Oracle® Communications Converged Application Server

Developing SIP Applications

Release 4.0

August 2008

ORACLE®

Contents

1. Overview of SIP Servlets

What is a SIP Servlet? 1-1

Differences from HTTP Servlets 1-3

Multiple Responses. 1-3

Receiving Responses 1-4

Proxy Functions 1-6

Message Body. 1-6

ServletRequest. 1-7

ServletResponse. 1-7

SipServletMessage. 1-8

Roles of a Servlet Container 1-8

Application Management. 1-8

SIP Messaging 1-10

Utility Functions 1-13

2. Porting Existing Applications to Oracle Communications Converged Application Server

Application Router and Legacy Application Composition 2-2

SipSession and SipApplicationSession Not Serializable. 2-2

SipServletResponse.setCharacterEncoding() API Change 2-2

Transactional Restrictions for SipServletRequest and SipServletResponse 2-3

Immutable Parameters for New Parameterable Interface 2-3

Stateless Transaction Proxies Deprecated 2-4
Backward-Compatibility Mode for v1.0 Deployments 2-4
Validation Warnings for v1.0 Servlet Deployments 2-4
Modifying Committed Messages 2-4
Path Header as System Header 2-5
SipServletResponse.createPrack() Exception 2-5
Proxy.proxyTo() Exceptions 2-5
Changes to Proxy Branch Timers 2-5
Deprecated APIs 2-6
SNMP MIB Changes 2-6
Renamed Diagnostic Monitors and Actions 2-7

3. Requirements and Best Practices for SIP Applications

Overview of Developing Distributed Applications for Oracle Communications Converged Application Server 3-2
Applications Must Not Create Threads 3-2
Servlets Must Be Non-Blocking 3-3
Store all Application Data in the Session 3-3
All Session Data Must Be Serializable 3-3
. . . . Use setAttribute() to Modify Session Data in "No-Call" Scope3-4
send() Calls Are Buffered 3-6
Mark SIP Servlets as Distributable 3-7
Use SipApplicationSessionActivationListener Sparingly 3-7
Observe Best Practices for Java EE Applications 3-7
. . . . 3-7

4. Using Compact and Long Header Formats for SIP Messages

Overview of Header Format APIs and Configuration 4-1
Summary of Compact Headers 4-2

Assigning Header Formats with WlssSipServletMessage. 4-2
Summary of API and Configuration Behavior 4-3

5. Composing SIP Applications

Application Composition Model 5-1
Using the Default Application Router 5-3
Configuring a Custom Application Router 5-4
Session Key-Based Request Targeting 5-6

6. Developing Converged Applications

Overview of Converged Applications 6-1
Assembling and Packaging a Converged Application. 6-2
Working with SIP and HTTP Sessions 6-2
    Modifying the SipApplicationSession 6-5
Using the Converged Application Example 6-6

7. Developing Custom Profile Providers

Overview of the Profile Service API 7-1
Implementing Profile API Methods. 7-3
Configuring and Packaging Profile Providers 7-4
    Mapping Profile Requests to Profile Providers 7-5
Configuring Profile Providers Using the Administration Console 7-6

8. Using Content Indirection in SIP Servlets

Overview of Content Indirection 8-1
Using the Content Indirection API 8-2
Additional Information 8-2

9. Securing SIP Servlet Resources

Overview of SIP Servlet Security 9-1

Oracle Communications Converged Application Server Role Mapping Features 9-2
Using Implicit Role Assignment 9-3
Assigning Roles Using security-role-assignment 9-4
Important Requirement for Oracle Communications Converged Application Server 9-4
Assigning Roles at Deployment Time 9-6
Dynamically Assigning Roles Using the Administration Console 9-6
Assigning run-as Roles 9-8
Role Assignment Precedence for SIP Servlet Roles 9-8
Debugging Security Features 9-9
weblogic.xml Deployment Descriptor Reference 9-9

10.Developing SIP Servlets Using Eclipse

Overview 10-1
SIP Servlet Organization 10-2
Setting Up the Development Environment 10-2
Creating a Oracle Communications Converged Application Server Domain 10-3
Configure the Default Eclipse JVM 10-3
Creating a New Eclipse Project 10-4
Creating an Ant Build File 10-4
Building and Deploying the Project 10-6
Debugging SIP Servlets 10-7

11.Enabling Message Logging

Overview 11-1
Enabling Message Logging 11-2
Specifying a Predefined Logging Level. 11-2
Customizing Log Records 11-3
Specifying Content Types for Unencrypted Logging 11-5

Example Message Log Configuration and Output 11-6
Configuring Log File Rotation 11-8

12.Generating SNMP Traps from Application Code

Overview 12-1
Requirement for Accessing SipServletSnmpTrapRuntimeMBean 12-2
Obtaining a Reference to SipServletSnmpTrapRuntimeMBean 12-3
Generating a SNMP Trap 12-5

CHAPTER 1

Overview of SIP Servlets

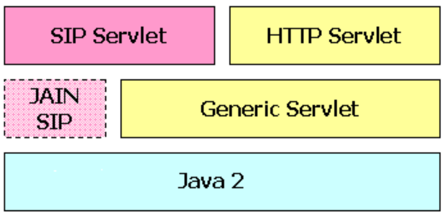
What is a SIP Servlet?

The SIP Servlet API is a part of JAIN APIs and is standardized as JSR289 of JCP (Java Community Process).

Note: In this document, the term "SIP Servlet" is used to represent the API, and "SIP servlet" is used to represent an application created with the API.

Java EE provides Java Servlet that is a main technology of building Web applications. Although Java Servlet is used only to develop HTTP protocol-based applications on a Web application server, it basically has functions as a generic API for server applications. SIP Servlet is defined as the generic servlet API with SIP-specific functions added.

Figure 1-1 Servlet API and SIP Servlet API



SIP Servlets are very similar to HTTP Servlets, and HTTP servlet developers will quickly adapt to the programming model. The service level defined by both HTTP and SIP Servlets is very similar, and you can easily design applications that support both HTTP and SIP. Listing 1 shows an example of a simple SIP servlet.

Listing 1-1 List 1: SimpleSIPServlet.java

```
package com.bea.example.simple;
import java.io.IOException;
import javax.servlet.*;
import javax.servlet.sip.*;

public class SimpleSIPServlet extends SipServlet {

    protected void doMessage(SipServletRequest req)
       throws ServletException, IOException
    {
       SipServletResponse res = req.createResponse(200);
       res.send();
    }
}
```

The above example shows a simple SIP servlet that sends back a 200 OK response to the SIP MESSAGE request. As you can see from the list, SIP Servlet and HTTP Servlet have many things in common:

1. Servlets must inherit the base class provided by the API. HTTP servlets must inherit HttpServlet, and SIP servlets must inherit SipServlet.
2. Methods doXxx must be overridden and implemented. HTTP servlets have doGet/doPost methods corresponding to GET/POST methods. Similarly, SIP servlets have doXxx methods corresponding to the method name (in the above example, the MESSAGE method). Application developers override and implement necessary methods.
3. The lifecycle and management method (init, destroy) of SIP Servlet are exactly the same as HTTP Servlet. Manipulation of sessions and attributes is also the same.
4. Although not appeared in the API, there is a deployment descriptor called sip.xml for a SIP servlet, which corresponds to web.xml. Application developers and service managers can edit this file to configure applications using multiple SIP servlets.

However, there are several differences between SIP and HTTP servlets. A major difference comes from protocols. The next section describes these differences as well as features of SIP servlets.

Differences from HTTP Servlets

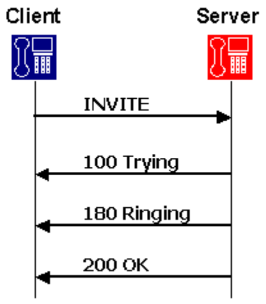
Multiple Responses

You might notice from the List 1 that the doMessage method has only one argument. In HTTP, a transaction consists of a pair of request and response, so arguments of a doXxx method specify a request (HttpServletRequest) and its response (HttpServletResponse). An application takes information such as parameters from the request to execute it, and returns its result in the body of the response.

```
protected void doGet(HttpServletRequest req, HttpServletResponse res)
   throws ServletException, IOException
```

For SIP, more than one response may be returned to a single request.

Figure 1-2 Example of Request and Response in SIP



The above figure shows an example of a response to the INVITE request. In this example, the server sends back three responses 100, 180, and 200 to the single INVITE request. To implement such sequence, in SIP Servlet, only a request is specified in a doXxx method, and an application generates and returns necessary responses in an overridden method.

Currently, SIP Servlet defines the following doXxx methods:

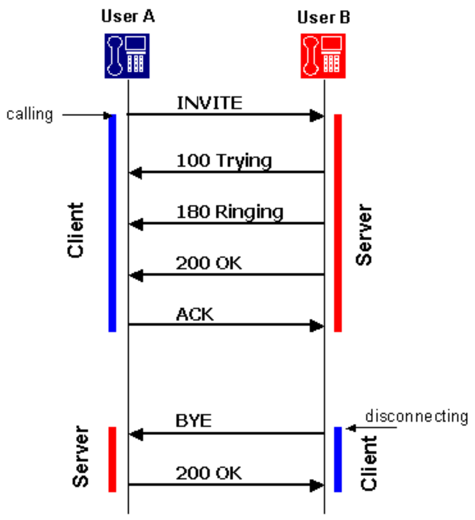
```
protected void doInvite(SipServletRequest req);
protected void doAck(SipServletRequest req);

protected void doOptions(SipServletRequest req);

protected void doBye(SipServletRequest req);

protected void doCancel(SipServletRequest req);

protected void doSubscribe(SipServletRequest req);

protected void doNotify(SipServletRequest req);

protected void doMessage(SipServletRequest req);

protected void doInfo(SipServletRequest req);

protected void doPrack(SipServletRequest req);
```

Receiving Responses

One of the major features of SIP is that roles of a client and server are not fixed. In HTTP, Web browsers always send HTTP requests and receive HTTP responses: They never receive HTTP requests and send HTTP responses. In SIP, however, each terminal needs to have functions of both a client and server.

For example, both of two SIP phones must call to the other and disconnect the call.

Figure 1-3 Relationship between Client and Server in SIP



The above example indicates that a calling or disconnecting terminal acts as a client. In SIP, roles of a client and server can be changed in one dialog. This client function is called UAC (User Agent Client) and server function is called UAS (User Agent Server), and the terminal is called UA (User Agent). SIP Servlet defines methods to receive responses as well as requests.

```
protected void doProvisionalResponse(SipServletResponse res);
```

```
protected void doSuccessResponse(SipServletResponse res);
```

```
protected void doRedirectResponse(SipServletResponse res);
```

```
protected void doErrorResponse(SipServletResponse res);
```

These doXxx response methods are not the method name of the request. They are named by the type of the response as follows:

- doProvisionalResponse—A method invoked on the receipt of a provisional response (or 1xx response).
- doSuccessResponse—A method invoked on the receipt of a success response.
- doRedirectResponse—A method invoked on the receipt of a redirect response.

- doErrorResponse—A method invoked on the receipt of an error response (or 4xx, 5xx, 6xx responses).

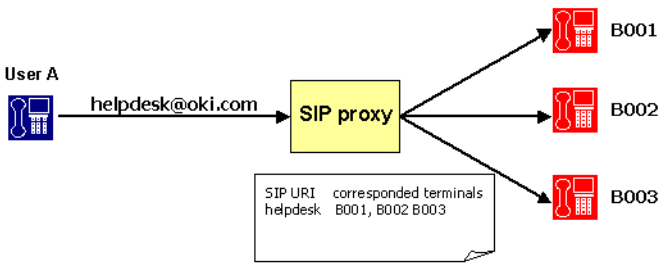
Existence of methods to receive responses indicates that in SIP Servlet requests and responses are independently transmitted an application in different threads. Applications must explicitly manage association of SIP messages. An independent request and response makes the process slightly complicated, but enables you to write more flexible processes.

Also, SIP Servlet allows applications to explicitly create requests. Using these functions, SIP servlets can not only wait for requests as a server (UAS), but also send requests as a client (UAC).

Proxy Functions

Another function that is different from the HTTP protocol is "forking." Forking is a process of proxying one request to multiple servers simultaneously (or sequentially) and used when multiple terminals (operators) are associated with one telephone number (such as in a call center).

Figure 1-4 Proxy Forking

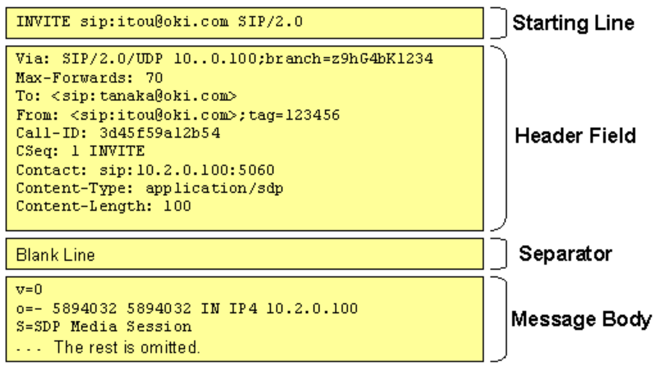


SIP Servlet provides a utility to proxy SIP requests for applications that have proxy functions.

Message Body

As the figure below, the structure of SIP messages is the same as HTTP.

Figure 1-5 SIP Message Example



```
INVITE sip:itou@oki.com SIP/2.0
Via: SIP/2.0/UDP 10..0.100;branch=z9hG4bK1234
Max-Forwards: 70
To: <sip:tanaka@oki.com>
From: <sip:itou@oki.com>;tag=123456
Call-ID: 3d45f59a12b54
CSeq: 1 INVITE
Contact: sip:10.2.0.100:5060
Content-Type: application/sdp
Content-Length: 100
Blank Line
v=0
o=- 5894032 5894032 IN IP4 10.2.0.100
S=SDP Media Session
... The rest is omitted.
```

HTTP is basically a protocol to transfer HTML files and images. Contents to be transferred are stored in the message body. HTTP Servlet defines stream manipulation-based API to enable sending and receiving massive contents.

ServletRequest

```
ServletInputStream getInputStream()

BufferedReader     getReader()
```

ServletResponse

```
ServletOutputStream getOutputStream()

PrintWriter         getWriter()

int  getBufferSize()

void setBufferSize(int size)

void resetBuffer()

void flushBuffer()
```

In SIP, however, only low-volume contents are stored in the message body since SIP is intended for real-time communication. Therefore, above methods are provided only for compatibility, and their functions are disabled.

In SIP, contents stored in the body include:

- SDP (Session Description Protocol)—A protocol to define multimedia sessions used between terminals. This protocol is defined in RFC2373.
- Presence Information—A message that describes presence information defined in CPIM.
- IM Messages—IM (instant message) body. User-input messages are stored in the message body.

