;transport=udp;wlsscid=1ae4479ac6ff7
1>
        CSeq: 1 NOTIFY
        Call-ID: 1-25923@2.3.4.5
        Via: SIP/2.0/UDP
1.2.3.4:5060;wlsscid=1ae4479ac6ff71;branch=z9hG4bKc5e4c3b4c22be517133ab
749adeece4e
        From: sut <sip:subscribe@1.2.3.4:5060>;tag=82722c03
        Max-Forwards: 70

No.     Time        Source                Destination
```

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

Internet Protocol, Src: 1.2.3.4 (1.2.3.4), Dst: 2.3.4.5 (2.3.4.5)
User Datagram Protocol, Src Port: 2222 (2222), Dst Port: 9999 (9999)
Session Initiation Protocol
    Request-Line: NOTIFY sip:sipp@2.3.4.5:9999 SIP/2.0
    Message Header
        To: sipp <sip:sipp@2.3.4.5>;tag=1
        Content-Length: 0
        Contact:
<sip:app-12eomtm5h5f77@1.2.3.4:5060;transport=udp;wlsscid=1ae4479ac6ff71>
        CSeq: 1 NOTIFY
        Call-ID: 1-25923@2.3.4.5
        Via: SIP/2.0/UDP
1.2.3.4:5060;wlsscid=1ae4479ac6ff71;branch=z9hG4bKc5e4c3b4c22be517133ab749ade
ece4e
        From: sut <sip:subscribe@1.2.3.4:5060>;tag=82722c03
        Max-Forwards: 70

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

Internet Protocol, Src: 2.3.4.5 (2.3.4.5), Dst: 1.2.3.4 (1.2.3.4)
User Datagram Protocol, Src Port: 9999 (9999), Dst Port: sip (5060)
Session Initiation Protocol
    Status-Line: SIP/2.0 200 OK
    Message Header
        Via: SIP/2.0/UDP
1.2.3.4:5060;wlsscid=1ae4479ac6ff71;branch=z9hG4bKc5e4c3b4c22be517133ab749ade
ece4e
        From: sut <sip:subscribe@1.2.3.4:5060>;tag=82722c03
        To: sipp <sip:sipp@2.3.4.5>;tag=1;tag=1
        Call-ID: 1-25923@2.3.4.5
        CSeq: 1 NOTIFY
        Contact: <sip:2.3.4.5:9999;transport=UDP>
```

If NAT is performed on both the source (SNAT) and destination IP addresses, the configuration does not work because the load balancer usually relies on a specific destination port number value to be sent in responses to requests. That port number value is dictated by RFC 3261, and must come from the Via header, which presents a conflict with load balancer's NAT requirements. RFC 3261 requires that responses to SIP requests be sent to the IP address used to send the request (unless maddr is present in the Via, as described in maddr-based Configuration). Consequently, in step 3 in Figure 5-8 below, Converged Application Server sends a 200 OK response to the load balancer internal IP address (10.1.3.4) and port 5060. That response is then dropped.

Figure 5-8   Source and Destination NAT



Example 5-6 shows the complete message trace.

Example 5-6   Complete Failing SUBSCRIBE Message Trace

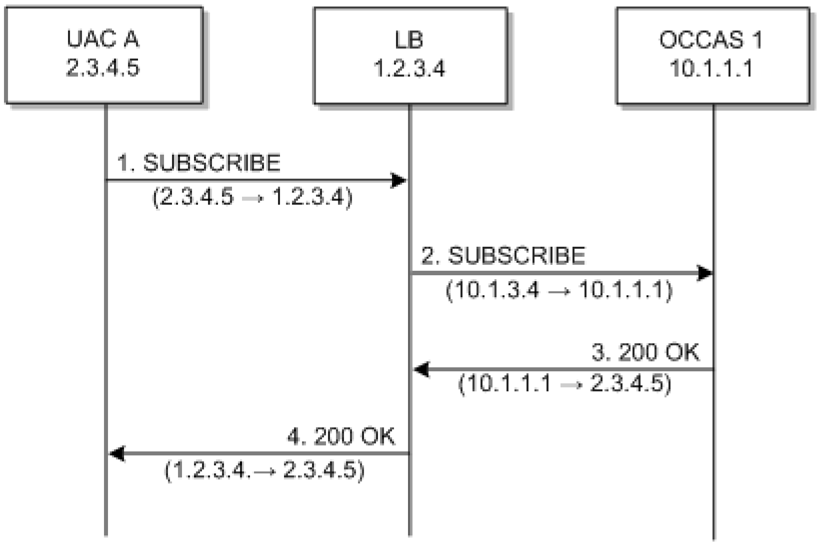
```
No.     Time        Source                Destination
Protocol Info
      1 1.425250    2.3.4.5            1.2.3.4          SIP
Request: SUBSCRIBE sip:subscribe@1.2.3.4:5060

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

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

Internet Protocol, Src: 10.1.3.4 (10.1.3.4), Dst: 10.1.1.1 (10.1.1.1)
User Datagram Protocol, Src Port: 2222 (2222), Dst Port: sip (5060)
Session Initiation Protocol
    Request-Line: SUBSCRIBE sip:subscribe@1.2.3.4:5060 SIP/2.0
    Message Header
        Via: SIP/2.0/UDP 2.3.4.5:9999;branch=1
        From: sipp <sip:sipp@2.3.4.5>;tag=1
        To: sut <sip:subscribe@1.2.3.4:5060>
        Call-ID: 1-25923@2.3.4.5
        Cseq: 1 SUBSCRIBE
        Contact: sip:sipp@2.3.4.5:9999
        Max-Forwards: 70
        Event: ua-profile
        Expires: 10
        Content-Length: 0

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

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

maddr-based Configuration

When the maddr parameter is present in the Via header, the response is sent to the IP address specified in the maddr rather than to the received IP address (even when SNAT is enabled).

In the example below, the UAC specifies a maddr set to 2.3.4.5 in the Via header. Consequently, the response from the SIP server makes it to the UAC.

Figure 5-9 maddr Sequence



Example 5-7 shows the complete message trace represented by Figure 5-9.

Example 5-7 Complete maddr Message Trace

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

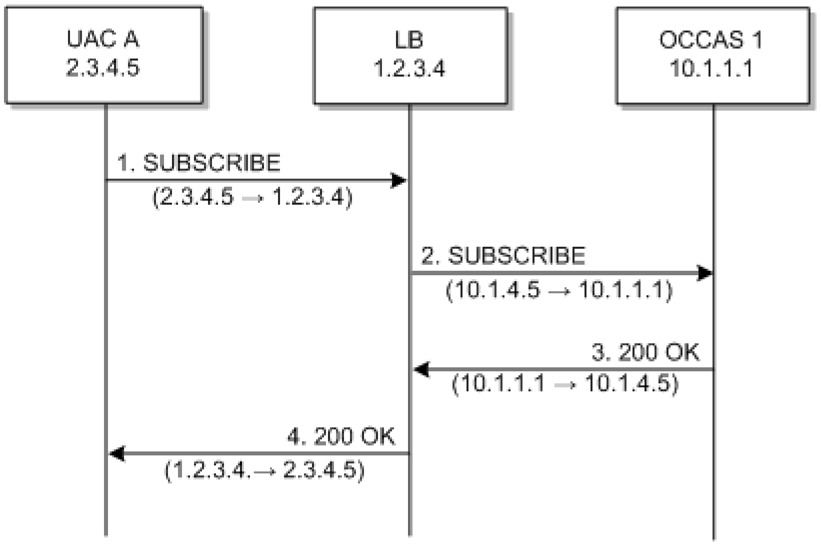
```
Internet Protocol, Src: 2.3.4.5 (2.3.4.5), Dst: 1.2.3.4 (1.2.3.4)
User Datagram Protocol, Src Port: 9999 (9999), Dst Port: sip (5060)
Session Initiation Protocol
    Request-Line: SUBSCRIBE sip:subscribe@1.2.3.4:5060 SIP/2.0
    Message Header
        Via: SIP/2.0/UDP 2.3.4.5:9999;maddr=2.3.4.5;branch=1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

