



BEA WebLogic SIP Server™

Developing Applications with WebLogic SIP Server

Version 2.2
Revised: May 16, 2006

Copyright

Copyright © 1995-2006 BEA Systems, Inc. All Rights Reserved.

Restricted Rights Legend

This software is protected by copyright, and may be protected by patent laws. No copying or other use of this software is permitted unless you have entered into a license agreement with BEA authorizing such use. This document is protected by copyright and may not be copied photocopied, reproduced, translated, or reduced to any electronic medium or machine readable form, in whole or in part, without prior consent, in writing, from BEA Systems, Inc.

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

Trademarks and Service Marks

Copyright © 1995-2006 BEA Systems, Inc. All Rights Reserved. BEA, BEA JRockit, BEA WebLogic Portal, BEA WebLogic Server, BEA WebLogic Workshop, Built on BEA, Jolt, JoltBeans, SteelThread, Top End, Tuxedo, and WebLogic are registered trademarks of BEA Systems, Inc. BEA AquaLogic, BEA AquaLogic Data Services Platform, BEA AquaLogic Enterprise Security, BEA AquaLogic Interaction, BEA AquaLogic Interaction Analytics, BEA AquaLogic Interaction Collaboration, BEA AquaLogic Interaction Content Services, BEA AquaLogic Interaction Data Services, BEA AquaLogic Interaction Integration Services, BEA AquaLogic Interaction Process, BEA AquaLogic Interaction Publisher, BEA AquaLogic Interaction Studio, BEA AquaLogic Service Bus, BEA AquaLogic Service Registry, BEA Builder, BEA Campaign Manager for WebLogic, BEA eLink, BEA Kodo, BEA Liquid Data for WebLogic, BEA Manager, BEA MessageQ, BEA SALT, BEA Service Architecture Leveraging Tuxedo, BEA WebLogic Commerce Server, BEA WebLogic Communications Platform, BEA WebLogic Enterprise, BEA WebLogic Enterprise Platform, BEA WebLogic Enterprise Security, BEA WebLogic Express, BEA WebLogic Integration, BEA WebLogic Java Adapter for Mainframe, BEA WebLogic JDriver, BEA WebLogic Log Central, BEA WebLogic Mobility Server, BEA WebLogic Network Gatekeeper, BEA WebLogic Personalization Server, BEA WebLogic Personal Messaging API, BEA WebLogic Platform, BEA WebLogic Portlets for Groupware Integration, BEA WebLogic Real Time, BEA WebLogic RFID Compliance Express, BEA WebLogic RFID Edge Server, BEA WebLogic RFID Enterprise Server, BEA WebLogic Server Process Edition, BEA WebLogic SIP Server, BEA WebLogic WorkGroup Edition, BEA Workshop for WebLogic Platform, BEA Workshop JSP, BEA Workshop JSP Editor, BEA Workshop Struts, BEA Workshop Studio, Dev2Dev, Liquid Computing, and Think Liquid are trademarks of BEA Systems, Inc. Accelerated Knowledge Transfer, AKT, BEA Mission Critical Support, BEA Mission Critical Support Continuum, and BEA SOA Self Assessment are service marks of BEA Systems, Inc.

All other names and marks are property of their respective owners.

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. Requirements and Best Practices for WebLogic SIP Server Applications

Overview of Developing and Porting Applications for WebLogic SIP Server 2.2	2-1
Applications Must Not Create Threads	2-2
Servlets Must Be Non-Blocking	2-2
Store all Application Data in the Session	2-3
All Session Data Must Be Serializable	2-3

..... Use setAttribute() to Modify Session Data in “No-Call” Scope	2-3
send() Calls Are Buffered	2-5
Mark SIP Servlets as Distributable	2-5
Observe Best Practices for J2EE Applications	2-5
.....	2-5

3. Composing SIP Applications

Overview of SIP Application Composition	3-1
Application Composition Model	3-1
Sample Composer Application	3-3
Troubleshooting Application Composition	3-5