Since the message body is in a small size, processing it in a streaming way increases overhead. SIP Servlet re-defines API to manipulate the message body on memory as follows:

SipServletMessage

```
void   setContent(Object content, String contentType)

Object getContent()

byte[] getRawContent()
```

Roles of a Servlet Container

The following sections describes major functions provided by Oracle Communications Converged Application Server as a SIP servlet container:

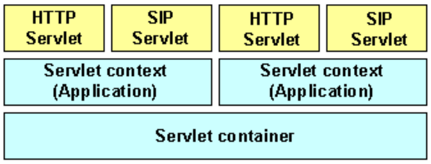
- Application Management—Describes functions such as application management by servlet context, lifecycle management of servlets, application initialization by deployment descriptors.
- SIP Messaging—Describes functions of parsing incoming SIP messages and delivering appropriate SIP servlets, sending messages created by SIP servlets to appropriate UAS, and automatically setting SIP header fields.
- Utility Functions—Describes functions such as sessions, factories, and proxying that are available in SIP servlets.

Application Management

Like HTTP servlet containers, SIP servlet containers manage applications by servlet context (see Figure 6). Servlet contexts (applications) are normally archived in a WAR format and deployed in each application server.

Note: The method of deploying in application servers varies depending on your product. Refer to the documentation of your application server.

Figure 1-6 Servlet Container and Servlet Context



A servlet context for a converged SIP and Web application can include multiple SIP servlets, HTTP servlets, and JSPs.

Oracle Communications Converged Application Server can deploy applications using the same method as the application server you use as the platform. However, if you deploy applications including SIP servlets, you need a SIP specific deployment descriptor (sip.xml) defined by SIP servlets. The table below shows the file structure of a general converged SIP and Web application.

Table 1-1 File Structure Example of Application

File	Description
WEB-INF/	Place your configuration and executable files of your converged SIP and Web application in the directory. You cannot directly refer to files in this directory on Web (servlets can do this).
WEB-INF/web.xml	The Java EE standard configuration file for the Web application.
WEB-INF/sip.xml	The SIP Servlet-defined configuration files for the SIP application.
WEB-INF/classes/	Store compiled class files in the directory. You can store both HTTP and SIP servlets in this directory.
WEB-INF/lib/	Store class files archived as Jar files in the directory. You can store both HTTP and SIP servlets in this directory.
*.jsp, *.jpg	Files comprising the Web application (e.g. JSP) can be deployed in the same way as Java EE.

Information specified in the sip.xml file is similar to that in the web.xml except <servlet-mapping> setting that is different from HTTP servlets. In HTTP you specify a servlet associated with the file name portion of URL. But SIP has no concept of the file name. You set filter conditions using URI or the header field of a SIP request. The following example shows that a SIP servlet called “register” is assigned all REGISTER methods.

Listing 1-2 List 1: Filter Condition Example of sip.xml

```
<servlet-mapping>
  <servlet-name>registrar</servlet-name>
  <pattern>
    <equal>
      <var>request.method</var>
      <value>REGISTER</value>
    </equal>
  </pattern>
</servlet-mapping>
```

Once deployed, lifecycle of the servlet context is maintained by the servlet container. Although the servlet context is normally started and shutdown when the server is started and shutdown, the system administrator can explicitly start, stop, and reload the servlet context.

SIP Messaging

SIP messaging functions provided by a SIP servlet container are classified under the following types:

- Parsing received SIP messages.
- Delivering parsed messages to the appropriate SIP servlet.
- Sending SIP servlet-generated messages to the appropriate UA
- Automatically generating a response (such as “100 Trying”).
- Automatically managing the SIP header field.

All SIP messages that a SIP servlet handles are represented as a SipServletRequest or SipServletResponse object. A received message is first parsed by the parser and then translated to one of these objects and sent to the SIP servlet container.

A SIP servlet container receives the following three types of SIP messages, for each of which you determine a target servlet.

- First SIP Request—When the SIP servlet container received a request that does not belong to any SIP session, it uses filter conditions in the sip.xml file (described in the previous section) to determine the target SIP servlet. Since the container creates a new SIP session when the initial request is delivered, any SIP requests received after that point are considered as subsequent requests.

  Note: Filtering should be done carefully. In Oracle Communications Converged Application Server, when the received SIP message matches multiple SIP servlets, it is delivered only to any one SIP servlet.

- Subsequent SIP Request—When the SIP Servlet container receives a request that belongs to any SIP session, it delivers the request to a SIP Servlet associated with that session. Whether the request belongs to a session or not is determined using dialog ID.

  Each time a SIP Servlet processes messages, a lock is established by the container on the call ID. If a SIP Servlet is currently processing earlier requests for the same call ID when subsequent requests are received, the SIP Servlet container queues the subsequent requests. The queued messages are processed only after the Servlet has finished processing the initial message and has returned control to the SIP Servlet container.

  This concurrency control is guaranteed both in a single containers and in clustered environments. Application developers can code applications with the understanding that only one message for any particular call ID will be processed at a given time.

- SIP Response—When the received response is to a request that a SIP servlet proxied, the response is automatically delivered to the same servlet since its SIP session had been determined. When a SIP servlet sends its own request, you must first specify a servlet that receives a response in the SIP session. For example, if the SIP servlet sending a request also receives the response, the following handler setting must be specified in the SIP session.

```
SipServletRequest req = getSipFactory().createRequest(appSession, ...);

req.getSession().setHandler(getServletName());
```

Normally, in SIP a “session” means a real-time session by RTP/RTSP. On the other hand, in HTTP Servlet a “session” refers to a way of relating multiple HTTP transactions. In this document, session-related terms are defined as follows:

Table 1-2 Session-Related Terminology

Realtime Session	A realtime session established by RTP/RTSP.
HTTP Session	A session defined by HTTP Servlet. A means of relating multiple HTTP transactions.
SIP Session	A means of implementing the same concept as in HTTP session in SIP. SIP (RFC3261) has a similar concept of “dialog,” but in this document this is treated as a different term since its lifecycle and generation conditions are different.
Application Session	A means for applications using multiple protocols and dialogs to associate multiple HTTP sessions and SIP sessions. Also called “AP session.”

Oracle Communications Converged Application Server automatically execute the following response and retransmission processes:

- Sending “100 Trying”—When WebLogic Communications Server receives an INVITE request, it automatically creates and sends “100 Trying.”
- Response to CANCEL—When WebLogic Communications Server receives a CANCEL request, it executes the following processes if the request is valid.
  a. Sends a 200 response to the CANCEL request.
  b. Sends a 487 response to the INVITE request to be cancelled.
  c. Invokes a doCancel method on the SIP servlet. This allows the application to abort the process within the doCancel method, eliminating the need for explicitly sending back a response.
- Sends ACK to an error response to INVITE—When a 4xx, 5xx, or 6xx response is returned for INVITE that were sent by a SIP servlet, WebLogic Communications Server automatically creates and sends ACK. This is because ACK is required only for a SIP sequence, and the SIP servlet does not require it.

  When the SIP servlet sends a 4xx, 5xx, or 6xx response to INVITE, it never receives ACK for the response.

- Retransmission process when using UDP—SIP defines that sent messages are retransmitted when low-trust transport including UDP is used. WebLogic Communications Server automatically do the retransmission process according to the specification.

Mostly, applications do not need to explicitly set and see header fields In HTTP Servlet since HTTP servlet containers automatically manage these fields such as Content-Length and Content-Type. SIP Servlet also has the same header management function.

In SIP, however, since important information about message delivery exists in some fields, these headers are not allowed to change by applications. Headers that can not be changed by SIP servlets are called "system headers." The table below lists system headers:

Table 1-3 System Headers

Header Name	Description
Call-ID	Contains ID information to associate multiple SIP messages as Call.
From, To	Contains Information on the sender and receiver of the SIP request (SIP, URI, etc.). tag parameters are given by the servlet container.
CSeq	Contains sequence numbers and method names.
Via	Contains a list of servers the SIP message passed through. This is used when you want to keep track of the pass to send a response to the request.
Record-Route, Route	Used when the proxy server mediates subsequent requests.
Contact	Contains network information (such as IP address and port number) that is used for direct communication between terminals. For a REGISTER message, 3xx, or 485 response, this is not considered as the system header and SIP servlets can directly edit the information.

Utility Functions

SIP Servlet defines the following utilities that are available to SIP servlets:

1. SIP Session, Application Session
2. SIP Factory
3. Proxy

SIP Session, Application Session

As stated before, SIP Servlet provides a “SIP session” whose concept is the same as a HTTP session. In HTTP, multiple transactions are associated using information like Cookie. In SIP, this association is done with header information (Call-ID and tag parameters in From and To). Servlet containers maintain and manage SIP sessions. Messages within the same dialog can refer to the same SIP session. Also, For a method that does not create a dialog (such as MESSAGE), messages can be managed as a session if they have the same header information.

SIP Servlet has a concept of an “application session,” which does not exist in HTTP Servlet. An application session is an object to associate and manage multiple SIP sessions and HTTP sessions. It is suitable for applications such as B2BUA.

SIP Factory

A SIP factory (SipFactory) is a factory class to create SIP Servlet-specific objects necessary for application execution. You can generate the following objects:

Table 1-4 Objects Generated with SipFactory

Class Name	Description
URI, SipURI, Address	Can generate address information including SIP URI from String.
SipApplicationSession	Creates a new application session. It is invoked when a SIP servlet starts a new SIP signal process.
SipServletRequest	Used when a SIP servlet acts as UAC to create a request. Such requests can not be sent with Proxy.proxyTo. They must be sent with SipServletRequest.send.

SipFactory is located in the servlet context attribute under the default name. You can take this with the following code.

```
ServletContext context = getServletContext();

SipFactory factory =

    (SipFactory) context.getAttribute("javax.servlet.sip.SipFactory");
```

Proxy

Proxy is a utility used by a SIP servlet to proxy a request. In SIP, proxying has its own sequences including forking. You can specify the following settings in proxying with Proxy:

- Recursive routing (recurse)—When the destination of proxying returns a 3xx response, the request is proxied to the specified target.
- Record-Route setting—Sets a <code>Record-Route</code> header in the specified request.
- Parallel/Sequential (parallel)—Determines whether forking is executed in parallel or sequentially.
- stateful—Determines whether proxying is transaction stateful.
- Supervising mode—In the event of the state change of proxying (response receipts), an application reports this.

Porting Existing Applications to Oracle Communications Converged Application Server

The following sections describe guidelines and issues related to porting existing applications based on the SIP Servlet v1.0 specification to Oracle Communications Converged Application Server and the SIP Servlet v1.1 specification:

- “Application Router and Legacy Application Composition” on page 2-2
- “SipSession and SipApplicationSession Not Serializable” on page 2-2
- “SipServletResponse.setCharacterEncoding() API Change” on page 2-2
- “Transactional Restrictions for SipServletRequest and SipServletResponse” on page 2-3
- “Immutable Parameters for New Parameterable Interface” on page 2-3
- “Stateless Transaction Proxies Deprecated” on page 2-4
- “Backward-Compatibility Mode for v1.0 Deployments” on page 2-4
- “Deprecated APIs” on page 2-6
- “SNMP MIB Changes” on page 2-6
- “Renamed Diagnostic Monitors and Actions” on page 2-7

Application Router and Legacy Application Composition

The SIP Servlet v1.1 specification describes a formal application selection and composition process, which is fully implemented in Oracle Communications Converged Application Server. Use the SIP Servlet v1.1 techniques, as described in "Porting Existing Applications to Oracle Communications Converged Application Server" on page 2-1, for all new development. Application composition techniques described in earlier versions of WebLogic SIP Server are now deprecated.

Oracle Communications Converged Application Server provides backwards compatibility for applications using version SIP Servlet 1.0 composition techniques, provided that:

- you *do not* configure a custom Application Router, and
- you *do not* configure Default Application Router properties.

SipSession and SipApplicationSession Not Serializable

The `SipSession` and `SipApplicationSession` interfaces are no longer serializable in the SIP Servlet v1.1 specification. Oracle Communications Converged Application Server maintains binary compatibility for the earlier v1.0 specification, so any compiled applications that treat these interfaces as serializable objects will continue to work. However, you must modify the source code of such applications before you can recompile them with Oracle Communications Converged Application Server.

Version 1.0 Servlets that stored the `SipSession` as a serializable info object using the `TimerService.createTimer` API can achieve similar functionality by storing the `SipSession` ID as the serializable `info` object. On receiving the timer expiration callback, applications should use the `SipApplicationSession` and the serialized ID object returned by the `ServletTimer` to find the `SipSession` within the `SipApplicationSession` using the retrieved ID. See the SIP Servlet v1.1 API JavaDoc for more information.

SipServletResponse.setCharacterEncoding() API Change

`SipServletResponse.setCharacterEncoding()` no longer throws `UnsupportedEncodingException`. If you have an application that explicitly catches `UnsupportedEncodingException` with this method, the existing, compiled application can be deployed to Oracle Communications Converged Application Server unchanged. However, the source code must be modified to not catch the exception before you can recompile.

Transactional Restrictions for SipServletRequest and SipServletResponse

SIP Servlet v1.1 acknowledges that `SipServletRequest` and `SipServletResponse` objects always belong to a SIP transaction. The specification further defines the conditions for committing a message, after which no application can modify or re-send the message. See *5.2 Implicit Transaction State* in the SIP Servlet Specification v1.1 for a list of conditions that commit SIP messages.

As a result of this change, any attempt to modify (set, add, or remove a header) or send a committed message now results in an `IllegalStateException`. Ensure that any existing code checks for the committed status of a message using `SipServletMessage.isCommitted()` before modifying or sending a message.

Immutable Parameters for New Parameterable Interface

SIP Servlet v1.1 introduces a new `javax.servlet.sip.Parameterable` interface for accessing, creating, and modifying parameters in various SIP headers. Note that the system header parameters described in Table 2-1 are immutable and cannot be modified using this new interface.

Table 2-1 Immutable System Header Parameters

Header	Immutable Parameters
Contact	&nbsp;
From	tag
To	tag
Via	branch, received, rport, wlsslport, wlssladdr, maddr, ttl
Record-Route	All parameters are immutable.
Route	For initial requests, the application that pushes the Route header can modify any of the header’s parameters. In all other cases, the parameters of the Route header are immutable.
Path	For Register requests, the application that pushes the Path header can modify any of the header’s parameters.In all other cases, the parameters of the Path header are immutable.

Stateless Transaction Proxies Deprecated

The SIP Servlet v1.1 specification requires that a Proxy is always transactionally stateful. Stateless proxying is supported only for v1.0 applications.

For v1.1 applications in Oracle Communications Converged Application Server, the `Proxy.setStateful()` and `Proxy.getStateful()` methods are redundant: `Proxy.getStateful()` always returns true, and `Proxy.setStateful()` performs no operation.