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

Internet Protocol, Src: 10.1.3.4 (10.1.3.4), Dst: 10.1.1.1 (10.1.1.1)
User Datagram Protocol, Src Port: 2222 (2222), Dst Port: sip (5060)
Session Initiation Protocol
    Request-Line: SUBSCRIBE sip:subscribe@1.2.3.4:5060 SIP/2.0
    Message Header
        Via: SIP/2.0/UDP 2.3.4.5:9999;maddr=2.3.4.5;branch=1
        From: sipp <sip:sipp@2.3.4.5>;tag=1
        To: sut <sip:subscribe@1.2.3.4:5060>
        Call-ID: 1-25923@2.3.4.5
        Cseq: 1 SUBSCRIBE
        Contact: sip:sipp@2.3.4.5:9999
        Max-Forwards: 70
        Event: ua-profile
        Expires: 10
        Content-Length: 0

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

Internet Protocol, Src: 10.1.1.1 (10.1.1.1), Dst: 2.3.4.5 (2.3.4.5)
User Datagram Protocol, Src Port: 42316 (42316), Dst Port: 9999 (9999)
Session Initiation Protocol
    Status-Line: SIP/2.0 200 OK
    Message Header
        To: sut <sip:subscribe@1.2.3.4:5060>;tag=82722c03
        Content-Length: 0
        Contact:
<sip:app-12eomtm5h5f77@1.2.3.4:5060;transport=udp;wlsscid=1ae4479ac6ff7
1>
```

rport-based Configuration

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

Figure 5-10 illustrates the SUBSCRIBE flow.

Figure 5-10 rport SUBSCRIBE Sequence



The complete message trace from the figure is shown in Example 5-8 below.

Example 5-8 Complete Message Trace for rport SUBSCRIBE

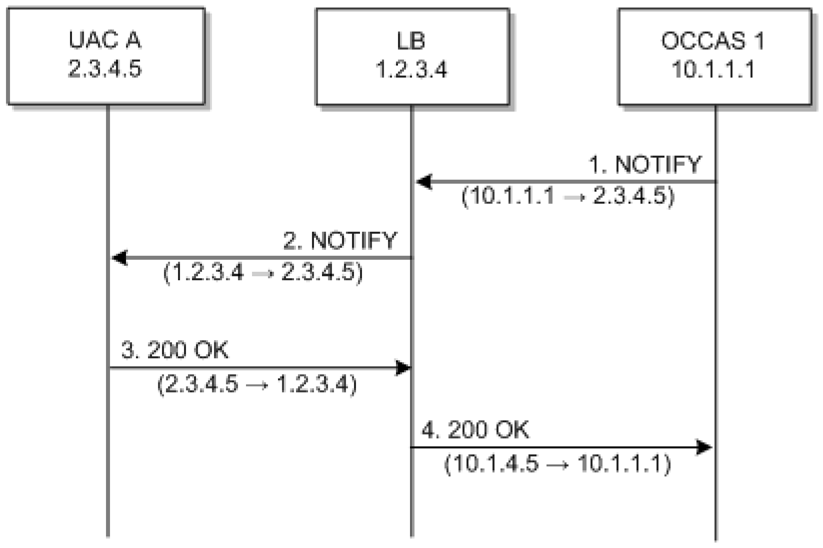
```
No.     Time          Source                Destination           Protocol Info
      1 1.425250    2.3.4.5             1.2.3.4          SIP      Request:
SUBSCRIBE sip:subscribe@1.2.3.4:5060

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

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

Internet Protocol, Src: 10.1.3.4 (10.1.3.4), Dst: 10.1.1.1 (10.1.1.1)
User Datagram Protocol, Src Port: 2222 (2222), Dst Port: sip (5060)
Session Initiation Protocol
    Request-Line: SUBSCRIBE sip:subscribe@1.2.3.4:5060 SIP/2.0
    Message Header
        Via: SIP/2.0/UDP 2.3.4.5:9999;rport;branch=1
        From: sipp <sip:sipp@2.3.4.5>;tag=1
        To: sut <sip:subscribe@1.2.3.4:5060>
        Call-ID: 1-25923@2.3.4.5
        Cseq: 1 SUBSCRIBE
        Contact: sip:sipp@2.3.4.5:9999
        Max-Forwards: 70
        Event: ua-profile
        Expires: 10
        Content-Length: 0