4. Developing Converged Applications

Overview of Converged Applications	4-1
Assembling and Packaging a Converged Application	4-2
Working with SIP and HTTP Sessions	4-2
Modifying the SipApplicationSession	4-4
Using the Converged Application Example	4-5

5. Using the Profile Service API (Diameter Sh Interface)

Overview of Profile Service API and Sh Interface Support	5-1
Enabling the Sh Interface Provider	5-2
Overview of the Profile Service API	5-2
Creating a Document Key for Application-Managed Profile Data	5-3
Using a Constructed Document Key to Manage Profile Data	5-5
Monitoring Profile Data with ProfileListener	5-6

6. Using Content Indirection in SIP Servlets

Overview of Content Indirection	6-1
---------------------------------------	-----

Using the Content Indirection API	6-2
Additional Information	6-2

7. Securing SIP Servlet Resources

Overview of SIP Servlet Security	7-1
WebLogic SIP Server Role Mapping Features	7-2
Using Implicit Role Assignment	7-3
Assigning Roles Using security-role-assignment	7-4
Important Requirement for WebLogic SIP Server 2.2	7-4
Assigning Roles at Deployment Time	7-6
Dynamically Assigning Roles Using the Administration Console	7-6
Assigning run-as Roles	7-7
Role Assignment Precedence for SIP Servlet Roles	7-8
Debugging Security Features	7-9
weblogic.xml Deployment Descriptor Reference	7-9

8. Developing SIP Servlets Using Eclipse

Overview	8-1
SIP Servlet Organization	8-2
Setting Up the Development Environment	8-2
Creating a WebLogic SIP Server Domain	8-3
Configure the Default Eclipse JVM	8-3
Creating a New Eclipse Project	8-3
Creating an Ant Build File	8-4
Building and Deploying the Project	8-6
Debugging SIP Servlets	8-6

9. Enabling Access Logging

Overview	9-1
----------------	-----

Enabling Access Logging	9-2
Specifying a Predefined Logging Level.	9-2
Customizing Log Records	9-3
Specifying Content Types for Unencrypted Logging	9-5
Example Access Log Configuration and Output.	9-6

10. Generating SNMP Traps from Application Code

Overview	10-1
Requirement for Accessing SipServletSnmpTrapRuntimeMBean.	10-2
Obtaining a Reference to SipServletSnmpTrapRuntimeMBean.	10-3
Generating a SNMP Trap	10-5

Overview of SIP Servlets

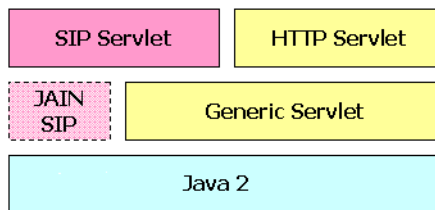
What is a SIP Servlet?

The SIP Servlet API is a part of JAIN APIs and being standardized as JSR116 of JCP (Java Community Process). The SIP Servlet API version 1.0 was published in February, 2003.

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.

J2EE 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

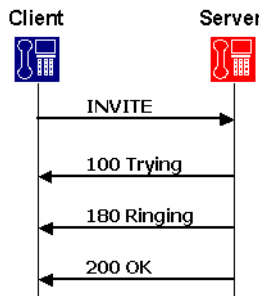
Multiple Responses

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:

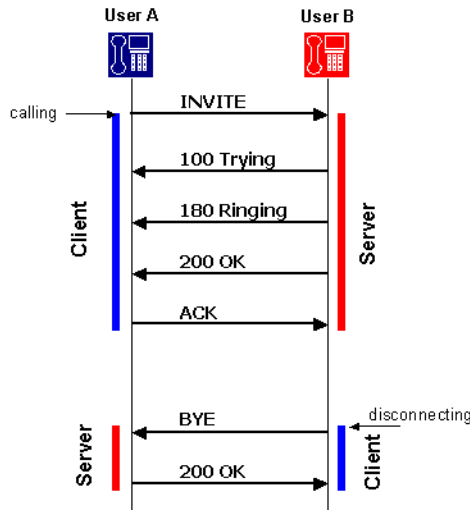
```
protected void doInvite(SipServletRequest req);
```

```
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).

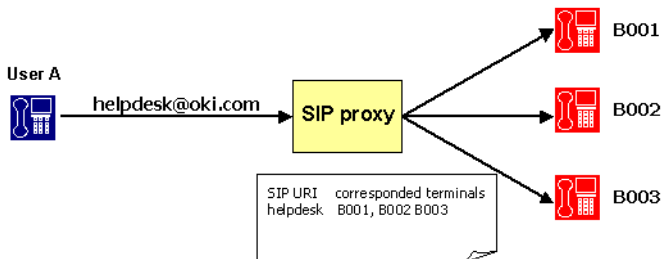
Existence of methods to receive responses indicates that in SIP Servlet requests and responses are independently transmitted an application in different threads. Applications must explicitly manage association of SIP messages. An independent request and response makes the process slightly complicated, 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	Starting Line
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	Header Field
Blank Line	Separator
v=0 o=- 5894032 5894032 IN IP4 10.2.0.100 S=SDP Media Session ... The rest is omitted.	Message Body

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

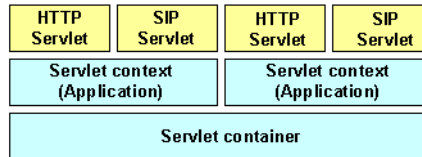
The following sections describes major functions provided by WebLogic SIP Server as a SIP servlet container:

- 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.

WebLogic SIP 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 J2EE 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 J2EE.

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 WebLogic SIP 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.”

WebLogic SIP Server automatically execute the following response and retransmission processes:

- Sending “100 Trying”—When WebLogic SIP Server receives an INVITE request, it automatically creates and sends “100 Trying.”
- Response to CANCEL—When WebLogic SIP 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 SIP 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 SIP 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.

Note: In WebLogic SIP Server, HTTP sessions are not associated with application sessions.

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 `Record-Route` 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.

Requirements and Best Practices for WebLogic SIP Server Applications

The following sections requirements and best practices for developing applications for deployment to WebLogic SIP Server:

- [“Overview of Developing and Porting Applications for WebLogic SIP Server 2.2” on page 2-1](#)
- [“Applications Must Not Create Threads” on page 2-2](#)
- [“Servlets Must Be Non-Blocking” on page 2-2](#)
- [“Store all Application Data in the Session” on page 2-3](#)
- [“All Session Data Must Be Serializable” on page 2-3](#)
- [“Use setAttribute\(\) to Modify Session Data in “No-Call” Scope” on page 2-3](#)
- [“send\(\) Calls Are Buffered” on page 2-5](#)
- [“Mark SIP Servlets as Distributable” on page 2-5](#)
- [“Observe Best Practices for J2EE Applications” on page 2-5](#)

Overview of Developing and Porting Applications for WebLogic SIP Server 2.2

In a typical production environment, SIP applications are deployed to a cluster of WebLogic SIP Server instances that form the engine tier cluster. A separate cluster of servers in the 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 WebLogic SIP Server, many of the conventions and restrictions described below may be new to you, because previous WebLogic SIP 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

WebLogic SIP 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 WebLogic SIP Server architecture, construct your application modules according to the SIP Servlet and J2EE 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 `doxxx` method of a SIP Servlet. If the `doxxx` method spawns additional threads or accesses a different call state before returning control, deadlock scenarios 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 WebLogic SIP 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.

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 WebLogic SIP 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 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, WebLogic SIP 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 `doxxx` method of a SIP Servlet. However, applications may also attempt to modify session data in “no-call” scope. No-call scope refers the context where call state data is modified outside the scope of a normal `doxxx` 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. This ensures that only one application modifies the data at a time, and also ensures that your changes are replicated across data tier nodes in a cluster.

If you use other set methods to change objects within a session, WebLogic SIP Server cannot replicate those changes.

Note that the WebLogic SIP 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:

Requirements and Best Practices for WebLogic SIP Server Applications