Backward-Compatibility Mode for v1.0 Deployments

Oracle Communications Converged Application Server automatically detects precompiled, v1.0 deployments and alters the SIP container behavior to maintain backward compatibility. The sections that follow describe differences in behavior that occur when deploying v1.0 SIP Servlets to Oracle Communications Converged Application Server.

Validation Warnings for v1.0 Servlet Deployments

The SIP Servlet v1.1 specification requires more strict validation of Servlet deployments than the previous specification. In the following cases, v1.0 SIP Servlets can be successfully deployed to Oracle Communications Converged Application Server, but a warning message will be displayed at deployment:

- If a listener is declared in the `listener-class` element of a v1.0 deployment descriptor but the corresponding class does not implement the `EventListener` interface, a warning is displayed during deployment. (Version 1.1 SIP Servlets that declare a listener *must* implement `EventListener`, or the application cannot be deployed).
- If a SIP Servlet is declared in the `servlet-class` element of a v1.0 deployment descriptor, but the corresponding class does extend the `SipServlet` abstract class, a warning is displayed. (Version 1.1 SIP Servlets *must* extend `SipServlet`, or the application cannot be deployed).

Modifying Committed Messages

The SIP Servlet v1.1 specification now recommends that the SIP container throw an `IllegalStateException` if an application attempts to modify a committed message. To maintain backward compatibility, Oracle Communications Converged Application Server throws the `IllegalStateException` only when a version 1.1 SIP Servlet deployment modifies a committed message.

Path Header as System Header

The SIP Servlet v1.1 specification now defines the `Path` header as a system header, which cannot be modified by an application. Version 1.0 SIP Servlets can still modify the `Path` header, but a warning message is generated. Version 1.1 SIP Servlets that attempt to modify the `Path` header fail with an `IllegalArgumentException`.

SipServletResponse.createPrack() Exception

In Oracle Communications Converged Application Server, `SipServletResponse.createPrack()` can throw `Rel100Exception` only for version 1.1 SIP Servlets. `createPrack()` does not throw the exception for version 1.0 SIP Servlets to maintain backward compatibility.

Proxy.proxyTo() Exceptions

For version 1.1 SIP Servlets, Oracle Communications Converged Application Server throws an `IllegalStateException` if a version 1.1 SIP Servlet specifies a duplicate branch URI with `Proxy.proxyTo(uri)` or `Proxy.proxyTo(uris)`. To maintain backward compatibility, Oracle Communications Converged Application Server ignores the duplicate URIs (and throws no exception) if a version 1.0 SIP Servlet specifies duplicate URIs with these methods.

Changes to Proxy Branch Timers

SIP Servlet v1.1 makes several protocol changes that effect the behavior of proxy branching for both sequential and parallel proxying.

For sequential proxying, the v1.1 specification requires that Oracle Communications Converged Application Server start a branch timer using the maximum of the `sequential-search-timeout` value, which is configured in `sip.xml`, or SIP protocol Timer C (> 3 minutes). Prior versions of Oracle Communications Converged Application Server always set sequential branch proxy timeouts using the value of `sequential-search-timeout`; this behavior is maintained for v1.0 deployments.

For parallel proxying, the v1.1 specification provides a new `proxyTimeout` value that controls proxying. The specification requires that Oracle Communications Converged Application Server reset a branch timer using the configured `proxyTimeout` value, rather than using the Timer C value as required in the SIP Servlet v1.0 specification. The Timer C value is still used for v1.0 deployments.

Deprecated APIs

Earlier versions of WebLogic SIP Server provided proprietary APIs to support functionality and RFCs that were not supported in the SIP Servlet v1.0 specification. The SIP Servlet v1.1 specification adds new RFC support and functionality, making the proprietary APIs redundant. Table 2-2 shows newly-available SIP Servlet v1.1 methods that should be used in place of now-deprecated WebLogic SIP Server methods. The deprecated methods are still available in this release to provide backward compatibility for v1.0 applications.

Table 2-2 Deprecated APIs

Deprecated Methods (WebLogic SIP Server Proprietary)	Replacement Method (SIP Servlet v1.1)
`WlssSipServlet.doRefer()`, `WlssSipServlet.doUpdate()`, `WlssSipServlet.doPrack()`	`SipServlet.doRefer()`, `SipServlet.doUpdate()`, `SipServlet.doPrack()`
`WlssSipServletResponse.createPrack()`	`SipServletResponse.createPrack()`
`WlssProxy.getAddToPath()`, `WlssProxy.setAddToPath()`	`Proxy.getAddToPath()`, `Proxy.setAddToPath()`
`WlssSipServletMessage.setHeaderForm()`, `WlssSipServletMessage.getHeaderForm()`	`SipServletMessage.setHeaderForm()`, `SipServletMessage.getHeaderForm()`
`com.bea.wcp.util.Sessions`	See Table 6-1, “Deprecated com.bea.wcp.util.Sessions Methods,” on page 6-4.

SNMP MIB Changes

Previous versions of the Oracle Communications Converged Application Server SNMP MIB definition did not follow the WebLogic MIB naming convention. Specifically, the MIB table column name label did not begin with the table name. Oracle Communications Converged Application Server changes the SNMP MIB definition to prepend labels with `sipServer` in order to comply with the WebLogic naming convention and provide compatibility with WebLogic tools that generate the metadata file.

For example, in version 3.x the `SipServerEntry` MIB definition was:

```
SipServerEntry  ::=  SEQUENCE {
sipServerIndex  DisplayString,
t1TimeoutInterval  INTEGER,
t2TimeoutInterval  INTEGER,
t4TimeoutInterval  INTEGER,
....
}
```

In Oracle Communications Converged Application Server, the definition is now:

```
SipServerEntry  ::=  SEQUENCE {
sipServerIndex  DisplayString,
sipServerT1TimeoutInterval  Counter64,
sipServerT2TimeoutInterval  INTEGER,
sipServerT4TimeoutInterval  INTEGER,
.....
}
```

This change in the MIB may cause backwards compatibility issues if an application or script uses the MIB table column name labels directly. All hard-coded labels, such as `iso.org.dod.internet.private.enterprises.bea.wlss.sipServerTable.t1TimeoutInterval` must be changed to prepend the table name (`iso.org.dod.internet.private.enterprises.bea.wlss.sipServerTable.sipServerT1TimeoutInterval`).

Notes: Client-side SNMP tools generally load a MIB and issue commands to retrieve values based on the loaded MIB labels. These tools are unaffected by the above change.

The complete Oracle Communications Converged Application Server MIB file is installed as `$WLSS_HOME/server/lib/wlss/BEA-WLSS-MIB.asn1`.

Renamed Diagnostic Monitors and Actions

The diagnostic monitors and diagnostic actions provided in Oracle Communications Converged Application Server are now prefixed with `occas/`. For example, the SIP Server 3.1 `Sip_Servlet_Before_Service` monitor is now named `occas/Sip_Servlet_Before_Service`. You must update any existing diagnostic configuration files or applications that reference the non-prefixed names before they can work with Oracle Communications Converged Application Server.

See Using the WebLogic Server Diagnostic Framework (WLDF) in the *Operations Guide*.

Requirements and Best Practices for SIP Applications

The following sections describe requirements and best practices for developing applications for deployment to Oracle Communications Converged Application Server:

- "Overview of Developing Distributed Applications for Oracle Communications Converged Application Server" on page 3-2
- "Applications Must Not Create Threads" on page 3-2
- "Servlets Must Be Non-Blocking" on page 3-3
- "Store all Application Data in the Session" on page 3-3
- "All Session Data Must Be Serializable" on page 3-3
- "Use setAttribute() to Modify Session Data in "No-Call" Scope" on page 3-4
- "send() Calls Are Buffered" on page 3-6
- "Mark SIP Servlets as Distributable" on page 3-7
- "Use SipApplicationSessionActivationListener Sparingly" on page 3-7
- "Observe Best Practices for Java EE Applications" on page 3-7

Overview of Developing Distributed Applications for Oracle Communications Converged Application Server

In a typical production environment, SIP applications are deployed to a cluster of Oracle Communications Converged Application Server instances that form the engine tier cluster. A separate cluster of servers in the SIP data tier provides a replicated, in-memory database of the call states for active calls. In order for applications to function reliably in this environment, you must observe the programming practices and conventions described in the sections that follow to ensure that multiple deployed copies of your application perform as expected in the clustered environment.

If you are porting an application from a previous version of Oracle Communications Converged Application Server, many of the conventions and restrictions described below may be new to you, because previous Oracle Communications Converged Application Server implementations did not support a clustering. As always, thoroughly test and profile your ported applications to discover problems and ensure adequate performance in the new environment.

Applications Must Not Create Threads

Oracle Communications Converged Application Server is a multi-threaded application server that carefully manages resource allocation, concurrency, and thread synchronization for the modules it hosts. To obtain the greatest advantage from the Oracle Communications Converged Application Server architecture, construct your application modules according to the SIP Servlet and Java EE API specifications.

Avoid application designs that require creating new threads in server-side modules such as SIP Servlets:

- The SIP Servlet container automatically locks the associated call state when invoking the `do`*`xxx`* method of a SIP Servlet. If the `do`*`xxx`* method spawns additional threads or accesses a different call state before returning control, *deadlock scenarios and lost updates to session data can occur.*
- Applications that create their own threads do not scale well. Threads in the JVM are a limited resource that must be allocated thoughtfully. Your applications may break or cause poor Oracle Communications Converged Application Server performance when the server load increases. Problems such as deadlocks and thread starvation may not appear until the application is under a heavy load.

- Multithreaded modules are complex and difficult to debug. Interactions between application-generated threads and WebLogic Server threads are especially difficult to anticipate and analyze.
- The `WlssSipApplicationSession.doAction()` method, described in "Use setAttribute() to Modify Session Data in "No-Call" Scope" on page 3-4, does not provide synchronization for spawned Java threads. Any threads created within `doAction()` can execute another `doAction()` on the same `WlssSipApplicationSession`. Similarly, main threads that use `doAction()` to access a different `wlssSipApplicationSession` can lead to deadlocks, because the container automatically locks main threads when processing incoming SIP messages. "Use setAttribute() to Modify Session Data in "No-Call" Scope" on page 3-4 describes a potential deadlock situation.

WARNING: If your application must spawn threads, you must guard against deadlocks and carefully manage concurrent access to session data. At a minimum, never spawn threads inside the service method of a SIP Servlet. Instead, maintain a separate thread pool outside of the service method, and be careful to synchronize access to all session data.

Servlets Must Be Non-Blocking

SIP and HTTP Servlets must not block threads in the body of a SIP method because the call state remains locked while the method is invoked. For example, no Servlet method should actively wait for data to be retrieved or written before returning control to the SIP Servlet container.

Store all Application Data in the Session

If you deploy your application to more than one engine tier server (in a replicated Oracle Communications Converged Application Server configuration) you must store all application data in the session as session attributes. In a replicated configuration, engine tier servers maintain no cached information; all application data must be de-serialized from the session attribute available in SIP data tier servers.

All Session Data Must Be Serializable

To support in-memory replication of SIP application call states, you must ensure that all objects stored in the SIP Servlet session are serializable. Every field in an object must be serializable or transient in order for the object to be considered serializable. If the Servlet uses a combination of serializable and non-serializable objects, Oracle Communications Converged Application Server cannot replicate the session state of the non-serializable objects.

Use setAttribute() to Modify Session Data in “No-Call” Scope

The SIP Servlet container automatically locks the associated call state when invoking the `do`*`xxx`* method of a SIP Servlet. However, applications may also attempt to modify session data in “no-call” scope. No-call scope refers to the context where call state data is modified outside the scope of a normal `do`*`xxx`* method. For example, data is modified in no-call scope when an HTTP Servlet attempts to modify SIP session data, or when a SIP Servlet attempts to modify a call state other than the one that the container locked before invoking the Servlet.

Applications must always use the SIP Session’s `setAttribute` method to change attributes in no-call scope. Likewise, use `removeAttribute` to remove an attribute from a session object. Each time `setAttribute/removeAttribute` is used to update session data, the SIP Servlet container obtains and releases a lock on the associated call state. (The methods enqueue the object for updating, and return control immediately.) This ensures that only one application modifies the data at a time, and also ensures that your changes are replicated across SIP data tier nodes in a cluster.

If you use other set methods to change objects within a session, Oracle Communications Converged Application Server cannot replicate those changes.

Note that the Oracle Communications Converged Application Server container does not persist changes to a call state attribute that are made *after* calling `setAttribute`. For example, in the following code sample the `setAttribute` call immediately modifies the call state, but the subsequent call to `modifyState()` does not:

```
Foo foo = new Foo(..);

appSession.setAttribute("name", foo); // This persists the call state.

foo.modifyState(); // This change is not persisted.
```

Instead, ensure that your Servlet code modifies the call state attribute value *before* calling `setAttribute`, as in:

```
Foo foo = new Foo(..);

foo.modifyState();

appSession.setAttribute("name", foo);
```

Also, keep in mind that the SIP Servlet container obtains a lock to the call state for *each* individual `setAttribute` call. For example, when executing the following code in an HTTP Servlet, the SIP Servlet container obtains and releases a lock on the call state lock twice:

```
appSess.setAttribute("foo1", "bar2");

appSess.setAttribute("foo2", "bar2");
```

This locking behavior ensures that only one thread modifies a call state at any given time. However, another process could potentially modify the call state between sequential updates. The following code is not considered thread safe when done no-call state:

```
Integer oldValue = appSession.getAttribute("counter");

Integer newValue = incrementCounter(oldValue);

appSession.setAttribute("counter", newValue);
```

To make the above code thread safe, you must enclose it using the `wlssAppSession.doAction` method, which ensures that all modifications made to the call state are performed within a single transaction lock, as in:

```
wlssAppSession.doAction(new WlssAction() {

       public Object run() throws Exception {

         Integer oldValue = appSession.getAttribute("counter");

         Integer newValue = incrementCounter(oldValue);

         appSession.setAttribute("counter", newValue);

         return null;

       }

     });
```

Finally, be careful to avoid deadlock situations when locking call states in a “`do`*`SipMethod`*” call, such as `doInvite()`. Keep in mind that the Oracle Communications Converged Application Server container has already locked the call state when the instructions of a `do`*`SipMethod`* are executed. If your application code attempts to access the current call state from within such a method (for example, by accessing a session that is stored within a data structure or attribute), the lock ordering results in a deadlock.

Listing 3-1 shows an example that can result in a deadlock. If the code is executed by the container for a call associated with `callAppSession`, the locking order is reversed and the attempt to obtain the session with `getApplicationSession(callId)` causes a deadlock.