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

Internet Protocol, Src: 10.1.1.1 (10.1.1.1), Dst: 10.1.3.4 (10.1.3.4)
User Datagram Protocol, Src Port: 42316 (42316), Dst Port: 2222 (2222)
Session Initiation Protocol
    Status-Line: SIP/2.0 200 OK
    Message Header
        To: sut <sip:subscribe@1.2.3.4:5060>;tag=82722c03
        Content-Length: 0
        Contact:
<sip:app-12eomtm5h5f77@1.2.3.4:5060;transport=udp;wlsscid=1ae4479ac6ff7
1>
        CSeq: 1 SUBSCRIBE
        Call-ID: 1-25923@2.3.4.5
```

Figure 5-11 shows the NOTIFY message flow.

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

Figure 5-11 rport NOTIFY Sequence



Example 5-9 shows the complete message trace from the figure.

Example 5-9 Complete Message Trace for rport NOTIFY

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

Internet Protocol, Src: 10.1.1.1 (10.1.1.1), Dst: 2.3.4.5 (2.3.4.5)
User Datagram Protocol, Src Port: 42316 (42316), Dst Port: 9999 (9999)
Session Initiation Protocol
    Request-Line: NOTIFY sip:sipp@2.3.4.5:9999 SIP/2.0
    Message Header
        To: sipp <sip:sipp@2.3.4.5>;tag=1
        Content-Length: 0
        Contact:
<sip:app-12eomtm5h5f77@1.2.3.4:5060;transport=udp;wlsscid=1ae4479ac6ff71>
        CSeq: 1 NOTIFY
        Call-ID: 1-25923@2.3.4.5
        Via: SIP/2.0/UDP
1.2.3.4:5060;wlsscid=1ae4479ac6ff71;branch=z9hG4bKc5e4c3b4c22be517133ab749ade
ece4e;rport
        From: sut <sip:subscribe@1.2.3.4:5060>;tag=82722c03
        Max-Forwards: 70

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

Internet Protocol, Src: 1.2.3.4 (1.2.3.4), Dst: 2.3.4.5 (2.3.4.5)
User Datagram Protocol, Src Port: 2222 (2222), Dst Port: 9999 (9999)
Session Initiation Protocol
    Request-Line: NOTIFY sip:sipp@2.3.4.5:9999 SIP/2.0
    Message Header
```

```
        To: sipp <sip:sipp@2.3.4.5>;tag=1
        Content-Length: 0
        Contact:
<sip:app-12eomtm5h5f77@1.2.3.4:5060;transport=udp;wlsscid=1ae4479ac6ff7
1>
        CSeq: 1 NOTIFY
        Call-ID: 1-25923@2.3.4.5
        Via: SIP/2.0/UDP
1.2.3.4:5060;wlsscid=1ae4479ac6ff71;branch=z9hG4bKc5e4c3b4c22be517133ab
749adeece4e;rport
        From: sut <sip:subscribe@1.2.3.4:5060>;tag=82722c03
        Max-Forwards: 70


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

Internet Protocol, Src: 2.3.4.5 (2.3.4.5), Dst: 1.2.3.4 (1.2.3.4)
User Datagram Protocol, Src Port: 9999 (9999), Dst Port: (2222)
Session Initiation Protocol
    Status-Line: SIP/2.0 200 OK
    Message Header
        Via: SIP/2.0/UDP
1.2.3.4:5060;wlsscid=1ae4479ac6ff71;branch=z9hG4bKc5e4c3b4c22be517133ab
749adeece4e;rport
        From: sut <sip:subscribe@1.2.3.4:5060>;tag=82722c03
        To: sipp <sip:sipp@2.3.4.5>;tag=1;tag=1
        Call-ID: 1-25923@2.3.4.5
        CSeq: 1 NOTIFY
        Contact: <sip:2.3.4.5:9999;transport=UDP
```

6

Standards Alignment

This chapter describes how Oracle Communications Converged Application Server complies with various specifications and RFCs.

Overview of Converged Application Server Standards Alignment

Converged Application Server is developed with special attention to Internet Engineering Task Force (IETF) and 3rd Generation Partnership Project (3GPP) specifications. Feature development is prioritized according to general market trends, both observed and predicted. In cases where certain specifications are obsolete or where Internet drafts are formalized as 'Request For Comments' standards, Converged Application Server places priority on compliance with those specifications. In cases where specifications are part of a larger release plan, as with the 3GPP, Oracle prioritizes compliance with the latest ratified release (in this case, Release 12). This should not be presumed to mean that the product is not compliant with subsequent versions of component specifications, although this document does not summarize compliance with those specifications.

Java Sun Recommendation (JSR) Standards Compliance

Converged Application Server is compliant with Java EE version 7.0 and the corresponding Java EE component specifications.

Converged Application Server is further enhanced by the addition of a SIP Servlet container defined by JSR 359: "SIP Servlet API."

Converged Application Server has executed all related Test Compatibility Kits (TCKs) and has met the formal requirements established by Sun Microsystems for formal public statements of compliance.

IETF RFC Compliance

The following table lists the Converged Application Server level of compliance to common IETF Requests for Comment (RFCs) and Internet drafts. The level of compliance is defined as follows:

- **Yes**—Indicates that Converged Application Server directly supports the feature or specification.
- **Yes (Platform)**—Indicates Converged Application Server can host applications or components that implement the RFC. However, the RFC or feature has no impact on the transaction layer of the protocol or on the behavior of the SIP Servlet container.