```
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;

    }

});
```

```
});
```

See [“Modifying the SipApplicationSession” on page 4-4](#) 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 WebLogic SIP 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, WebLogic SIP 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) WebLogic SIP Server instance.

Observe Best Practices for J2EE Applications

If you are deploying applications that use other J2EE 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 WebLogic Server 8.1 Documentation for more information.

Requirements and Best Practices for WebLogic SIP Server Applications

Composing SIP Applications

The following sections describe how to use WebLogic SIP Server 2.2 application composition features:

- [“Overview of SIP Application Composition” on page 3-1](#)
- [“Application Composition Model” on page 3-1](#)
- [“Sample Composer Application” on page 3-3](#)
- [“Troubleshooting Application Composition” on page 3-5](#)

Overview of SIP Application Composition

Application composition is the process of “chaining” multiple SIP applications, such as Proxies, User Agent Servers (UAS), User Agent Clients (UAC), redirect servers, and Back-to-Back User Agents (B2BUA), into a logical path that processes a given SIP request. The sections that follow describe an application composition programming model that can be deployed to WebLogic SIP Server. By using this programming model, you can define a set of applications responsible for processing a given initial SIP request, as well as the logic for how and when each application should modify the request. The WebLogic SIP Server container ensures that each application remains on the call path for subsequent message processing requests.

Application Composition Model

The basic WebLogic SIP Server application composition model involves creating a main “composer” application that examines initial SIP requests to determine which deployed

applications should process the request. For example, a composer application may examine the user specified in the Request-URI header and select applications based on the user's subscription level.

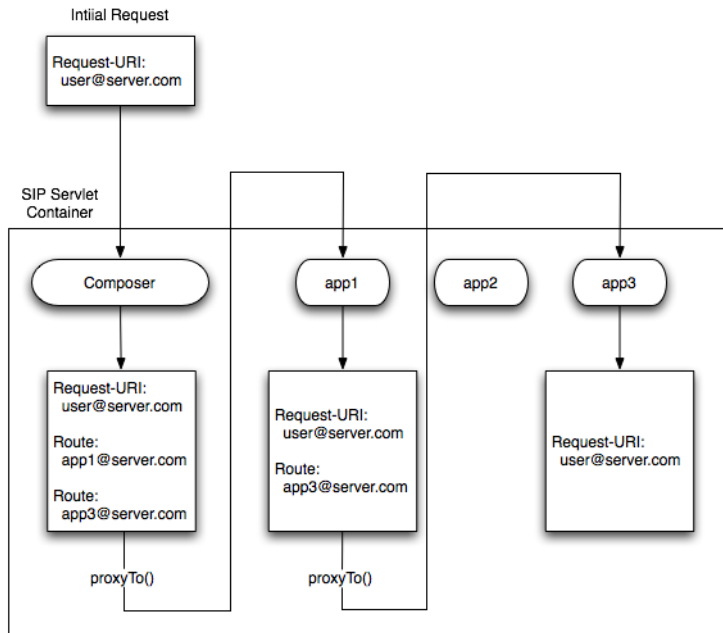
The composer application should insert one or more Route headers into the request, with each Route header specifying the name and location of a deployed SIP application that should process the request. Application names are defined similar to user addresses, using the format:

application@address

where *application* is the deployment name of the SIP application and *address* is the address of the load balancer used to contact the WebLogic SIP Server installation, the cluster address, or the listen address of the server itself (for example, `proxyappl@mycompany.com`). The order of the Route headers in the message should dictate the required order of application execution. The Request-URI header of the initial request should remain unchanged.

After inserting Route headers to chain the required applications, the composer application then proxies the message using the original Request-URI. The container examines the contents of the initial Route header in the request. If the user name portion of the route header matches a deployed application name and the address matches a configured server address, then the container delivers the request to the named application.

After each application processes the request, the top Route header is removed the message is then proxied to the next application as shown in [Figure 3-1](#).

Figure 3-1 Composed Application Model

If a request is proxied to another server, the SIP Servlet container inserts the session IDs of chained applications into the Record-Route header of the message. The session IDs ensure that each server hosting a chained application remains in the call path for subsequent requests.

Sample Composer Application

[Listing 3-1](#) shows the organization of a simple composer application.

Listing 3-1 Sample Composer Application