Listing 3-1 Session Access Resulting in a Deadlock

```
WlssSipApplicationSession confAppSession = (WlssSipApplicationSession)
appSession;

confAppSession.doAction(new WlssAction() {

  // confAppSession is locked

  public Object run() throws Exception {

    String callIds = confAppSession.getAttribute("callIds");

    for (each callId in callIds) {

      callAppSess = Session.getApplicationSession(callId);

      // callAppSession is locked

      attributeStr += callAppSess.getAttribute("someattrib");

    }

    confAppSession.setAttribute("attrib", attributeStr);

  }

}
```

See "Modifying the SipApplicationSession" on page 6-5 for more information about using the `com.bea.wcp.sip.WlssAction` interface.

send() Calls Are Buffered

If your SIP Servlet calls the `send()` method within a SIP request method such as `doInvite()`, `doAck()`, `doNotify()`, and so forth, keep in mind that the Oracle Communications Converged Application Server container buffers all `send()` calls and transmits them in order *after* the SIP method returns. Applications cannot rely on `send()` calls to be transmitted immediately as they are called.

WARNING: Applications must not wait or sleep after a call to `send()`, because the request or response is not transmitted until control returns to the SIP Servlet container.

Mark SIP Servlets as Distributable

If you have designed and programmed your SIP Servlet to be deployed to a cluster environment, you must include the `distributable` marker element in the Servlet's deployment descriptor when deploying the application to a cluster of engine tier servers. If you omit the `distributable` element, Oracle Communications Converged Application Server will not deploy the Servlet to a cluster of engine tier servers.

The `distributable` element is not required, and is ignored if you deploy to a single, combined-tier (non-replicated) Oracle Communications Converged Application Server instance.

Use SipApplicationSessionActivationListener Sparingly

The SIP Servlet 1.1 specification introduces `SipApplicationSessionActivationListener`, which can provide callbacks to an application when SIP Sessions are passivated or activated. Keep in mind that callbacks occur only in a replicated Oracle Communications Converged Application Server deployment. Single-server deployments use no SIP data tier, so SIP Sessions are never passivated.

Also, keep in mind that in a replicated deployment Oracle Communications Converged Application Server activates and passivates a SIP Session many times, before and after SIP messages are processed for the session. (This occurs normally in any replicated deployment, even when RDBMS-based persistence is not configured.) Because this constant cycle of activation and passivation results in frequent callbacks, use `SipApplicationSessionActivationListener` sparingly in your applications.

Observe Best Practices for Java EE Applications

If you are deploying applications that use other Java EE APIs, observe the basic clustering guidelines associated with those APIs. For example, if you are deploying EJBs you should design all methods to be idempotent and make EJB homes clusterable in the deployment descriptor. See Clustering Best Practices in the Oracle WebLogic Server 10*g* Release 3 Documentation for more information.

CHAPTER 4

Using Compact and Long Header Formats for SIP Messages

The following sections describe how to use the Oracle Communications Converged Application Server `WlssSipServletMessage` interface and configuration parameters to control SIP message header formats:

- "Overview of Header Format APIs and Configuration" on page 4-1
- "Summary of Compact Headers" on page 4-2
- "Assigning Header Formats with WlssSipServletMessage" on page 4-2
- "Summary of API and Configuration Behavior" on page 4-3

Overview of Header Format APIs and Configuration

Applications that operate on wireless networks may want to limit the size of SIP headers to reduce the size of messages and conserve bandwidth. JSR 116 provides the `SipServletMessage.setHeader` method, which enables application developers to set a given header to a specific string value, such as the compact (1-letter) format.

Oracle Communications Converged Application Server extends the `SipServletMessage` interface with `WlssSipServletMessage`. One feature of the `WlssSipServletMessage` API is the ability to set long or compact header formats for the entire SIP message using the `setUseHeaderForm` method.

In addition to `WlssSipServletMessage`, Oracle Communications Converged Application Server provides a container-wide configuration parameter that can control SIP header formats for all system-generated headers. This system-wide parameter can be used along with

`WlssSipServletMessage.setUseHeaderForm` and `SipServletMessage.setHeader` to further customize header formats.

Summary of Compact Headers

Table 4-1 defines the compact header abbreviations described in the SIP specification (RFC3261). Specifications that introduce additional headers may also include compact header abbreviations.

Table 4-1 Compact Header Abbreviations

Header Name (Long Format)	Compact Format
Call-ID	i
Contact	m
Content-Encoding	e
Content-Length	l
Content-Type	c
From	f
Subject	s
Supported	k
To	t
Via	v

Assigning Header Formats with WlssSipServletMessage

All instances of `SipServletRequest`, `SipServletResponse`, and `WlssSipServletResponse` can be cast to `WlssSipServletMessage` in order to use the extended API. A pair of getter/setter methods, `setUseHeaderForm` and `getUseHeaderForm`, are used to assign or retrieve the header formats used in the message. These methods assign or return a `HeaderForm` object, which is a simple Enumeration that describes the header format:

- `COMPACT`—Forces all headers in the message to use compact format. This behavior is similar to the container-wide configuration value of "force compact," as described in `use-compact-form` in the *Configuration Reference Manual*.

- `LONG`—Forces all headers in the message to use long format. This behavior is similar to the container-wide configuration value of "force long," as described in `use-compact-form` in the *Configuration Reference Manual*.
- `DEFAULT`—Defers the header format to the container-wide configuration value set in `use-compact-form`.

`WlssSipServletResponse.setUseHeaderForm` can be used in combination with `SipServletMessage.setHeader` and the container-level configuration parameter, `use-compact-form.` "Summary of API and Configuration Behavior" on page 4-3.

Summary of API and Configuration Behavior

Header formats can be specified at the header, message, and SIP Servlet container levels. Table 4-2 shows the header format that results when adding a new header with `SipServletMessage.setHeader`, given different container configurations and message-level settings with `WlssSipServletResponse.setUseHeaderForm.`

Table 4-2 API Behavior when Adding Headers

SIP Servlet Container Header Configuration (use-compact-form Setting)	WlssSipServletMessage. setUseHeaderForm Setting	SipServletMessage. setHeader Value	Resulting Header
COMPACT	DEFAULT	"Content-Type"	"Content-Type"
COMPACT	DEFAULT	"c"	"c"
COMPACT	COMPACT	"Content-Type"	"c"
COMPACT	COMPACT	"c"	"c"
COMPACT	LONG	"Content-Type"	"Content-Type"
COMPACT	LONG	"c"	"Content-Type"
LONG	DEFAULT	"Content-Type"	"Content-Type"
LONG	DEFAULT	"c"	"c"
LONG	COMPACT	"Content-Type"	"c"
LONG	COMPACT	"c"	"c"
LONG	LONG	"Content-Type"	"Content-Type"
LONG	LONG	"c"	"Content-Type"

Table 4-2 API Behavior when Adding Headers

FORCE_COMPACT	DEFAULT	“Content-Type”	“c”
FORCE_COMPACT	DEFAULT	“c”	“c”
FORCE_COMPACT	COMPACT	“Content-Type”	“c”
FORCE_COMPACT	COMPACT	“c”	“c”
FORCE_COMPACT	LONG	“Content-Type”	“Content-Type”
FORCE_COMPACT	LONG	“c”	“Content-Type”
FORCE_LONG	DEFAULT	“Content-Type”	“Content-Type”
FORCE_LONG	DEFAULT	“c”	“Content-Type”
FORCE_LONG	COMPACT	“Content-Type”	“c”
FORCE_LONG	COMPACT	“c”	“c”
FORCE_LONG	LONG	“Content-Type”	“Content-Type”
FORCE_LONG	LONG	“c”	“Content-Type”

Table 4-3 shows the system header format that results when setting the header format with `WlssSipServletResponse.setUseHeaderForm` given different container configuration values.

Table 4-3 API Behavior for System Headers

SIP Servlet Container Header Configuration (use-compact-form Setting)	WlssSipServletMessage. setUseHeaderForm Setting	Resulting Contact Header
COMPACT	DEFAULT	“m”
COMPACT	COMPACT	“m”
COMPACT	LONG	“Contact”
LONG	DEFAULT	“Contact”
LONG	COMPACT	“m”
LONG	LONG	“Contact”

Table 4-3 API Behavior for System Headers

FORCE_COMPACT	DEFAULT	“m”
FORCE_COMPACT	COMPACT	“m”
FORCE_COMPACT	LONG	“Contact”
FORCE_LONG	DEFAULT	“Contact”
FORCE_LONG	COMPACT	“m”
FORCE_LONG	LONG	“Contact”

Composing SIP Applications

The following sections describe how to use Oracle Communications Converged Application Server application composition features:

- "Application Composition Model" on page 5-1
- "Using the Default Application Router" on page 5-3
- "Configuring a Custom Application Router" on page 5-4
- "Session Key-Based Request Targeting" on page 5-6

Notes: The SIP Servlet v1.1 specification describes a formal application selection and composition process, which is fully implemented in Oracle Communications Converged Application Server. Use the SIP Servlet v1.1 techniques, as described in this document, for all new development. Application composition techniques described in earlier versions of Oracle Communications Converged Application Server are now deprecated.

Oracle Communications Converged Application Server provides backwards compatibility for applications using version 1.0 composition techniques, provided that:

- you *do not* configure a custom Application Router, and
- you *do not* configure Default Application Router properties.

Application Composition Model

Application composition is the process of "chaining" multiple SIP applications into a logical path to apply services to a SIP request. The SIP Servlet v1.1 specification introduces an Application

Router (AR) deployment, which performs a key role in composing SIP applications. The Application Router examines an initial SIP request and uses custom logic to determine which SIP application should process the request. In Oracle Communications Converged Application Server, all initial requests are first delivered to the AR, which determines the application used to process the request.

Oracle Communications Converged Application Server provides a default Application Router, which can be configured using a text file. However, most installations will develop and deploy a custom Application Router by implementing the `SipApplicationRouter` interface. A custom Application Router enables you to consult data stores when determining which SIP application should handle a request.

In contrast to the Application Router, which requires knowledge of which SIP applications are available for processing a message, individual SIP applications remain independent from one another. An individual application should perform a very specific service for a SIP request, without requiring any knowledge of other applications deployed on the system. (The Application Router does require knowledge of deployed applications, and the `SipApplicationRouter` interface provides for automatic notification of application deployment and undeployment.)

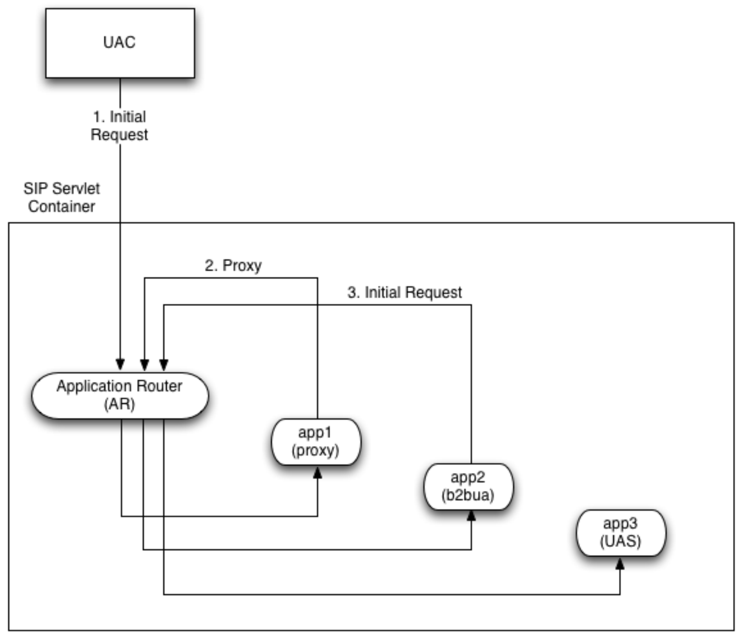
Individual SIP applications may complete their processing of an initial request by proxying or relaying the request, or by terminating the request as a User Agent Server (UAS). If an initial request is proxied or relayed, the SIP container again forwards the request to the Application Router, which selects the next SIP application to provide a service for the request. In this way, the AR can chain multiple SIP applications as needed to process a request. The chaining process is terminated when:

- a selected SIP application acts as a UAS to terminate the chain, or
- there are no more applications to select for that request. (In this case, the request is sent out.)

When the chain is terminated and the request sent, the SIP container maintains the established path of applications for processing subsequent requests, and the AR is no longer consulted.

Figure 5-1 shows the use of an Application Router for applying multiple service to a SIP request.

Figure 5-1 Composed Application Model



Note that the AR may select remote as well as local applications; the chain of services need not reside within the same Oracle Communications Converged Application Server container.

Using the Default Application Router

Oracle Communications Converged Application Server includes a Default Application Router (DAR) having the basic functionality described in the SIP Servlet Specification v1.1, Appendix C: Default Application Router. In summary, the Oracle Communications Converged Application Server DAR implements all methods of the `SipApplicationRouter` interface, and is configured using the simple Java properties file described in the v1.1 specification.

Each line of the DAR properties file specifies one or more SIP methods, and is followed by SIP routing information in comma-delimited format. The DAR initially reads the properties file on startup, and then reads it each time a SIP application is deployed or undeployed from the container.

To specify the location of the configuration file used by the DAR, configure the properties using the Administration Console, as described in "Configuring a Custom Application Router" on page 5-4, or include the following parameter when starting the Oracle Communications Converged Application Server instance:

```
-Djavax.servlet.sip.ar.dar.configuration
```

(To specify a property file, rather than a URI, include the prefix `file:///`) This Java parameter is specified at the command line, or it can be included in your server startup script.

See Appendix C in the SIP Servlet Specification v1.1 for detailed information about the format of routing information used by the Default Application Router.

Note that the Oracle Communications Converged Application Server DAR accepts route region strings in addition to "originating," "terminating," and "neutral." Each new string value is treated as an extended route region. Also, the Oracle Communications Converged Application Server DAR uses the order of properties in the configuration file to determine the route entry sequence; the `state_info` value has no effect when specified in the DAR configuration.

Configuring a Custom Application Router

By default Oracle Communications Converged Application Server uses its DAR implementation.

If you develop a custom Application Router, you must store the implementation for the AR in the `/approuter` subdirectory of the domain home directory. Supporting libraries for the AR can be stored in a `/lib` subdirectory within `/approuter`. (If you have multiple implementations of `SipApplicationRouter`, use the `-Djavax.servlet.sip.ar.spi.SipApplicationRouterProvider` option at startup to specify which one to use.)

Note: In a clustered environment, the custom AR is deployed to all engine tier instances of the domain; you cannot deploy different AR implementations within the same domain.

Oracle Communications Converged Application Server provides several configuration parameters to specify the AR class and to pass initialization properties to the AR or AR. To configure these parameters using the Administration Console:

1. Access the Administration Console for your domain.
2. Select the SipServer node in the left pane.
3. Select Configuration->Application Router in the right pane.
4. Use the options on the Application Router pane to configure the custom AR:

- **Use Custom AR**: Select this option to use a custom AR instead of the Default AR. Note that you must restart the server after selecting or clearing this option, to switch between using the DAR and a custom AR.
- **Custom AR filename**: Specify only the filename of the custom AR (packaged as a JAR) to use. The custom AR implementation must reside in the `$DOMAIN_HOME/approuter` subdirectory.
- **AR configuration data**: Enter properties to pass to the AR in the `init` method. The options are passed either to the DAR or custom AR, depending on whether the **Use Custom AR** option is selected.