Table 6-1 Converged Application Server IETF Compliance

RFC or Specification Number	Title	Compliant?	Additional Information
761	DoD Standard Transmission Control Protocol	Yes	Converged Application Server supports applications that conform to this specification. See `http://www.ietf.org/rfc/rfc761.txt`
768	User Datagram Protocol	Yes	Converged Application Server supports applications that conform to this specification. See `http://www.ietf.org/rfc/rfc768.txt`
1157	A Simple Network Management Protocol (SNMP)	Yes	Converged Application Server supports SNMP V2c traps. See `http://www.ietf.org/rfc/rfc1157.txt`
1847	Security Multiparts for MIME: Multipart/ Signed and Multipart/Encrypted	Yes (Platform)	Converged Application Server supports applications that consume or generate signed or encrypted multipart MIME objects. See `http://www.ietf.org/rfc/rfc1847.txt`
1901	Introduction to Community-based SNMPv2	Yes	Converged Application Server supports SNMP V2c traps. See `http://www.ietf.org/rfc/rfc1901.txt`
1905	Protocol Operations for Version 2 of the Simple Network Management Protocol (SNMPv2)	Yes	Converged Application Server supports SNMP V2c traps. See `http://www.ietf.org/rfc/rfc1905.txt`
1906	Transport Mappings for Version 2 of the Simple Network Management Protocol (SNMPv2)	Yes	Converged Application Server supports SNMP over both TCP and UDP. See `http://www.ietf.org/rfc/rfc1906.txt`
1907	Management Information Base for Version 2 of the Simple Network Management Protocol (SNMPv2)	Yes (Platform)	Converged Application Server supports applications that conform to this specification. See `http://www.ietf.org/rfc/rfc1907.txt`
2183	Communicating Presentation Information in Internet Messages: The Content-Disposition Header Field	Yes (Platform)	Converged Application Server supports applications that conform to this specification. See `http://www.ietf.org/rfc/rfc2183.txt`
2246	The TLS Protocol Version 1.0	Yes	Converged Application Server supports TLS. See `http://www.ietf.org/rfc/rfc2246.txt`
2460	Internet Protocol, Version 6 (IPv6) Specification	Yes	Converged Application Server supports applications that conform to this specification. See `http://www.ietf.org/rfc/rfc2460.txt`

Table 6-1 (Cont.) Converged Application Server IETF Compliance

RFC or Specification Number	Title	Compliant?	Additional Information
2543	SIP: Session Initiation Protocol (v1)	Yes	Converged Application Server supports backward compatibility as described in this specification. See `http://www.ietf.org/rfc/rfc2543.txt`
2571	An Architecture for Describing SNMP Management Frameworks	Yes	Converged Application Server supports applications that conform to this specification. See `http://www.ietf.org/rfc/rfc2571.txt`
2572	Message Processing and Dispatching for the Simple Network Management Protocol (SNMP)	Yes	Converged Application Server supports applications that conform to this specification. See `http://www.ietf.org/rfc/rfc2572.txt`
2573	SNMP Applications	Yes	Converged Application Server supports applications that conform to this specification. See `http://www.ietf.org/rfc/rfc2573.txt`
2574	User-based Security Model (USM) for version 3 of theSimple Network Management Protocol (SNMPv3)	Yes	Converged Application Server supports applications that conform to this specification. See `http://www.ietf.org/rfc/rfc2574.txt`
2575	View-based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP)	Yes	Converged Application Server supports applications that conform to this specification. See `http://www.ietf.org/rfc/rfc2575.txt`
2576	Coexistence between Version 1, Version 2, and Version 3 of the Internet-standard Network Management Framework	Yes	Converged Application Server supports applications that conform to this specification. See `http://www.ietf.org/rfc/rfc2576.txt`
2616	Hypertext Transfer Protocol — HTTP 1.1	Yes	Converged Application Server supports applications that conform to this specification. See `http://www.ietf.org/rfc/rfc2616.txt`
2617	HTTP Authentication: Basic and Digest Access Authentication	Yes	Converged Application Server supports applications that conform to this specification. See `http://www.ietf.org/rfc/rfc2617.txt`

Table 6-1 (Cont.) Converged Application Server IETF Compliance

RFC or Specification Number	Title	Compliant?	Additional Information
2782	A DNS RR for specifying the location of services (DNS SRV)	Yes	Converged Application Server supports applications that conform to this specification. See `http://www.ietf.org/rfc/rfc2782.txt`
2786	Diffie-Helman USM Key Management Information Base and Textual Convention	Yes	Converged Application Server supports applications that conform to this specification. See `http://www.ietf.org/rfc/rfc2786.txt`
2806	URLs for Telephone Calls	Yes	Converged Application Server supports applications that conform to this specification. See `http://www.ietf.org/rfc/rfc2806.txt`
2848	The PINT Service Protocol: Extensions to SIP and SDP for IP Access to Telephone Call Services	Yes (Platform)	Note that implementing PINT services implies a pre-IMS architecture. Although Oracle favors the 3GPP/TISPAN architecture and approach to class 4/5 Service Emulation and does not advocate PINT, it is possible to implement PINT service elements using Converged Application Server. See `http://www.ietf.org/rfc/rfc2848.txt`
2960	Stream Control Transmission Protocol	Yes	SCTP supported only for Diameter traffic. See `http://www.ietf.org/rfc/rfc2960.txt`
2976	The SIP INFO Method	Yes	Converged Application Server supports applications that conform to this specification. See `http://www.ietf.org/rfc/rfc2976.txt`
3204	MIME media types for ISUP and QSIG Objects	Yes (Platform)	Converged Application Server does not directly consume or generate ISUP and QSIG objects, but it supports applications that consume or generate these objects. See `http://www.ietf.org/rfc/rfc3204.txt`
3261	SIP: Session Initiation Protocol	Yes	Converged Application Server supports applications that conform to this specification. See `http://www.ietf.org/rfc/rfc3261.txt`
3262	Reliability of Provisional Responses in the Session Initiation Protocol (SIP)	Yes	Converged Application Server supports applications that conform to this specification. See `http://www.ietf.org/rfc/rfc3262.txt`
3263	Session Initiation Protocol (SIP): Locating SIP Servers	Yes	Converged Application Server supports applications that conform to this specification. See `http://www.ietf.org/rfc/rfc3263.txt`
3264	An Offer/Answer Model with Session Description Protocol (SDP)	Yes (Platform)	Converged Application Server supports applications that conform to this specification. See `http://www.ietf.org/rfc/rfc3264.txt`