```
package example;

import javax.servlet.sip.SipFactory;
import javax.servlet.sip.SipServletRequest;
```

Composing SIP Applications

```
import javax.servlet.sip.SipURI;
import javax.servlet.sip.SipServlet;
import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import java.io.IOException;

public class Composer extends SipServlet {
    private SipFactory factory;

    private static final String CLUSTER_ADDRESS = "example.com";

    public void init(ServletConfig sc) throws ServletException {
        super.init(sc);
        factory = (SipFactory)
            getServletContext().getAttribute("javax.servlet.sip.SipFactory");
    }

    protected void doRequest(SipServletRequest req)
        throws ServletException, IOException {

        if (!req.isInitial()) {
            super.doRequest(req);
            return;
        }

        SipURI[] routeSet = getRouteSet(req);
        for (int i = 0; i < routeSet.length; i++) {
```

```

        req.pushRoute(routeSet[i]);
    }

    req.getProxy().proxyTo(req.getRequestURI());
}

/*
 * Returns application route set for specified request. Ideally, this route
 * set should be based on the requesting user's subscribed services. In
 * this example, it is fixed for all users.
 */
private SipURI[] getRouteSet(SipServletRequest req) {
    return new SipURI[] { createRouteURI("app1"), createRouteURI("app2") };
}