  All configuration properties must conform to the Java Properties format. DAR properties must further adhere to the detailed property format described in Appendix C of the SIP Servlet Specification v1.1. Each property must be listed on a separate, single line without line breaks or spaces, as in:

  ```
  INVITE:("OriginatingCallWaiting","DAR:From","ORIGINATING","","NO_ROU
  TE","0"),("CallForwarding","DAR:To","TERMINATING","","NO_ROUTE","1")
  ```

  ```
  SUBSCRIBE:("CallForwarding","DAR:To","TERMINATING","","NO_ROUTE","1"
  )
  ```

  You can optionally specify AR initialization properties when starting the Oracle Communications Converged Application Server instance by including the `-Djavax.servlet.sip.ar.dar.configuration` Java option. (To specify a property file, rather than a URI, include the prefix `file:///`) If you specify the Java startup option, the container ignores any configuration properties defined in **AR configuration data** (stored in `sipserver.xml`). You can modify the properties in **AR configuration data** at any time, but the properties are not passed to the AR until the server is restarted with the `-Djavax.servlet.sip.ar.dar.configuration` option omitted.
- **Default application name**: Enter the name of a default application that the container should call when the custom AR cannot find an application to process an initial request. If no default application is specified, the container returns a 500 error if the AR cannot select an application.

  Note: You must first deploy an application before specifying its name as the value of **Default application name**.

5. Select Save.

Note: These configuration options are persisted as XML elements in the `sipserver.xml` file. See the *Configuration Reference Manual* for more information.

See Section 15 in the SIP Servlet Specification v1.1 for more information about the function of the AR. See also the SIP Servlet v1.1 API for information about how to implement a custom AR.

Session Key-Based Request Targeting

The SIP Servlet v1.1 specification also provides a mechanism for associating an initial request with an existing `SipApplicationSession` object. This mechanism is called session key-based targeting. Session key-based targeting is used to direct initial requests having a particular subscriber (request URI) or region, or other feature to an already-existing `SipApplicationSession`, rather than generating a new session. To use this targeting mechanism with an application, you create a method that generates a unique key and annotate that method with `@SipApplicationKey`. When the SIP container selects that application (for example, as a result of the AR choosing it for an initial request), it obtains a key using the annotated method, and uses the key and application name to determine if the `SipApplicationSession` exists. If one exists, the container associates the new request with the existing session, rather than generating a new session.

Note: If you develop a spiral proxy application using this targeting mechanism, and the application modifies the record-route more than once, it should generate different keys for the initial request, if necessary, when processing record-route hops. If it does not, then the application cannot discriminate record-route hops for subsequent requests.

See section 15 in the SIP Servlet Specification v1.1 for more information about using session key-based targeting.

Developing Converged Applications

The following sections describe how to develop converged HTTP and SIP applications with Oracle Communications Converged Application Server:

- "Overview of Converged Applications" on page 6-1
- "Assembling and Packaging a Converged Application" on page 6-2
- "Working with SIP and HTTP Sessions" on page 6-2
- "Using the Converged Application Example" on page 6-6

Overview of Converged Applications

In a *converged application*, SIP protocol functionality is combined with HTTP or Java EE components to provide a unified communication service. For example, an online push-to-talk application might enable a customer to initiate a voice call to ask questions about products in their shopping cart. The SIP session initiated for the call is associated with the customer's HTTP session, which enables the employee answering the call to view customer's shopping cart contents or purchasing history.

You must package converged applications that utilize Java EE components into an application archive (.EAR file). Converged applications that use utilize SIP and HTTP protocols should be packaged in a single SAR or WAR file containing both a `sip.xml` and a `web.xml` deployment descriptor file.You can optionally package the SIP and HTTP Servlets of a converged application into separate SAR and WAR components within a single EAR file.

The HTTP and SIP sessions used in a converged application can be accessed programmatically via a common application session object. The SIP Servlet API also helps you associate HTTP sessions with an application session.

Assembling and Packaging a Converged Application

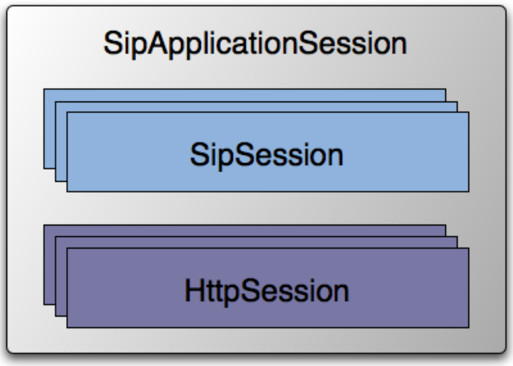
The SIP Servlet specification fully describes the requirements and restrictions for assembling converged applications. The following statements summarize the information in the SIP Servlet specification:

- Use the standard SIP Servlet directory structure for converged applications.
- Store all SIP Servlet files under the `WEB-INF` subdirectory; this ensures that the files are not served up as static files by an HTTP Servlet.
- Include deployment descriptors for both the HTTP and SIP components of your application. This means that both `sip.xml` and `web.xml` descriptors are required. A `weblogic.xml` deployment descriptor may also be included to configure Servlet functionality in the Oracle Communications Converged Application Server container.
- Observe the following restrictions on deployment descriptor elements:
  - The `distributable` tag must be present in both `sip.xml` and `web.xml`, or it must be omitted entirely.
  - `context-param` elements are shared for a given converged application. If you define the same `context-param` element in `sip.xml` and in `web.xml`, the parameter must have the same value in each definition.
  - If either the `display-name` or `icons` element is required, the element must be defined in both `sip.xml` and `web.xml`, and it must be configured with the same value in each location.

Working with SIP and HTTP Sessions

As shown in Figure 6-1, each converged application deployed to the Oracle Communications Converged Application Server container has a unique `SipApplicationSession`, which can contain one or more `SipSession` and `HttpSession` objects.

Figure 6-1 Sessions in a Converged Application



The API provided by `javax.servlet.SipApplicationSession` enables you to iterate through all available sessions in a given `SipApplicationSession`. It also provides methods to encode a URL with the unique application session when developing converged applications.

In prior releases, Oracle Communications Converged Application Server extended the basic SIP Servlet API to provide methods for:

- Creating new HTTP sessions from a SIP Servlet
- Adding and removing HTTP sessions from `SipApplicationSession`
- Obtaining `SipApplicationSession` objects using either the call ID or session ID
- Encoding HTTP URLs with session IDs from within a SIP Servlet

This functionality is now provided directly as part of the SIP Servlet API version 1.1, and the proprietary API (`com.bea.wcp.util.Sessions`) is now deprecated. Table 6-1 lists the SIP

Servlet APIs to use in place of now deprecated methods. See the SIP Servlet v1.1 API JavaDoc for more information.

Table 6-1 Deprecated com.bea.wcp.util.Sessions Methods

Deprecated Method (in com.bea.wcp.util.Sessions)	Replacement Method	Description
getApplicationSession	javax.servlet.sip.SipSessionsUtil.getApplicationSession	Obtains the `SipApplicationSession` object with a specified session ID.
getApplicationSessionsByCallId	None.	Obtains an Iterator of `SipApplicationSession` objects associated with the specified call ID.
createHttpSession	None.	Applications can instead cast an `HttpSession` into `ConvergedHttpSession`.
setApplicationSession	javax.servlet.sip.ConvergedHttpSession.getApplicationSession	Associates an HTTP session with an existing `SipApplicationSession`.
removeApplicationSession	None.	Removes an HTTP session from an existing `SipApplicationSession`.
getEncodeURL	javax.servlet.sip.ConvergedHttpSession.encodeURL	Encodes an HTTP URL with the `jsessionid` of an existing HTTP session object.

Notes: The `com.bea.wcp.util.Sessions` API is provided only for backward compatibility. Use the SIP Servlet APIs for all new development. Oracle Communications Converged Application Server does not support converged applications that mix the `com.bea.wcp.util.Sessions` API and JSR 289 convergence APIs.

Specifically, the deprecated `Sessions.getApplicationSessionsByCallId(String callId)` method cannot be used with v1.1 SIP Servlets that use the session key-based targeting method for associating an initial request with an existing `SipApplicationSession` object. See Section 15.11.2 in the SIP Servlet Specification v1.1 for more information about this targeting mechanism.

Modifying the SipApplicationSession

When using a replicated domain, Oracle Communications Converged Application Server automatically provides concurrency control when a SIP Servlet modifies a `SipApplicationSession` object. In other words, when a SIP Servlet modifies the `SipApplicationSession` object, the SIP container automatically locks other applications from modifying the object at the same time.

Non-SIP applications, such as HTTP Servlets, must themselves ensure that the application call state is locked before modifying it in a replicated environment. This is also required if a single SIP Servlet needs to modify other call state objects, such as when a conferencing Servlet joins multiple calls.

To help application developers manage concurrent access to the application session object, Oracle Communications Converged Application Server extends the standard `SipApplicationSession` object with `com.bea.wcp.sip.WlssSipApplicationSession`, and adds a new interface, `com.bea.wcp.sip.WlssAction` to encapsulate changes to the session. When these APIs are used, the SIP container ensures that all business logic contained within the `WlssAction` object is executed on a locked copy of the associated `SipApplicationSession` instance.

Listing 6-1 Example Code using WlssSipApplicationSession and WlssAction API

```
SipApplicationSession appSession = ...;

WlssSipApplicationSession wlssAppSession = (WlssSipApplicationSession)
appSession;

wlssAppSession.doAction(new WlssAction() {

       public Object run() throws Exception {

         // Add all business logic here.

         appSession.setAttribute("counter", latestCounterValue);

         sipSession.setAttribute("currentState", latestAppState);

         // The SIP container ensures that the run method is invoked

         // while the application session is locked.

         return null;

       }
```

```
        });
```

Using the Converged Application Example

Oracle Communications Converged Application Server includes a sample converged application that uses the com.bea.`wcp.util.Sessions` API. All source code, deployment descriptors, and build files for the example can be installed in *`WLSS_HOME`*`\samples\sipserver\examples\src\convergence`. See the `readme.html` file in the example directory for instructions about how to build and run the example.

Developing Custom Profile Providers

The following sections describe how to use the profile service API to develop custom profile providers:

- "Overview of the Profile Service API" on page 7-1
- "Implementing Profile API Methods" on page 7-3
- "Configuring and Packaging Profile Providers" on page 7-4
- "Configuring Profile Providers Using the Administration Console" on page 7-6

Overview of the Profile Service API

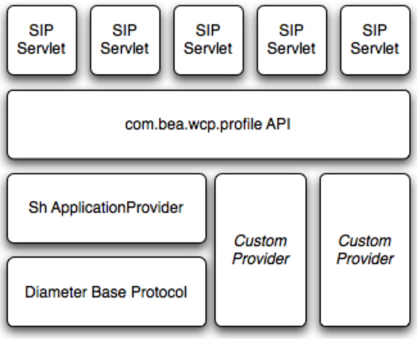
Oracle Communications Converged Application Server includes a profile service API, `com.bea.wcp.profile.API`, that can be used to create profile provider implementations. A profile provider performs the work of accessing XML documents from a data repository using a defined protocol. Deployed SIP Servlets and other applications need not understand the underlying protocol or the data repository in which the document is stored; they simply reference profile data using a custom URL, and Oracle Communications Converged Application Server delegates the request processing to the correct profile provider.

The provider performs the necessary protocol operations for manipulating the document. All providers work with documents in XML DOM format, so client code can work with many different types of profile data in a common way.

You can also use the profile service API to create a custom provider for retrieving document schemas using another protocol. For example, a profile provider could be created to retrieve subscription data from an LDAP store or RDBMS.

Note: The Diameter Sh application also accesses profile data from a Home Subscriber Server using the Sh protocol. Profile. Although applications access this profile data using a simple URL, the Diameter applications are implemented using the Diameter base protocol implementation rather than the profile provider API.

Figure 7-1 Profile Service API and Provider Implementation



Each profile provider implemented using the API may enable the following operations against profile data:

- Creating new documents.
- Querying and updating existing documents.
- Deleting documents.
- Managing subscriptions for receiving notifications of profile document changes.

Clients that want to use a profile provider obtain a profile service instance via a Servlet context attribute. They then construct an appropriate URL and use that URL with one of the available profile API methods to work with profile data. The contents of the URL, combined with the configuration of profile providers, determines the provider implementation that Oracle Communications Converged Application Server to process the client’s requests.

The sections that follow describe how to implement the profile service API interfaces in a custom profile provider.

Implementing Profile API Methods

A custom profile providers is implemented as a shared Java EE library (typically a simple JAR file) deployed to the engine tier cluster. The provider JAR file should include, at minimum, a class that implements com.bea.wcp.profile.ProfileServiceSpi. This interface inherits methods from `com.bea.wcp.profile.ProfileService` and defines new methods that are called during provider registration and unregistration.

In addition to the provider implementation, you must implement the com.bea.wcp.profile.ProfileSubscription interface if your provider supports subscription-based notification of profile data updates. A `ProfileSubscription` is returned to the client subscriber when the profile document is modified.

The Oracle Communications Converged Application Server JavaDoc describes each method of the profile service API in detail. Also keep in mind the following notes and best practices when implementing the profile service interfaces:

- The `putDocument`, `getDocument`, and `deleteDocument` methods each have two distinct method signatures. The basic version of a method passes only the document selector on which to operate. The alternate method signature also passes the address of the sender of the request for protocols that require explicit information about the requestor.
- The `subscribe` method has multiple method signatures to allow passing the sender's address, as well as for supporting time-based subscriptions.
- If you do not want to implement a method in `com.bea.wcp.profile.ProfileServiceSpi`, include a "no-op" method implementation that throws the OperationNotSupportedException.

com.bea.wcp.profile.ProfileServiceSpi defines provider methods that are called during registration and unregistration. Providers can create connections to data stores or perform any required initializing in the `register` method. The `register` method also supplies a `ProviderBean` instance, which includes any context parameters configured in the provider's configuration elements in `profile.xml`.

Providers should release any backing store connections, and clean up any state that they maintain, in the `unregister` method.

Configuring and Packaging Profile Providers

Providers must be deployed as a shared Java EE library, because all other deployed applications must be able to access the implementation. Creating Shared Java EE Libraries and Optional Packages in the Oracle WebLogic Server 10*g* Release 3 documentation describes how to assemble Java EE libraries. For most profile providers, you can simply package the implementation classes in a JAR file. Then register the library with Oracle Communications Converged Application Server using the instructions in Install a Java EE Library in the Oracle WebLogic Server documentation.