Table 6-1 (Cont.) Converged Application Server IETF Compliance

RFC or Specification Number	Title	Compliant?	Additional Information
3265	Session Initiation Protocol (SIP)-Specific Event Notification	Yes (Platform)	Converged Application Server supports applications that conform to this specification. See `http://www.ietf.org/rfc/rfc3265.txt`
3268	Advanced Encryption Standard (AES) Ciphersuites for Transport Layer Security (TLS)	Yes (Platform)	Converged Application Server supports cryptographic services, but specific algorithms that are used are subject to local availability and export control. See `http://www.ietf.org/rfc/rfc3268.txt`
3311	The Session Initiation Protocol (SIP) UPDATE Method	Yes	Converged Application Server supports applications that conform to this specification. See `http://www.ietf.org/rfc/rfc3311.txt`
3312	Integration of Resource Management and Session Initiation Protocol (SIP).	Yes (Platform)	Converged Application Server supports applications that conform to this specification. See `http://www.ietf.org/rfc/rfc3312.txt`
3313	Private Session Initiation Protocol (SIP) Extensions for Media Authorization	Yes (Platform)	Converged Application Server supports applications that conform to this specification. See `http://www.ietf.org/rfc/rfc3313.txt`
3323	A Privacy Mechanism for the Session Initiation Protocol (SIP)	Yes (Platform)	Converged Application Server supports applications that conform to this specification. See `http://www.ietf.org/rfc/rfc3323.txt`
3325	Private Extensions to the Session Initiation Protocol (SIP) for Asserted Identity within Trusted Networks	Yes	Converged Application Server supports applications that conform to this specification. See `http://www.ietf.org/rfc/rfc3325.txt`
3326	The Reason Header Field for the Session Initiation Protocol (SIP)	Yes (Platform)	Converged Application Server supports applications that conform to this specification. See `http://www.ietf.org/rfc/rfc3326.txt`
3327	Session Initiation Protocol (SIP) Extension Header Field for Registering Non-Adjacent Contacts.	Yes (Platform)	Converged Application Server supports applications that conform to this specification. See `http://www.ietf.org/rfc/rfc3327.txt`

Table 6-1 (Cont.) Converged Application Server IETF Compliance

RFC or Specification Number	Title	Compliant?	Additional Information
3351	User Requirements for the Session Initiation Protocol (SIP) in Support of Deaf, Hard of Hearing and Speech-impaired Individuals	Yes (Platform)	Converged Application Server supports applications that conform to this specification. See `http://www.ietf.org/rfc/rfc3351.txt`
3372	Session Initiation Protocol for Telephones (SIP-T): Context and Architectures	Yes (Platform)	Converged Application Server supports applications that conform to this specification. See `http://www.ietf.org/rfc/rfc3372.txt`
3420	Internet Media Type message/sipfrag		Converged Application Server supports applications that conform to this specification. See `http://www.ietf.org/rfc/rfc3420.txt`
3428	Session Initiation Protocol (SIP) Extension for Instant Messaging	Yes	Converged Application Server supports applications that conform to this specification. See `http://www.ietf.org/rfc/rfc3428.txt`
3455	Private Header (P-Header) Extensions to the Session Initiation Protocol (SIP) for the 3rd-Generation Partnership Project (3GPP)	Yes (Platform)	Converged Application Server supports applications that conform to this specification. See `http://www.ietf.org/rfc/rfc3455.txt`
3515	The Session Initiation Protocol (SIP) Refer Method.	Yes	Converged Application Server supports applications that conform to this specification. See `http://www.ietf.org/rfc/rfc3515.txt`
3524	Mapping of Media Streams to Resource Reservation Flows	Yes (Platform)	Converged Application Server supports applications that conform to this specification. See `http://www.ietf.org/rfc/rfc3524.txt`
3556	Session Description Protocol (SDP) Bandwidth Modifiers for RTP Control Protocol (RTCP) Bandwidth	Yes (Platform)	Converged Application Server supports applications that conform to this specification. See `http://www.ietf.org/rfc/rfc3556.txt`

Table 6-1 (Cont.) Converged Application Server IETF Compliance

RFC or Specification Number	Title	Compliant?	Additional Information
3578	Mapping of Integrated Services Digital Network (ISDN) User Part (ISUP) Overlap Signalling to the Session Initiation Protocol (SIP)	Yes (Platform)	Converged Application Server supports applications that conform to this specification, but it does not provide an ISUP interface. See `http://www.ietf.org/rfc/rfc3578.txt`
3581	An Extension to the Session Initiation Protocol (SIP) for Symmetric Response Routing	Yes	Converged Application Server supports applications that conform to this specification. See `http://www.ietf.org/rfc/rfc3581.txt`
3589	Diameter Command Codes for Third Generation Partnership Project (3GPP) Release 5	Yes	Converged Application Server supports applications that conform to this specification. See `http://www.ietf.org/rfc/rfc3589.txt`
3588	Diameter Base Protocol	Yes	Converged Application Server supports applications that conform to this specification. See `http://www.ietf.org/rfc/rfc3588.txt`
3605	Real Time Control Protocol (RTCP) attribute in Session Description Protocol ((SDP)	Yes (Platform)	Converged Application Server supports applications that conform to this specification. See `http://www.ietf.org/rfc/rfc3605.txt`
3608	Session Initiation Protocol (SIP) Extension Header Field for Service Route Discovery During Registration.	Yes (Platform)	Converged Application Server supports applications that conform to this specification, but it does not provide a means of storing the ServiceRoute established during registration. This functionality can be implemented as part of the application. See `http://www.ietf.org/rfc/rfc3608.txt`
3665	Session Initiation Protocol (SIP) Basic Call Flow Examples.	Yes	Converged Application Server supports applications that conform to this specification. See `http://www.ietf.org/rfc/rfc3665.txt`
3666	Session Initiation Protocol (SIP) Public Switched Telephone Network (PSTN) Call Flows	Yes (Platform)	Converged Application Server supports applications that conform to this specification. See `http://www.ietf.org/rfc/rfc3666.txt`
3680	A Session Initiation Protocol (SIP) Event Package for Registrations	Yes (Platform)	Converged Application Server supports applications that conform to this specification. See `http://www.ietf.org/rfc/rfc3680.txt`