private SipURI createRouteURI(String appName) {
    SipURI uri = factory.createSipURI(appName, CLUSTER_ADDRESS);
    uri.setLrParam(true);
    return uri;
}
}

```

Troubleshooting Application Composition

WebLogic SIP Server examines the first Route header in a message to determine two things:

1. Does the username portion of the header match the name of a deployed SIP application?
2. Does the address portion of the header indicate that the application is intended for this WebLogic SIP Server instance?

Both of these conditions must be met in order for the SIP Servlet container to route a request to an application specified in the Route header. If either condition is not met, Weblogic SIP Server uses the default Servlet mapping rules defined in the Servlet's deployment descriptor to process the request.

For example, if the username portion of the first Route header does not match a deployed application name, default Servlet mapping rules are used to process the request. Always ensure that the composer application embeds the correct application names into Route headers when chaining applications together.

Even if the username matches a deployed application, the address portion must also match one of the configured addresses for the WebLogic SIP Server instance:

- A load balancer URI configured in `sipserver.xml`
- The cluster address for the WebLogic SIP Server engine tier
- A listen address for the server instance itself (default listen address or the listen address of a network channel)

To ensure that the address of an application matches the server address, ensure that the composer application is embedding the proper address string in Route headers. Also ensure that the server instances are configured using the same address string. See [loadbalancer](#) and [Configuring WebLogic SIP Server Network Resources](#) in *Configuring and Managing WebLogic SIP Server*.

Developing Converged Applications

The following sections describe how to develop converged HTTP and SIP applications with WebLogic SIP Server:

- [“Overview of Converged Applications” on page 4-1](#)
- [“Assembling and Packaging a Converged Application” on page 4-2](#)
- [“Working with SIP and HTTP Sessions” on page 4-2](#)
- [“Using the Converged Application Example” on page 4-5](#)

Overview of Converged Applications

In a *converged application*, SIP protocol functionality is combined with other protocols (generally HTTP) 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 assemble converged applications using the basic SIP Servlet directory structure outlined in JSR 116. Converged applications require both a `sip.xml` and a `web.xml` deployment descriptor files.

The HTTP and SIP sessions used in a converged application can be accessed programmatically via a common application session object. WebLogic SIP Server provides an extended API to help you associate HTTP sessions with an application session.

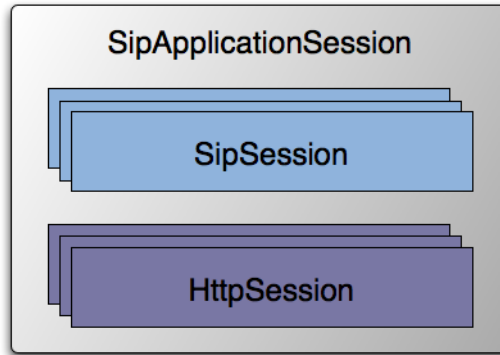
Assembling and Packaging a Converged Application

JSR 116 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 WebLogic SIP 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 4-1](#), each application deployed to the WebLogic SIP Server container has a single `SipApplicationSession`, which can contain one or more `SipSession` and `HttpSession` objects. The basic API provided by `javax.servlet.SipApplicationSession` enables you to iterate through all available sessions available in a given `SipApplicationSession`. However, the basic API specified by JSR 116 does not define methods to obtain a given `SipApplicationSession` or to create or associate HTTP sessions with a `SipApplicationSession`.

Figure 4-1 Sessions in a Converged Application

WebLogic SIP Server extends the basic 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

These extended API methods are available in the utility class `com.bea.wcp.util.Sessions`. [Table 4-1](#) provides a summary of each method. See the [JavaDoc](#) for more details on this utility class.

Table 4-1 Summary of `com.bea.wcp.util.Sessions` Methods

Method	Description
<code>getApplicationSession</code>	Obtain a <code>SipApplicationSession</code> object with a specified session ID.
<code>getApplicationSessionsByCallId</code>	Obtain an Iterator of <code>SipApplicationSession</code> objects associated with the specified call ID.
<code>createHttpSession</code>	Create an HTTP session from within a SIP Servlet. You can modify the HTTP session state and associate the new session with an existing <code>SipApplicationSession</code> for later use.

Table 4-1 Summary of com.bea.wcp.util.Sessions Methods

Method	Description
setApplicationSession	Associate an HTTP session with an existing SipApplicationSession.
removeApplicationSession	Removes an HTTP session from an existing SipApplicationSession.
getEncodeURL	Encodes an HTTP URL with the sessionId of an existing HTTP session object.

Modifying the SipApplicationSession

When using a replicated domain, WebLogic SIP 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, WebLogic SIP 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 4-1 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

WebLogic SIP 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 are installed in `WLSS_HOME\samples\server\examples\src\converged`. See the `readme.html` file in the example directory for instructions about how to build and run the example.

Using the Profile Service API (Diameter Sh Interface)

The following sections describe how to use the profile service API, based on the WebLogic SIP Server Diameter protocol implementation, in your own applications:

- [“Overview of Profile Service API and Sh Interface Support” on page 5-1](#)
- [“Enabling the Sh Interface Provider” on page 5-2](#)
- [“Overview of the Profile Service API” on page 5-2](#)
- [“Creating a Document Key for Application-Managed Profile Data” on page 5-3](#)
- [“Using a Constructed Document Key to Manage Profile Data” on page 5-5](#)
- [“Monitoring Profile Data with ProfileListener” on page 5-6](#)