After installing the provider as a library, you must also identify the provider class as a provider in a `profile.xml` file. The name element uniquely identifies a provider configuration, and the `class` element identifies the Java class that implements the profile service API interfaces. One or more context parameters can also be defined for the provider, which are delivered to the implementation class in the `register` method. For example, context parameters might be used to identify backing stores to use for retrieving profile data.

Listing 7-1 shows a sample configuration for a provider that accesses data using XCAP.

Listing 7-1 Provider Mapping in profile.xml

```
<profile-service xmlns="http://www.bea.com/ns/wlcp/wlss/profile/300"
                 xmlns:sec="http://www.bea.com/ns/weblogic/90/security"
                 xmlns:xsi="http://www.w3.org/2001/XMLSchema=instance"
                 xmlns:wls="http;//www.bea.com/ns/weblogic/90/security/wls
">
 <mapping>
   <map-by>provider-name</map-by>
 </mapping>
 <provider>
    <name>xcap</name>

<provider-class>com.mycompany.profile.XcapProfileProvider</provider-class>
```

```
    <param>
       <name>server</name>
       <value>example.com</name>
    </param>
    ...
 </provider>
</profile-service>
```

Mapping Profile Requests to Profile Providers

When an application makes a request using the profile API, Oracle Communications Converged Application Server must find a corresponding provider to process the request. By default, Oracle Communications Converged Application Server maps the prefix of the requested URL to a provider `name` element defined in `profile.xml`. For example, with the basic configuration shown in Listing 7-1, Oracle Communications Converged Application Server would map profile API requests beginning with `xcap://` to the provider class `com.mycompany.profile.XcapProfileProvider`.

Alternately, you can define a `mapping` entry in `profile.xml` that lists the prefixes corresponding to each named provider. Listing 7-2 shows a mapping with two alternate prefixes.

Listing 7-2 Mapping a Provider to Multiple Prefixes

```
...
<mapping>
   <map-by>prefix</map-by>
      <provider>
          <provider-name>xcap</provider-name>
          <doc-prefix>sip</doc-prefix>
          <doc-prefix>subscribe</doc-prefix>
      </provider>
   <by-prefix>
```

```
<mapping>
...
```

If the explicit mapping capabilities of `profile.xml` are insufficient, you can create a custom mapping class that implements the `com.bea.wcp.profile.ProfileRouter` interface, and then identify that class in the `map-by-router` element. Listing 7-3 shows an example configuration.

Listing 7-3 Using a Custom Mapping Class

```
...
<mapping>
   <map-by-router>
      <class>com.bea.wcp.profile.ExampleRouter</class>
   </map-by-router>
</mapping>
...
```

Configuring Profile Providers Using the Administration Console

You can optionally use the Administration Console to create or modify a `profile.xml` file. To do so, you must enable the profile provider console extension in the `config.xml` file for your domain.

Listing 7-4 Enabling the Profile Service Resource in config.xml

```
...
<custom-resource>
    <name>ProfileService</name>
    <target>AdminServer</target>
    <descriptor-file-name>custom/profile.xml</descriptor-file-name>
```

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

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

The profile provider extension appears under the SipServer node in the left pane of the console, and enables you to configure new provider classes and mapping behavior.

Using Content Indirection in SIP Servlets

The following sections describe how to develop SIP Servlets that work with indirect content specified in the SIP message body:

- "Overview of Content Indirection" on page 8-1
- "Using the Content Indirection API" on page 8-2

Overview of Content Indirection

Data provided by the body of a SIP message can be included either directly in the SIP message body, or indirectly by specifying an HTTP URL and metadata that describes the URL content. Indirectly specifying the content of the message body is used primarily in the following scenarios:

- When the message bodies include large volumes of data. In this case, content indirection can be used to transfer the data outside of the SIP network (using a separate connection or protocol).
- For bandwidth-limited applications. In this case, content indirection provides enough metadata for the application to determine whether or not it should retrieve the message body (potentially degrading performance or response time).

Oracle Communications Converged Application Server provides a simple API that you can use to work with indirect content specified in SIP messages.

Using the Content Indirection API

The content indirection API provided by Oracle Communications Converged Application Server helps you quickly determine if a SIP message uses content indirection, and to easily retrieve all metadata associated with the indirect content. The basic API consists of a utility class, `com.bea.wcp.sip.engine.server.ContentIndirectionUtil`, and an interface for accessing content metadata, `com.bea.wcp.sip.engine.server.ICParsedData`.

SIP Servlets can use the utility class to identify SIP messages having indirect content, and to retrieve an `ICParsedData` object representing the content metadata. The `ICParsedData` object has simple “getter” methods that return metadata attributes.

Additional Information

Complete details about content indirection are available in a draft document: http://www.ietf.org/internet-drafts/draft-ietf-sip-content-indirect-mech-05.txt.

See also the Oracle Communications Converged Application Server JavaDoc for additional documentation about the content indirection API.

CHAPTER 9

Securing SIP Servlet Resources

The following sections describe how to apply security constraints to SIP Servlet resources when deploying to Oracle Communications Converged Application Server:

- "Overview of SIP Servlet Security" on page 9-1
- "Oracle Communications Converged Application Server Role Mapping Features" on page 9-2
- "Using Implicit Role Assignment" on page 9-3
- "Assigning Roles Using security-role-assignment" on page 9-4
- "Assigning run-as Roles" on page 9-8
- "Role Assignment Precedence for SIP Servlet Roles" on page 9-8
- "Debugging Security Features" on page 9-9
- "weblogic.xml Deployment Descriptor Reference" on page 9-9

Overview of SIP Servlet 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 in the `sip.xml` deployment descriptor. The `security-constraint` element defines the actual resources in the SIP Servlet, defined in `resource-collection elements`, that are to be protected. `security-constraint` also identifies the role names that are authorized to access the resources. All role names used in the `security-constraint` are defined elsewhere in `sip.xml` in a `security-role` element.

SIP Servlets can also programmatically refer to a role name within the Servlet code, and then map the hard-coded role name to an alternate role in the `sip.xml` `security-role-ref` element during deployment. Roles must be defined elsewhere in a `security-role` element before they can be mapped to a hard-coded name in the `security-role-ref` element.

The SIP Servlet specification also enables Servlets to propagate a security role to a called Enterprise JavaBean (EJB) using the `run-as` element. Once again, roles used in the `run-as` element must be defined in a separate `security-role` element in `sip.xml`.

Chapter 14 in the SIP Servlet API specification provides more details about the types of security available to SIP Servlets. SIP Servlet security features are similar to security features available with HTTP Servlets; you can find additional information about HTTP Servlet security by referring to these sections in the Oracle WebLogic Server 10*g* Release 3 documentation:

- Securing Web Applications in *Programming WebLogic Security* provides an overview of declarative and programmatic security models for Servlets.
- EJB Security-Related Deployment Descriptors in *Securing Enterprise JavaBeans (EJBs)* describes all security-related deployment descriptor elements for EJBs, including the `run-as` element used for propagating roles to called EJBs.

See also the example `sip.xml` excerpt in Listing 9-1, “Declarative Security Constraints in sip.xml,” on page 9-5.

Oracle Communications Converged Application Server Role Mapping Features

When you deploy a SIP Servlet, `security-role` definitions that were created for declarative and programmatic security must be assigned to actual principals and/or roles available in the Servlet container. Oracle Communications Converged Application Server uses the `security-role-assignment` element in `weblogic.xml` to help you map `security-role` definitions to actual principals and roles. `security-role-assignment` provides two different ways to map security roles, depending on how much flexibility you require for changing role assignment at a later time:

- The `security-role-assignment` element can define the complete list of principal names and roles that map to roles defined in `sip.xml`. This method defines the role assignment at deployment time, but at the cost of flexibility; to add or remove principals from the role, you must edit `weblogic.xml` and redeploy the SIP Servlet.

- The `externally-defined` element in `security-role-assignment` enables you to assign principal names and roles to a `sip.xml` role at any time using the Administration Console. When using the `externally-defined` element, you can add or remove principals and roles to a `sip.xml` role without having to redeploy the SIP Servlet.

Two additional XML elements can be used for assigning roles to a `sip.xml` `run-as` element: `run-as-principal-name` and `run-as-role-assignment`. These role assignment elements take precedence over `security-role-assignment` elements if they are used, as described in “Assigning run-as Roles” on page 9-8.

Optionally, you can choose to specify no role mapping elements in `weblogic.xml` to use implicit role mapping, as described in “Using Implicit Role Assignment” on page 9-3.

The sections that follow describe Oracle Communications Converged Application Server role assignment in more detail.

Using Implicit Role Assignment

With implicit role assignment, Oracle Communications Converged Application Server assigns a `security-role` name in `sip.xml` to a role of the exact same name, which should be configured in the Oracle Communications Converged Application Server security realm. To use implicit role mapping, you omit the `security-role-assignment` element in `weblogic.xml`, as well as any `run-as-principal-name`, and `run-as-role-assignment` elements use for mapping `run-as` roles.

When no role mapping elements are available in `weblogic.xml`, Oracle Communications Converged Application Server implicitly maps `sip.xml` `security-role` elements to roles having the same name. Note that implicit role mapping takes place regardless of whether the role name defined in `sip.xml` is actually available in the security realm. Oracle Communications Converged Application Server display a warning message anytime it uses implicit role assignment. For example, if you use the “everyone” role in `sip.xml` but you do not explicitly assign the role in `weblogic.xml`, the server displays the warning:

```
<Webapp: ServletContext(id=id,name=application,context-path=/context), the
role: everyone defined in web.xml has not been mapped to principals in
security-role-assignment in weblogic.xml. Will use the rolename itself as
the principal-name.>
```

You can ignore the warning message if the corresponding role has been defined in the Oracle Communications Converged Application Server security realm. The message can be disabled by defining an explicit role mapping in `weblogic.xml`.

Use implicit role assignment if you want to hard-code your role mapping at deployment time to a known principal name.

Assigning Roles Using security-role-assignment

The `security-role-assignment` element in `weblogic.xml` enables you to assign roles either at deployment time or at any time using the Administration Console. The sections that follow describe each approach.

Important Requirement for Oracle Communications Converged Application Server

If you specify a `security-role-assignment` element in `weblogic.xml`, Oracle Communications Converged Application Server requires that you also define a duplicate `security-role` element in a `web.xml` deployment descriptor. This requirement applies even if you are deploying a pure SIP Servlet, which would not normally require a `web.xml` deployment descriptor (generally reserved for HTTP Web Applications).

Note: If you specify a `security-role-assignment` in `weblogic.xml` but there is no corresponding `security-role` element in `web.xml`, Oracle Communications Converged Application Server generates the error message:

```
The security-role-assignment references an invalid security-role:
rolename
```

The server then implicitly maps the `security-role` defined in `sip.xml` to a role of the same name, as described in "Using Implicit Role Assignment" on page 9-3.

For example, Listing 9-1 shows a portion of a `sip.xml` deployment descriptor that defines a security constraint with the role, `roleadmin`. Listing 9-2 shows that a `security-role-assignment` element has been defined in `weblogic.xml` to assign principals and roles to `roleadmin`. In Oracle Communications Converged Application Server, this Servlet *must* be deployed with a `web.xml` deployment descriptor that also defines the `roleadmin` role, as shown in Listing 9-3.

If the `web.xml` contents were not available, Oracle Communications Converged Application Server would use implicit role assignment and assume that the `roleadmin` role was defined in the security realm; the principals and roles assigned in `weblogic.xml` would be ignored.

Listing 9-1 Declarative Security Constraints in sip.xml

```
...
  <security-constraint>
      <resource-collection>
      <resource-name>RegisterRequests</resource-name>
      <servlet-name>registrar</servlet-name>
    </resource-collection>
    <auth-constraint>
      <role-name>roleadmin</role-name>
    </auth-constraint>
  </security-constraint>

  <security-role>
    <role-name>roleadmin</role-name>
  </security-role>
...
```

Listing 9-2 Example security-role-assignment in weblogic.xml

```
<weblogic-web-app>
  <security-role-assignment>
       <role-name>roleadmin</role-name>
       <principal-name>Tanya</principal-name>
       <principal-name>Fred</principal-name>
       <principal-name>system</principal-name>
  </security-role-assignment>
</weblogic-web-app>
```

Listing 9-3 Required security-role Element in web.xml

```
<!DOCTYPE web-app
    PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
    "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
  <security-role>
    <role-name>roleadmin</role-name>
  </security-role>
</web-app>
```

Assigning Roles at Deployment Time

A basic `security-role-assignment` element definition in `weblogic.xml` declares a mapping between a `security-role` defined in `sip.xml` and one or more principals or roles available in the Oracle Communications Converged Application Server security realm. If the `security-role` is used in combination with the `run-as` element in `sip.xml`, Oracle Communications Converged Application Server assigns the first principal or role name specified in the `security-role-assignment` to the `run-as` role.

Listing 9-2, "Example security-role-assignment in weblogic.xml," on page 9-5 shows an example `security-role-assignment` element. This example assigns three users to the `roleadmin` role defined in Listing 9-1, "Declarative Security Constraints in sip.xml," on page 9-5. To change the role assignment, you must edit the `weblogic.xml` descriptor and redeploy the SIP Servlet.

Dynamically Assigning Roles Using the Administration Console

The `externally-defined` element can be used in place of the `<principal-name>` element to indicate that you want the security roles defined in the `role-name` element of `sip.xml` to use mappings that you assign in the Administration Console. The `externally-defined` element gives you the flexibility of not having to specify a specific security role mapping for each security role at deployment time. Instead, you can use the Administration Console to specify and modify role assignments at anytime.

Additionally, because you may elect to use this element for some SIP Servlets and not others, it is not necessary to select the **ignore roles and polices from DD** option for the security realm. (You select this option in the **On Future Redeploys:** field on the **General** tab of the **Security->Realms->myrealm** control panel on the Administration Console.) Therefore, within the same security realm, deployment descriptors can be used to specify and modify security for some applications while the Administration Console can be used to specify and modify security for others.

Note: When specifying security role names, observe the following conventions and restrictions:

- The proper syntax for a security role name is as defined for an Nmtoken in the Extensible Markup Language (XML) recommendation available on the Web at: http://www.w3.org/TR/REC-xml#NT-Nmtoken.
- Do not use blank spaces, commas, hyphens, or any characters in this comma-separated list: \t, < >, #, |, &, ~, ?, ( ), { }.
- Security role names are case sensitive.
- The Oracle-suggested convention for security role names is that they be singular.

Listing 9-4 shows an example of using the `externally-defined` element with the `roleadmin` role defined in Listing 9-1, "Declarative Security Constraints in sip.xml," on page 9-5. To assign existing principals and roles to the `roleadmin` role, the Administrator would use the Oracle Communications Converged Application Server Administration Console.