Table 6-1 (Cont.) Converged Application Server IETF Compliance

RFC or Specification Number	Title	Compliant?	Additional Information
3689	General Requirements for Emergency Telecommunication Service (ETS)	Yes (Platform)	Converged Application Server supports applications that conform to this specification. See `http://www.ietf.org/rfc/rfc3689.txt`
3690	IP Telephony Requirements for Emergency Telecommunication Service (ETS)	Yes (Platform)	Converged Application Server supports applications that conform to this specification. See `http://www.ietf.org/rfc/rfc3690.txt`
3702	Authentication, Authorization, and Accounting Requirements for the Session Initiation Protocol (SIP)	Yes	Converged Application Server version supports JDBC and LDAP. See `http://www.ietf.org/rfc/rfc3702.txt`
3725	Best Current Practices for Third Party Call Control (3pcc) in the Session Initiation Protocol (SIP)	Yes	Converged Application Server supports applications that conform to this specification. See `http://www.ietf.org/rfc/rfc3725.txt`
3761	The E.164 to Uniform Resource Identifiers (URI) Dynamic Delegation Discovery System (DDDS) Application (ENUM)	Yes	Converged Application Server supports applications that conform to this specification. See `http://www.ietf.org/rfc/rfc3761.txt`
3764	Enumservice Registration for Session Initiation Protocol (SIP) Addresses-of-Record	Yes (Platform)	Converged Application Server supports applications that conform to this specification. See `http://www.ietf.org/rfc/rfc3764.txt`
3824	Using E.164 numbers with the Session Initiation Protocol (SIP)	Yes	Converged Application Server supports applications that conform to this specification. See `http://www.ietf.org/rfc/rfc3824.txt`
3826	The Advanced Encryption Standard (AES) Cipher Algorithm in the SNMP User-based Security Model	Yes	Converged Application Server supports applications that conform to this specification. See `http://www.ietf.org/rfc/rfc3826.txt`

Table 6-1 (Cont.) Converged Application Server IETF Compliance

RFC or Specification Number	Title	Compliant?	Additional Information
3840	Indicating User Agent Capabilities in the Session Initiation Protocol (SIP)	Yes (Platform)	Converged Application Server supports applications that conform to this specification. See `http://www.ietf.org/rfc/rfc3840.txt`
3841	Caller Preferences for the Session Initiation Protocol (SIP)	Yes (Platform)	Converged Application Server supports applications that conform to this specification. See `http://www.ietf.org/rfc/rfc3841.txt`
3853	S/MIME Advanced Encryption Standard (AES) Requirement for the Session Initiation Protocol (SIP)	Yes (Platform)	Converged Application Server supports applications that conform to this specification. See `http://www.ietf.org/rfc/rfc3853.txt`
3891	The Session Initiation Protocol (SIP) 'Replaces' Header	Yes (Platform)	Converged Application Server supports applications that conform to this specification. See `http://www.ietf.org/rfc/rfc3891.txt`
3892	The Session Initiation Protocol (SIP) Referred-By Mechanism	Yes (Platform)	Converged Application Server supports applications that conform to this specification. See `http://www.ietf.org/rfc/rfc3892.txt`
3893	Session Initiation Protocol (SIP) Authenticated Identity Body (AIB) Format	Yes (Platform)	Converged Application Server supports applications that conform to this specification. See `http://www.ietf.org/rfc/rfc3893.txt`
3903	Session Initiation Protocol (SIP) Extension for Event State Publication	Yes	Converged Application Server supports applications that conform to this specification. See `http://www.ietf.org/rfc/rfc3903.txt`
3911	The Session Initiation Protocol (SIP) "Join" Header	Yes (Platform)	Converged Application Server supports applications that conform to this specification. See `http://www.ietf.org/rfc/rfc3911.txt`
3959	The Early Session Disposition Type for the Session Initiation Protocol (SIP)		Converged Application Server supports applications that conform to this specification. See `http://www.ietf.org/rfc/rfc3959.txt`
3960	Early Media and Ringing Tone Generation in the Session Initiation Protocol (SIP)	Yes (Platform)	Converged Application Server supports applications that conform to this specification. See `http://www.ietf.org/rfc/rfc3960.txt`
3966	The tel URI for Telephone Numbers	Yes	See `http://www.ietf.org/rfc/rfc3966.txt`

Table 6-1 (Cont.) Converged Application Server IETF Compliance

RFC or Specification Number	Title	Compliant?	Additional Information
4028	Session Timers in the Session Initiation Protocol (SIP)	Yes (Platform)	Converged Application Server supports applications that conform to this specification. See `http://www.ietf.org/rfc/rfc4028.txt`
4032	Update to the Session Initiation Protocol (SIP) Preconditions Framework	Yes (Platform)	Converged Application Server supports applications that conform to this specification. See `http://www.ietf.org/rfc/rfc4032.txt`
4244	An Extension to the Session Initiation Protocol (SIP) for Request History Information	Yes (Platform)	Converged Application Server supports applications that conform to this specification. See `http://www.ietf.org/rfc/rfc4244.txt`
4320	Actions Addressing Identified Issues with the Session Initiation Protocol's (SIP) Non-INVITE Transaction	Yes	Converged Application Server supports applications that conform to this specification, specifically including: • If the application or proxy does not respond to a non-invite request before TimerE reaches T2, the container responds with a 100 TRYING message. • A system parameter that disables this feature: **-Dwlss.send100ForNonInviteTransaction=false**. The parameter is true by default. See `http://www.ietf.org/rfc/rfc4320.txt`
4321	Problems Identified Associated with the Session Initiation Protocol's (SIP) Non_INVITE Transaction	Yes	Converged Application Server supports applications that conform to this specification. See `http://www.ietf.org/rfc/rfc4321.txt`
4474	Enhancements for Authenticated Identity Management in the Session Initiation Protocol (SIP)	Yes	Converged Application Server supports applications that conform to this specification. See `http://www.ietf.org/rfc/rfc4474.txt`.
4483	A Mechanism for Content Indirection in Session Initiation Protocol (SIP) Messages	Yes (Platform)	Converged Application Server supports applications that conform to this specification. See `http://www.ietf.org/rfc/rfc4483.txt`.
4566	SDP: Session Description Protocol	Yes	Converged Application Server supports applications that consume or generate SDP. See `http://www.ietf.org/rfc/rfc4566.txt`
4916	Connected Identity in the Session Initiation Protocol (SIP)	Yes	See `https://tools.ietf.org/html/rfc4916`