Overview of Profile Service API and Sh Interface Support

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

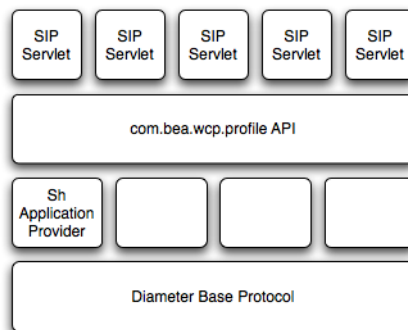
- To query or update a user’s data stored on the HSS
- To subscribe to and receive notifications when a user’s data changes on the HSS

The user data available to an AS may be defined by a service running on the AS (*repository data*), or it may be a subset of the user’s IMS profile data hosted on the HSS. The Sh interface specification, 3GPP TS 29.328 V5.11.0, defines the IMS profile data that can be queried and

updated via Sh. All user data accessible via the Sh interface is presented as an XML document with the schema defined in 3GPP TS 29.328.

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

Figure 5-1 Profile Service API and Sh Provider Implementation



WebLogic SIP Server 2.2 includes only a single provider for the Sh interface. Future versions of WebLogic SIP Server may include new providers to support additional interfaces defined in the IMS specification. Applications using the profile service API will be able to use additional providers as they are made available.

Enabling the Sh Interface Provider

See [Configuring Diameter Sh Client Nodes and Relay Agents](#) in *Configuring and Managing WebLogic SIP Server* for full instructions on setting up Diameter support.

Overview of the Profile Service API

WebLogic SIP Server provides a simple profile service API that SIP Servlets can use to query or modify subscriber profile data, or to manage subscriptions for receiving notifications about

changed profile data. Using the API, a SIP Servlet explicitly requests user profile documents via the Sh provider application. The provider returns an XML document, and the Servlet can then use standard DOM techniques to read or modify profile data in the local document. Updates to the local document are applied to the HSS after a “put” operation.

Creating a Document Key for Application-Managed Profile Data

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

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

The `protocol` portion of the selector identifies the Diameter interface provider to use for retrieving the document. In WebLogic SIP Server version 2.2, only the Sh interface is provided, so only Sh XML documents (beginning with the `sh://` protocol designation) can be retrieved.

With Sh document selectors, the next element, `uri`, generally corresponds to the User-Identity or Public-Identity of the user whose profile data is being retrieved. If you are requesting an Sh data reference of type LocationInformation or UserState, the URI value can be the User-Identity or MSISDN for the user.

[Table 5-1](#) summarizes the possible URI values that can be supplied depending on the Sh data reference you are requesting. 3GPP TS 29.328 V5.11.0 describes the possible data references and associated reference types in more detail.

Table 5-1 Possible URI Values for Sh Data References

Sh Data Reference Number	Data Reference Type	Possible URI Value in Document Selector
0	RepositoryData	User-Identity or Public-Identity
10	IMSPublicIdentity	
11	IMSUserState	
12	S-CSCFName	
13	InitialFilterCriteria	

Table 5-1 Possible URI Values for Sh Data References

Sh Data Reference Number	Data Reference Type	Possible URI Value in Document Selector
14	LocationInformation	User-Identity or MSISDN
15	UserState	
17	Charging information	User-Identity or Public-Identity
17	MSISDN	

The final element of the document selector, *reference_type*, specifies the data reference type being requested. For some data reference requests, only the *uri* and *reference_type* are required. Other Sh requests use an access key, which requires a third element in the document selector corresponding to the value of the Attribute-Value Pair (AVP) defined in the key.

[Table 5-2](#) summarizes the required document selector elements for each type of Sh data reference request.

Table 5-2 Summary of Document Selector Elements for Sh Data Reference Requests

Data Reference Type	Required Document Selector Elements	Example Document Selector
RepositoryData	sh://uri/reference_type/Service-Indication	sh://sip:user@bea.com/RepositoryData/Call Screening/
IMSPublicIdentity	sh://uri/reference_type/[<i>Identity-Set</i>] where <i>Identity-Set</i> is one of: <ul style="list-style-type: none"> • All-Identities • Registered-Identities • Implicit-Identities 	sh://sip:user@bea.com/IMSPublicIdentity/Registered-Identities
IMSUserState	sh://uri/reference_type	sh://sip:user@bea.com/IMSUserState/
S-CSCFName	sh://uri/reference_type	sh://sip:user@bea.com/S-CSCFName/
InitialFilterCriteria	sh://uri/reference_type/Server-Name	sh://sip:user@bea.com/InitialFilterCriteria/www.bea.com/

Table 5-2 Summary of Document Selector Elements for Sh Data Reference Requests

Data Reference Type	Required Document Selector Elements	Example Document Selector
LocationInformation	sh://uri/reference_type/(CS-Domain PS-Domain)	sh://sip:user@bea.com/LocationInformation/CS-Domain/
UserState	sh://uri/reference_type/(CS-Domain PS-Domain)	sh://sip:user@bea.com/UserState/PS-Domain/
Charging information	sh://uri/reference_type	sh://sip:user@bea.com/Charging information/
MSISDN	sh://uri/reference_type	sh://sip:user@bea.com/MSISDN/

Using a Constructed Document Key to Manage Profile Data

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

Note: If Diameter Sh client node services are not available on the WebLogic SIP Server instance when `getDocument()` the profile service throws a “No registered provider for protocol” exception.

WebLogic SIP Server caches the documents returned from the profile service for the duration of the service method invocation (for example, when a `doRequest()` method is invoked). If the service method requests the same profile document multiple times, the subsequent requests are served from the cache rather than by re-querying the HSS.

[Listing 5-1](#) shows a sample SIP Servlet that obtains and modifies profile data.

Listing 5-1 Sample Servlet Using ProfileService to Retrieve and Write User Profile Data