See Users, Groups, and Security Roles in the Oracle WebLogic Server 10*g* Release 3 documentation for information about adding and modifying security roles using the Administration Console.

Listing 9-4 Example externally-defined Element in weblogic.xml

```
<weblogic-web-app>
     <security-role-assignment>
         <role-name>webuser</role-name>
         <externally-defined/>
     </security-role-assignment>
</weblogic-web-app>
```

Assigning run-as Roles

The `security-role-assignment` described in “Assigning Roles Using security-role-assignment” on page 9-4 can be also be used to map `run-as` roles defined in `sip.xml`. Note, however, that two additional elements in `weblogic.xml` take precedence over the `security-role-assignment` if they are present: `run-as-principal-name` and `run-as-role-assignment`.

`run-as-principal-name` specifies an existing principle in the security realm that is used for all `run-as` role assignments. When it is defined within the `servlet-descriptor` element of `weblogic.xml`, `run-as-principal-name` takes precedence over any other role assignment elements for `run-as` roles.

`run-as-role-assignment` specifies an existing role or principal in the security realm that is used for all `run-as` role assignments, and is defined within the `weblogic-web-app` element.

See “weblogic.xml Deployment Descriptor Reference” on page 9-9 for more information about individual `weblogic.xml` descriptor elements. See also “Role Assignment Precedence for SIP Servlet Roles” on page 9-8 for a summary of the role mapping precedence for declarative and programmatic security as well as `run-as` role mapping.

Role Assignment Precedence for SIP Servlet Roles

Oracle Communications Converged Application Server provides several ways to map `sip.xml` roles to actual roles in the SIP Container during deployment. For declarative and programmatic security defined in `sip.xml`, the order of precedence for role assignment is:

1. If `weblogic.xml` assigns a `sip.xml` role in a `security-role-assignment` element, the `security-role-assignment` is used.

   Note: Oracle Communications Converged Application Server also requires a role definition in `web.xml` in order to use a `security-role-assignment`. See “Important Requirement for Oracle Communications Converged Application Server” on page 9-4.

2. If no `security-role-assignment` is available (or if the required `web.xml` role assignment is missing), implicit role assignment is used.

For `run-as` role assignment, the order of precedence for role assignment is:

1. If `weblogic.xml` assigns a `sip.xml` `run-as` role in a `run-as-principal-name` element defined within `servlet-descriptor`, the `run-as-principal-name` assignment is used.

Note: Oracle Communications Converged Application Server also requires a role definition in `web.xml` in order to assign roles with `run-as-principal-name`. See "Important Requirement for Oracle Communications Converged Application Server" on page 9-4.

2. If `weblogic.xml` assigns a `sip.xml` `run-as` role in a `run-as-role-assignment` element, the `run-as-role-assignment` element is used.

   Note: Oracle Communications Converged Application Server also requires a role definition in `web.xml` in order to assign roles with `run-as-role-assignment`. See "Important Requirement for Oracle Communications Converged Application Server" on page 9-4.

3. If `weblogic.xml` assigns a `sip.xml run-as` role in a `security-role-assignment` element, the `security-role-assignment` is used.

   Note: Oracle Communications Converged Application Server also requires a role definition in `web.xml` in order to use a `security-role-assignment`. See "Important Requirement for Oracle Communications Converged Application Server" on page 9-4.

4. If no `security-role-assignment` is available (or if the required `web.xml` role assignment is missing), implicit role assignment is used.

Debugging Security Features

If you want to debug security features in SIP Servlets that you develop, specify the `-Dweblogic.Debug=wlss.Security startup` option when you start Oracle Communications Converged Application Server. Using this debug option causes Oracle Communications Converged Application Server to display additional security-related messages in the standard output.

weblogic.xml Deployment Descriptor Reference

The `weblogic.xml` DTD contains detailed information about each of the role mapping elements discussed in this section. See weblogic.xml Deployment Descriptor Elements in the Oracle WebLogic Server 10*g* Release 3 documentation.

Developing SIP Servlets Using Eclipse

The following sections describe how to use Eclipse to develop SIP Servlets for use with Oracle Communications Converged Application Server:

- "Overview" on page 10-1
- "Setting Up the Development Environment" on page 10-2
- "Building and Deploying the Project" on page 10-6
- "Debugging SIP Servlets" on page 10-7

Overview

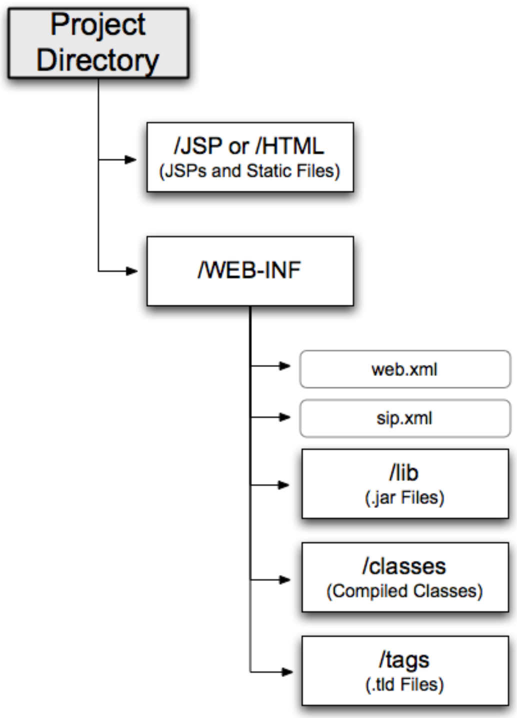
This document provides detailed instructions for using the Eclipse IDE as a tool for developing and deploying SIP Servlets with Oracle Communications Converged Application Server. The full development environment requires the following components, which you must obtain and install before proceeding:

- Oracle Communications Converged Application Server
- JDK 1.4.2
- Ant (installed with Oracle Communications Converged Application Server)
- Eclipse version 3.1
- CVS client and server (required only for version control)

SIP Servlet Organization

Building a SIP Servlet produces a Web Archive (WAR file or directory) as an end product. A basic SIP Servlet WAR file contains the subdirectories and contents described in Figure 10-1.

Figure 10-1 SIP Servlet WAR Contents



Setting Up the Development Environment

Follow these steps to set up the development environment for a new SIP Servlet project:

1. Create a new Oracle Communications Converged Application Server Domain.
2. Create a new Eclipse project.
3. Create an Ant build file.

The sections that follow describe each step in detail.

Creating a Oracle Communications Converged Application Server Domain

In order to deploy and test your SIP Servlet, you need access to a Oracle Communications Converged Application Server domain that you can reconfigure and restart as necessary. Follow the instructions in Create an Administrative Domain in the *Installation Guide* to create a new domain using the Configuration Wizard. When generating a new domain:

- Select Development Mode as the startup mode for the new domain.
- Select Sun SDK 1.4.2 as the SDK for the new domain.

Configure the Default Eclipse JVM

The latest versions of Eclipse use the version 1.5 JRE by default. Follow these steps to configure Eclipse to use the version 1.4.2 JRE installed with Oracle Communications Converged Application Server:

1. Start Eclipse.
2. Select Window->Preferences
3. Expand the Java category in the left pane, and select Installed JREs.
4. Click Add... to add the new JRE.
5. Enter a name to use for the new JRE in the JRE name field.
6. Click the Browse... button next to the JRE home directory field. Then navigate to the *`BEA_HOME`*`/jdk160_05` directory and click OK.
7. Click OK to add the new JRE.
8. Select the check box next to the new JRE to make it the default.
9. Click OK to dismiss the preferences dialog.

Creating a New Eclipse Project

Follow these steps to create a new Eclipse project for your SIP Servlet development, adding the Oracle Communications Converged Application Server libraries required for building and deploying the application:

1. Start Eclipse.
2. Select File->New->Project...
3. Select Java Project and click Next.
4. Enter a name for your project in the Project Name field.
5. In the Location field, select Create project in workspace if you have not yet begun writing the SIP Servlet code. If you already have source code available in another location, Select Create project at external location and specify the directory. Click Next.
6. Click the Libraries tab and follow these steps to add required JARs to your project:
   a. Click Add External JARs...
   b. Use the JAR selection dialog to add the `*WL_HOME*/server/lib/weblogic.jar` file to your project.
   c. Click Add External JARs... once again.
   d. Use the JAR selection dialog to add the `*WLSS_HOME*/server/lib/wlss/sipservlet.jar` file to your project.
7. Add any additional JAR files that you may require for your project.
8. Click Finish to create the new project. Eclipse displays your new project name in the Package Explorer.
9. Right-click on the name of your project and use the New->Folder command to recreate the directory structure shown in Figure 10-1, “SIP Servlet WAR Contents,” on page 10-2.

Creating an Ant Build File

Follow these steps to create an Ant build file that you can use for building and deploying your project:

1. Right-click on the name of your project in Eclipse, and select New->File

2. Enter the name `build.xml` and click Finish. Eclipse opens the empty file in a new window.
3. Copy the sample text from Listing 10-1, substituting your domain name and application name for *`myDomain`* and *`myApplication`*.

Listing 10-1 Ant Build File Contents

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<project default="all">

  <property environment="env"/>

  <property name="beahome" value="${env.BEA_HOME}"/>

  <target name="all" depends="compile,install"/>

  <target name="compile">

    <mkdir dir="WEB-INF/classes"/>

    <javac destdir="WEB-INF/classes" srcdir="src" debug="true"
debuglevel="lines,vars,source">

      <classpath>

        <pathelement path="${weblogic.jar}"/>

      </classpath>

    </javac>

  </target>

  <target name="install">

    <jar
destfile="${beahome}/user_projects/domains/myDomain/applications/myApplica
tion.war">

      <zipfileset dir="WEB-INF" prefix="WEB-INF"/>

      <zipfileset dir="WEB-INF" includes="*.html"/>

      <zipfileset dir="WEB-INF" includes="*.jsp"/>

    </jar>

  </target>
```

```
</project>
```

4. Close the `build.xml` file and save your changes.
5. Verify that the `build.xml` file is valid by selecting Window->Show View->Ant and dragging the `build.xml` file into the Ant view. Correct any problems before proceeding.
6. Right-click on the project name and select Properties.
7. Select the Builders property in the left column, and click New.
8. Select the Ant Build tool type and click OK to add an Ant builder.
9. In the Buildfile field, click Browse Workspace and select the `build.xml` file you created.
10. In the Base Directory field, click Browse Workspace and select the top-level directory for your project.
11. Click the JRE tab and choose Separate JRE in the Runtime JRE field. Use the drop-down list or the Installed JREs... button to select an installed version 1.4.2 JRE.
12. Click the Environment tab, and Click New. Enter a new name/value pair to define the `BEA_HOME` variable. The `BEA_HOME` variable must point to the home directory of the Oracle Communications Converged Application Server directory. For example:
    – Name: BEA_HOME
    – Value: c:\bea
13. Click OK to add the new Ant builder to the project.
14. De-select Java Builder in the builder list to remove the Java builder from the project.
15. Click OK to finish configuring Builders for the project.

Building and Deploying the Project

The `build.xml` file that you created compiles your code, packages the WAR, and copies the WAR file to the `/applications` subdirectory of your development domain. Oracle Communications Converged Application Server automatically deploys valid applications located in the `/applications` subdirectory.

Debugging SIP Servlets

In order to debug SIP Servlets, you must enable certain debug options when you start Oracle Communications Converged Application Server. Follow these steps to add the required debug options to the script used to start Oracle Communications Converged Application Server:

1. Use a text editor to open the `StartWebLogic.cmd` script for your development domain.
2. Beneath the line that reads:

   ```
   set JAVA_OPTIONS=
   ```

   Enter the following line:

   ```
   set DEBUG_OPTS=-Xdebug
   -Xrunjdwp:transport=dt_socket,address=9000,server=y,suspend=n
   ```

3. In the last line of the file, add the `%DEBUG_OPTS%` variable in the place indicated below:

   ```
   "%JAVA_HOME%\bin\java" %JAVA_VM% %MEM_ARGS% %JAVA_OPTIONS% %DEBUG_OPTS%

   -Dweblogic.Name=%SERVER_NAME% -Dweblogic.management.username=%WLS_USER%

   -Dweblogic.management.password=%WLS_PW%

   -Dweblogic.management.server=%ADMIN_URL%

   -Djava.security.policy="%WL_HOME%\server\lib\weblogic.policy"
   weblogic.Server
   ```

4. Save the file and use the script to restart Oracle Communications Converged Application Server.

Enabling Message Logging

The following sections describe how to use Oracle Communications Converged Application Server message logging features on a development system:

- "Overview" on page 11-1
- "Enabling Message Logging" on page 11-2
- "Specifying Content Types for Unencrypted Logging" on page 11-5
- "Example Message Log Configuration and Output" on page 11-6
- "Configuring Log File Rotation" on page 11-8

Overview

Message logging records SIP and Diameter messages (both requests and responses) received by Oracle Communications Converged Application Server. You can use the message log in a development environment to check how external SIP requests and SIP responses are received. By outputting the distinguishable information of SIP dialogs such as Call-IDs from the application log, and extracting relevant SIP messages from the message log, you can also check SIP invocations from HTTP servlets and so forth.

WARNING: The message logging functionality logs *all* SIP requests and responses; do not enable this feature in a production system. In a production system, you can instead configure one or more logging Servlets, which enable you to specify additional criteria for determining which messages to log. See Logging SIP Requests and Responses in the *Operations Guide*.

When you enable message logging, Oracle Communications Converged Application Server records log records in the Managed Server log file associated with each engine tier server instance by default. You can optionally log the messages in a separate, dedicated log file, as described in "Configuring Log File Rotation" on page 11-8.

Enabling Message Logging

You enable and configure message logging by adding a `message-debug` element to the `sipserver.xml` configuration file. Oracle Communications Converged Application Server provides two different methods of configuring the information that is logged:

- Specify a predefined logging level (terse, basic, or full), or
- Identify the exact portions of the SIP message that you want to include in a log record, in a specified order

The sections that follow describe each method of configuring message logging functionality using elements in the `sipserver.xml` file. Note that you can also set these elements using the Administration Console, in the Configuration->Message Debug tab of the SipServer console extension node.

Specifying a Predefined Logging Level

The optional `level` element in `message-debug` specifies a predefined collection of information to log for each SIP request and response. The following levels are supported:

- `terse`—Logs only the `domain` setting, logging Servlet name, logging `level`, and whether or not the message is an incoming message.
- `basic`—Logs the `terse` items plus the SIP message status, reason phrase, the type of response or request, the SIP method, the **From** header, and the **To** header.
- `full`—Logs the `basic` items plus all SIP message headers plus the timestamp, protocol, request URI, request type, response type, content type, and raw content.

Listing 11-1 shows a configuration entry that specifies the `full` logging level.

Listing 11-1 Sample Message Logging Level Configuration in sipserver.xml