Table 6-1 (Cont.) Converged Application Server IETF Compliance

RFC or Specification Number	Title	Compliant?	Additional Information
5393	Addressing an Amplification Vulnerability in Session Initiation Protocol (SIP) Forking Proxies	Partial	Converged Application Server supports the Max-Breadth Header portion of JSR 359, section 12.2.11, "Max-Breadth Check." Specifically, it implements a Max-Breadth header to limit the number of parallel forks that can be made on a SIP request by the downstream proxies. See `https://www.jcp.org/en/jsr/detail?id=359`
5626	Managing Client-Initiated Connections in the Session Initiation Protocol (SIP)	Yes	Converged Application Server supports JSR 359, section 17.4, "Managing Client Initiated Connections," which includes support for UDP/TCP with cluster deployment. See `https://www.jcp.org/en/jsr/detail?id=359`
5630	The Use of the SIPS URI Scheme in the Session Initiation Protocol (SIP)	Yes	When a proxy sends a request using a SIPS Request-URI and receives one of: • A 3XX response with a SIP Contact header field • A 416 response • A 480 (Temporarily Unavailable) response with a Warning header with warn-code 380 "SIPS Not Allowed" response The proxy must not recurse on the response. In this case, the proxy should forward the best response instead of recursing. This allows the UAC to take the appropriate action. When a proxy sends a request using a SIP Request-URI and receives: • A 3XX response with a SIPS Contact header field • A 480 (Temporarily Unavailable) response with a Warning header with warn-code 381 "SIPS Required" The proxy must not recurse on the response. In this case, the proxy should forward the best response instead of recursing. This allows the UAC to take the appropriate action.
5888	The Session Description Protocol (SDP) Grouping Framework	Yes (Platform)	Converged Application Server supports applications that conform to this specification. See `http://www.ietf.org/rfc/rfc5888.txt`
5806	Diversion Indication in SIP	Yes	Converged Application Server supports applications that conform to this specification. See `https://datatracker.ietf.org/doc/rfc5806/`
5954	Essential Correction for IPv6 ABNF and URI Comparison in RFC 3261	Yes	Converged Application Server now includes a task to update the **equals()** and **hashcode()** methods of **SipURIImpl** based on this RFC. See `https://tools.ietf.org/html/rfc5954`

Table 6-1 (Cont.) Converged Application Server IETF Compliance

RFC or Specification Number	Title	Compliant?	Additional Information
6026	Correct Transaction Handling for 2xx Responses to Session Initiation Protocol (SIP) INVITE Requests	Yes	Converged Application Server supports applications that conform to this specification. See `https://tools.ietf.org/html/rfc6026`
6141	Re-INVITE and Target-Refresh Request Handling in the Session Initiation Protocol (SIP)	Yes	Converged Application Server supports applications that conform to this specification. See `https://tools.ietf.org/html/rfc6141`
6665	SIP-Specific Event Notification	Yes	Converged Application Server supports applications that conform to this specification. See `https://tools.ietf.org/html/rfc6665`
draft-donovan-mmusic-183-00	SIP 183 Session Progress Message Draft	Yes (Platform)	Converged Application Server supports applications that conform to this specification. See `https://datatracker.ietf.org/doc/html/draft-donovan-mmusic-183-00`
draft-reeder-snmpv3-usm-3desede-00	Extension to the User-Based Security Model (USM) to Support Triple-DES EDE in "Outside" CBC Mode	Yes	See `https://tools.ietf.org/id/draft-reeder-snmpv3-usm-3desede-00.txt`

3GPP R12 Specification Conformance

Converged Application Server is fully compliant with the latest 3GPP Release 12 specification, and does not impose any restrictions on implementing applications or functions that are compliant with those associated with the Application Server entity described in the specification. In some cases, applications must implement support for SIP methods or headers. The default behavior of the Converged Application Server Sip Servlet Container is to pass unrecognized headers, request methods and payloads to SIP Servlets using normal SIP Servlet API procedures.

A

SIP Servlet API Service Invocation

This appendix describes the Service invocation method of the SIP Servlet API (JSR 116).

SIP Servlet API Overview

The SIP Servlet API provides a model for application composition and interaction. Service Interaction which is analogous with a simplistic implementation of the Service Capability Interaction Manager (SCIM) alluded to by the 3GPP. Handling of all incoming requests is governed by the Converged Application Server SIP Servlet Container in accordance with the SIP Servlet API specification.

Oracle Communications Converged Application Server's SIP Servlet Container filters received Initial SIP requests and applies a set of defined rules (Servlet Mapping Rules) to determine which SIP Servlets within the deployed applications shall be invoked to service that particular request. This order is always sequential and is defined in a configuration file built up through successive deployments of SIP applications.

Within the deployment descriptor for each SIP application that is deployed, a sequence of conditions, called Servlet Mapping Rules, is defined. These rules determine which Servlets will handle any initial request. As the request object is "routed" between Servlets, the path from Servlet to Servlet is recorded in a fashion equivalent to that of the Record-Route and Via headers used in SIP requests. This route is stored as part of the SIP application session and is appended to subsequent requests within the same dialogue in either "forward" or "reverse" order depending on the orientation of the “From" and “To" tags for the request. This internal "route" is stripped from the request object before a SIP request leaves Converged Application Server and is not visible to external SIP servers. It is again added whenever a new request within an existing dialog is received.

The SIP Servlets (SIP/HTTP application) that are invoked in this manner are unaware that any other SIP/HTTP application exists. This is one of the fundamental characteristics of the SIP Servlet programming model. Making maximal use of this model requires that the Servlet container be treated by the developer as if it is a logical sub-network, with the container effectively acting as an intermediary proxy. In many ways, the SIP Servlet Container may be compared with the Serving CSCF function in an IMS architecture.

Servlet Mapping Rules: Objects, Properties and Conditions

Servlet mapping rules are defined by the service developer and are detailed in the application's *deployment descriptor*. A deployment descriptor is an XML-based text file whose elements describe how to assemble and deploy the unit into a specific environment. Each element consists of a tag and a value expressed in the following syntax: `<tag>value</tag>`. Usually deployment descriptors are automatically generated by deployment tools, so you will not have to manage them directly. Deployment descriptor elements contain behavioral information about components not included directly in code. Their purpose is to tell the Deployer how to deploy an application, not tell the server how to manage components at runtime.

In the context of SIP applications, the deployment descriptor is contained within the Servlet archive (SAR) file that is deployed on Converged Application Server. There may be more than one Servlet mapping rule defined within the deployment descriptor for the application (SIP/HTTP application). In this case, these rules must be applied in the order in which they are defined in the deployment descriptor.

Example A-1 provides an example of a simple Servlet mapping rule found in a typical deployment descriptor.

Note:

Servlet mapping rules are entirely concerned with the content of the SIP message being processed. It is not possible to use information regarding the actual IP address and port number on which the request was received as service trigger points unless this information matches the request URI of the Sip message.

The Servlet mapping rule shown in Example A-1 illustrates the following Boolean expression:

(Method="INVITE" OR Method = "MESSAGE" OR Method="SUBSCRIBE") AND

(Method="INVITE" OR Method = "MESSAGE" OR (NOT Header = "from" Match = "Bob"))

Note:

This is the same logical condition used in the Initial filter Criteria example provided in 3GPP TS 29.228 Annex C expressed as a Servlet Mapping Rule.

Example A-1 Example Servlet Mapping Rule

```
<servlet-mapping>
<servlet-name>servlet1</servlet-name>
<pattern>
        <and>
        <or>
                <equal>
                <var>request.method</var>
                <value>INVITE</value>
                </equal>
                        <equal>
                <var>request.method</var>
                <value>MESSAGE</value>
                        </equal>
                <equal>
                <var>request.method</var>
                <value>SUBSCRIBE</value>
                        </equal>
        </or>
        <or>
                <equal>
                <var>request.method</var>
                <value>INVITE</value>
                </equal>
                        <equal>
                <var>request.method</var>
                <value>MESSAGE</value>
                    </equal>
                <not>
                    <equal>
                        <var>request.from.display-name</var>
                        <value>Bob</value>
                    </equal>
                </not>
        </or>
        </and>
</pattern>
</servlet-mapping>
```

Supported Service Trigger Points

Service Point Triggers are the attributes of a SIP request that may be evaluated by Servlet Mapping Rules. See "Section 11.1: Triggering Rules" in the JSR 116 specification for more information.

Request Object

The Request Object is a Java representation of a SIP request.

- **method**: the request method, a string
- **uri**: the request URI; for example a SipURI or a TelURL
- **from**: an Address representing the value of the From header
- **to**: an Address representing the value of the To header

URI

- **scheme**: the URI scheme

SipURI (extends URI)

- **scheme**: a literal string – either “sip" or “sips"
- **user**: the “user" part of the SIP/SIPS URI
- **host**: the “host" part of the SIP/SIPS URI. This may be a domain name or a dotted decimal IP address.
- **port**: the URI port number in decimal format; if absent the default value is used (5060 for UDP and TCP, 5061 for TLS).
- **tel**: if the “user" parameter is not “phone", this variable is undefined. Otherwise, its value is the telephone number contained in the “user" part of the SIP/SIPS URI with visual separators stripped. This variable is always matched case insensitively (the telephone numbers may contain the symbols ‘A', ‘B', ‘C' and ‘D').
- **param.name**: value of the named parameter within a SIP/SIPS URI; name must be a valid SIP/SIPS URI parameter name.

TelURL (extends URI)

- **scheme**: always the literal string “tel"
- **tel**: the tel URL subscriber name with visual separators stripped off
- **param.name**: value of the named parameter within a tel URL; name must be a valid tel URL parameter name

Address

- **uri**: the URI object; see URI, SipURI, TelURL types above
- **display-name**: the display-name portion of the From or To header

Conditions and Logical Connectors

- **equal**: compares the value of a variable with a literal value and evaluates to true if the variable is defined and its value equals that of the literal. Otherwise, the result is false.
- **exists**: takes a variable name and evaluates to true if the variable is defined, and false otherwise.
- **contains**: evaluates to true if the value of the variable specified as the first argument contains the literal string specified as the second argument.
- **subdomain-of**: given a variable denoting a domain name (SIP/SIPS URI host) or telephone subscriber (tel property of SIP or Tel URLs), and a literal value, this operator returns true if the variable denotes a subdomain of the domain given by the literal value. Domain names are matched according to the DNS definition of what constitutes a subdomain; for example, the domain names “example.com" and “research.example.com“are both subdomains of “example.com". IP addresses may be given as arguments to this operator; however, they only match exactly. In the case of the tel variables, the subdomain-of operator evaluates to true if the telephone number denoted by the first argument has a prefix that matches the literal value given in the second argument; for example, the telephone number “1 212 555 1212" would be considered a subdomain of “1212555".
- **and**: contains a number of conditions and evaluates to true if and only if all contained conditions evaluate to true
- **or**: contains a number of conditions and evaluates to true if and only if at least one contained condition evaluates to true
- **not**: negates the value of the contained condition.

The equal and contains operators optionally ignore character case when making comparisons. The default is case-sensitive matching.