```
<message-debug>
   <level>full</level>
```

```
</message-debug>
```

Customizing Log Records

Oracle Communications Converged Application Server also enables you to customize the exact content and order of each message log record. To configure a custom log record, you provide a `format` element that defines a log record `pattern` and one or more `tokens` to log in each record.

Note: If you specify a `format` element with a `<level>full</level>` element (or with the `level` element undefined) in `message-debug`, Oracle Communications Converged Application Server uses "full" message debugging and ignores the `format` entry. The `format` entry can be used in combination with either the "terse" or "basic" `message-debug` levels.

Table 11-1 describes the nested elements used in the `format` element.

Table 11-1 Nested format Elements

param-name	param-value Description
`pattern`	Specifies the pattern used to format a message log entry. The format is defined by specifying one or more integers, bracketed by "{" and "}". Each integer represents a `token` defined later in the `format` definition.
`token`	A string token that identifies a portion of the SIP message to include in a log record. `Table 11-2` provides a list of available string tokens. You can define multiple `token` elements as needed to customize your log records.

Table 11-2 describes the string `token` values used to specify information in a message log record:

Table 11-2 Available Tokens for Message Log Records

Token	Description	Example or Type
%call_id	The Call-ID header. It is blank when forwarding.	43543543
%content	The raw content.	Byte array
%content_length	The content length.	String value
%content_type	The content type.	String value

Table 11-2 Available Tokens for Message Log Records

Token	Description	Example or Type
%cseq	The CSeq header. It is blank when forwarding.	INVITE 1
%date	The date when the message was received. (“yyyy/MM/dd” format)	2004/05/16
%exception	The class name of the exception occurred when calling the AP. Detailed information is recorded to the run-time log.	NullPointerException
%from	The From header (all). It is blank when forwarding.	sip:foo@oracle.com;tag=438943
%from_addr	The address portion of the From header.	foo@oracle.com
%from_port	The port number portion of the From header.	7002
%from_tag	The tag parameter of the From header. It is blank when forwarding.	12345
%from_uri	The SIP URI part of the From header. It is blank when forwarding.	sip:foo@oracle.com
%headers	A List of message headers stored in a 2-element array. The first element is the name of the header, while the second is a list of all values for the header.	List of headers
%io	Whether the message is incoming or not.	TRUE
%method	The name of the SIP method. It records the method name to invoke when forwarding.	INVITE
%msg	Summary Call ID	String value
%mtype	The type of receiving.	SIPREQ
%protocol	The protocol used.	UDP
%reason	The response reason.	OK
%req_uri	The request URI. This token is only available for the SIP request.	sip:foo@oracle.com
%status	The response status.	200

Table 11-2 Available Tokens for Message Log Records

Token	Description	Example or Type
%time	The time when the message was received. ("HH:mm:ss" format)	18:05:27
%timestampmillis	Time stamp in milliseconds.	9295968296
%to	The To header (all). It is blank when forwarding.	sip:foo@oracle.com;tag=438943
%to_addr	The address portion of the To header.	foo@oracle.com
%to_port	The port number portion of the To header.	7002
%to_tag	The tag parameter of the To header. It is blank when forwarding.	12345
%to_uri	The SIP URI part of the To header. It is blank when forwarding.	sip:foo@oracle.com

See "Example Message Log Configuration and Output" on page 11-6 for an example `sipserver.xml` file that defines a custom log record using two tokens.

Specifying Content Types for Unencrypted Logging

By default Oracle Communications Converged Application Server uses String format (UTF-8 encoding) to log the content of SIP messages having a text or application/sdp Content-Type value. For all other Content-Type values, Oracle Communications Converged Application Server attempts to log the message content using the character set specified in the `charset` parameter of the message, if one is specified. If no `charset` parameter is specified, or if the `charset` value is invalid or unsupported, Oracle Communications Converged Application Server uses Base-64 encoding to encrypt the message content before logging the message.

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

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

Listing 11-2 shows a sample `message-debug` configuration that logs String content for three additional Content-Type values, in addition to text/* and application/sdp content.

Listing 11-2 Logging String Content for Additional Content Types

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

Example Message Log Configuration and Output

Listing 11-3 shows a sample message log configuration in `sipserver.xml`. Listing 11-4, "Sample Message Log Output," on page 11-7 shows sample output from the Managed Server log file.

Listing 11-3 Sample Message Log Configuration in sipserver.xml

```
<message-debug>
   <format>
      <pattern>{0} {1}</pattern>
      <token>%headers</token>
      <token>%content</token>
   </format>
</message-debug>
```

Listing 11-4 Sample Message Log Output

```
####<Aug 10, 2005 7:12:08 PM PDT> <Info> <WLSS.Trace> <jiri.bea.com>
<myserver> <ExecuteThread: '11' for queue: 'sip.transport.Default'> <<WLS
Kernel>> <> <BEA- 331802> <SIP Tracer: logger Message: To: sut
<sip:invite@10.32.5.230:5060> <mailto:sip:invite@10.32.5.230:5060>

Content-Length: 136

Contact: user:user@10.32.5.230:5061

CSeq: 1 INVITE

Call-ID: 59.3170.10.32.5.230@user.call.id

From: user <sip:user@10.32.5.230:5061> <mailto:sip:user@10.32.5.230:5061>
;tag=59

Via: SIP/2.0/UDP 10.32.5.230:5061

Content-Type: application/sdp

Subject: Performance Test

Max-Forwards: 70

 v=0

o=user1 53655765 2353687637 IN IP4 127.0.0.1

s=-

c=IN IP4       127.0.0.1

t=0 0

m=audio 10000 RTP/AVP 0

a=rtpmap:0 PCMU/8000

>

####<Aug 10, 2005 7:12:08 PM PDT> <Info> <WLSS.Trace> <jiri.bea.com>
<myserver> <ExecuteThread: '11' for queue: 'sip.transport.Default'> <<WLS
Kernel>> <> <BEA- 331802> <SIP Tracer: logger Message: To: sut
<sip:invite@10.32.5.230:5060> <mailto:sip:invite@10.32.5.230:5060>

Content-Length: 0

CSeq: 1 INVITE
```

```
Call-ID: 59.3170.10.32.5.230@user.call.id

Via: SIP/2.0/UDP 10.32.5.230:5061

From: user <sip:user@10.32.5.230:5061> <mailto:sip:user@10.32.5.230:5061>
;tag=59

Server: Oracle WebLogic Communications Server 10.3.1.0

 >
```

Configuring Log File Rotation

Message log entries for SIP and Diameter messages are stored in the main Oracle Communications Converged Application Server log file by default. You can optionally store the messages in a dedicated log file. Using a separate file makes it easier to locate message logs, and also enables you to use Oracle Communications Converged Application Server's log rotation features to better manage logged data.

Log rotation is configured using several elements nested within the main `message-debug` element in `sipserver.xml`. As with the other XML elements described in this section, you can also configure values using the Configuration->Message Debug tab of the SIP Server Administration Console extension.

Table 11-3 describes each element. Note that a server restart is necessary in order to initiate independent logging and log rotation.

Table 11-3 XML Elements for Configuring Log Rotation

Element	Description
`logging-enabled`	Determines whether a separate log file is used to store message debug log messages. By default, this element is set to false and messages are logged in the general Oracle Communications Converged Application Server log file.
`file-min-size`	Configures the minimum size, in kilobytes, after which the server automatically rotate log messages into another file. This value is used when the `rotation-type` element is set to `bySize`.
`log-filename`	Defines the name of the log file for storing messages. By default, the log files are stored under *`domain_home`*`/servers/`*`server_name`*`/logs`.

Table 11-3 XML Elements for Configuring Log Rotation

`rotation-type`	Configures the criterion for moving older log messages to a different file. This element may have one of the following values: • `bySize`—This default setting rotates log messages based on the specified `file-min-size`. • `byTime`—This setting rotates log messages based on the specified `rotation-time`. • `none`—Disables log rotation.
`number-of-files-limited`	Specifies whether or not the server places a limit on the total number of log files stored after a log rotation. By default, this element is set to false.
`file-count`	Configures the maximum number of log files to keep when `number-of-files-limited` is set to true.
`rotate-log-on-startup`	Determines whether the server should rotate the log file at server startup time.
`log-file-rotation-dir`	Configures a directory in which to store rotated log files. By default, rotated log files are stored in the same directory as the active log file.
`rotation-time`	Configures a start time for log rotation when using the `byTime` log rotation criterion.
`file-time-span`	Specifies the interval, in hours, after which the log file is rotated. This value is used when the `rotation-type` element is set to `byTime`.
`date-format-pattern`	Specifies the pattern to use for rending dates in log file entries. The value of this element must conform to the `java.text.SimpleDateFormat` class.

Listing 11-5 shows a sample `message-debug` configuration using log rotation.

Listing 11-5 Sample Log Rotation Configuration

```
<?xml version='1.0' encoding='UTF-8'?>
<sip-server xmlns="http://www.bea.com/ns/wlcp/wlss/300"
xmlns:sec="http://www.bea.com/ns/weblogic/90/security"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:wls="http://www.bea.com/ns/weblogic/90/security/wls">
  <message-debug>
```

```
        <logging-enabled>true</logging-enabled>
        <file-min-size>500</file-min-size>
        <log-filename>sip-messages.log</log-filename>
        <rotation-type>byTime</rotation-type>
        <number-of-files-limited>true</number-of-files-limited>
        <file-count>5</file-count>
        <rotate-log-on-startup>false</rotate-log-on-startup>
        <log-file-rotation-dir>old_logs</log-file-rotation-dir>
        <rotation-time>00:00</rotation-time>
        <file-time-span>20</file-time-span>
        <date-format-pattern>MMM d, yyyy h:mm a z</date-format-pattern>
      </message-debug>
    </sip-server>
```

Generating SNMP Traps from Application Code

The following sections describe how to use the Oracle Communications Converged Application Server `SipServletSnmpTrapRuntimeMBean` to generate SNMP traps from within a SIP Servlet:

- "Overview" on page 12-1
- "Requirement for Accessing SipServletSnmpTrapRuntimeMBean" on page 12-2
- "Obtaining a Reference to SipServletSnmpTrapRuntimeMBean" on page 12-3
- "Generating a SNMP Trap" on page 12-5

See Configuring SNMP in the *Operations Guide* for information about configuring SNMP in a Oracle Communications Converged Application Server domain.

Overview

Oracle Communications Converged Application Server includes a runtime MBean, `SipServletSnmpTrapRuntimeMBean,` that enables applications to easily generate SNMP traps. The Oracle Communications Converged Application Server MIB contains seven new OIDs that are reserved for traps generated by an application. Each OID corresponds to a severity level that the application can assign to a trap, in order from the least severe to the most severe:

- Info
- Warning
- Error

- Notice
- Critical
- Alert
- Emergency

To generate a trap, an application simply obtains an instance of the `SipServletSnmpTrapRuntimeMBean` and then executes a method that corresponds to the desired trap severity level (`sendInfoTrap()`, `sendWarningTrap()`, `sendErrorTrap()`, `sendNoticeTrap()`, `sendCriticalTrap()`, `sendAlertTrap()`, and `sendEmergencyTrap()`). Each method takes a single parameter—the String value of the trap message to generate.

For each SNMP trap generated in this manner, Oracle Communications Converged Application Server also automatically transmits the Servlet name, application name, and Oracle Communications Converged Application Server instance name associated with the calling Servlet.

Requirement for Accessing SipServletSnmpTrapRuntimeMBean

In order to obtain a `SipServletSnmpTrapRuntimeMBean`, the calling SIP Servlet must be able to perform MBean lookups from the Servlet context. To enable this functionality, you must assign a Oracle Communications Converged Application Server administrator `role-name` entry to the `security-role` and `run-as` role elements in the `sip.xml` deployment descriptor. Listing 12-1 shows a sample `sip.xml` file with the required role elements highlighted.

Listing 12-1 Sample Role Requirement in sip.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE sip-app
   PUBLIC "-//Java Community Process//DTD SIP Application 1.0//EN"
   "http://www.jcp.org/dtd/sip-app_1_0.dtd">
<sip-app>
  <display-name>My SIP Servlet</display-name>
  <distributable/>
  <servlet>
    <servlet-name>myservlet</servlet-name>
    <servlet-class>com.mycompany.MyServlet</servlet-class>
    <run-as>
      <role-name>weblogic</role-name>
    </run-as>
  </servlet>
  <servlet-mapping>
    <servlet-name>myservlet</servlet-name>
    <pattern>
      <equal>
       <var>request.method</var>
       <value>INVITE</value>
      </equal>
    </pattern>
  </servlet-mapping>
  <security-role>
    <role-name>weblogic</role-name>
  </security-role>
</sip-app>
```

Obtaining a Reference to SipServletSnmpTrapRuntimeMBean

Any SIP Servlet that generates SNMP traps must first obtain a reference to the `SipServletSnmpTrapRuntimeMBean`. Listing 12-2 shows the sample code for a method to obtain the MBean.

Listing 12-2 Sample Method for Accessing SipServletSnmpTrapRuntimeMBean

```
public SipServletSnmpTrapRuntimeMBean getServletSnmpTrapRuntimeMBean() {
    MBeanHome localHomeB = null;
    SipServletSnmpTrapRuntimeMBean ssTrapMB = null;

    try
    {
      Context ctx = new InitialContext();
      localHomeB = (MBeanHome)ctx.lookup(MBeanHome.LOCAL_JNDI_NAME);
      ctx.close();
    } catch (NamingException ne){
      ne.printStackTrace();
    }

    Set set = localHomeB.getMBeansByType("SipServletSnmpTrapRuntime");
    if (set == null || set.isEmpty()) {
      try {
        throw new ServletException("Unable to lookup type 
'SipServletSnmpTrapRuntime'");
      } catch (ServletException e) {
        e.printStackTrace();
      }
    }
    ssTrapMB = (SipServletSnmpTrapRuntimeMBean) set.iterator().next();
    return ssTrapMB;
}
```

Generating a SNMP Trap

In combination with the method shown in Listing 12-2, Listing 12-3 demonstrates how a SIP Servlet would use the MBean instance to generate an SNMP trap in response to a SIP INVITE.

Listing 12-3 Generating a SNMP Trap

```
public class MySservlet extends SipServlet {
  private SipServletSnmpTrapRuntimeMBean sipServletSnmpTrapMb = null;


  public MyServlet () {
  }


  public void init (ServletConfig sc) throws ServletException {
    super.init (sc);
    sipServletSnmpTrapMb = getServletSnmpTrapRuntimeMBean();
  }


  protected void doInvite(SipServletRequest req) throws IOException {
    sipServletSnmpTrapMb.sendInfoTrap("Rx Invite from " +
req.getRemoteAddr() + "with call id" + req.getCallId());
  }
}
```