```
package demo;

import com.bea.wcp.profile.*;
```

Using the Profile Service API (Diameter Sh Interface)

```
import javax.servlet.sip.SipServletRequest;
import javax.servlet.sip.SipServlet;
import org.w3c.dom.Document;
import java.io.IOException;

public class MyServlet extends SipServlet {
    private ProfileService psvc;

    public void init() {
        psvc = (ProfileService)
getServletContext().getAttribute(ProfileService.PROFILE_SERVICE);
    }

    protected void doInvite(SipServletRequest req) throws IOException {
        String docSel = "sh://" + req.getTo() + "/IMSUserState/";
        // Obtain and change a profile document.
        Document doc = psvc.getDocument(docSel); // Document is read only.
        Document docCopy = (Document) doc.cloneNode(true);
        // Modify the copy using DOM.
        psvc.putDocument(docSel, docCopy); // Apply the changes.
    }
}
```

Monitoring Profile Data with ProfileListener

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

Note: In a replicated environment, Diameter relay nodes always attempt to push notifications directly to the engine tier server that subscribed for profile updates. If that engine tier

server is unavailable, another server in the engine tier cluster is chosen to receive the notification. This model succeeds because session information is stored in the data tier, rather than the engine tier.

Actual subscriptions are managed using the `subscribe` method of the `com.bea.wcp.profile.ProfileService` helper class. The `subscribe` method requires that you supply the current `SipApplicationSession` and the key for the profile data document you want to monitor. See [“Creating a Document Key for Application-Managed Profile Data” on page 5-3](#).

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

[Listing 5-2](#) shows sample code for a Servlet that implements the `ProfileListener` interface.

Listing 5-2 Sample Servlet Implementing ProfileListener Interface

```
package demo;

import com.bea.wcp.profile.*;
import javax.servlet.sip.SipServletRequest;
import javax.servlet.sip.SipServlet;
import org.w3c.dom.Document;
import java.io.IOException;

public class MyServlet extends SipServlet implements ProfileListener {
    private ProfileService psvc;

    public void init() {
        psvc = (ProfileService)
getServletContext().getAttribute(ProfileService.PROFILE_SERVICE);
    }

    protected void doInvite(SipServletRequest req) throws IOException {
        String docSel = "sh://" + req.getTo() + "/IMSUserState/";
        // Subscribe to profile data.
        psvc.subscribe(req.getApplicationSession(), docSel, null);
    }
}
```

Using the Profile Service API (Diameter Sh Interface)

```
}  
  
    public void update(ProfileSubscription ps, Document document) {  
        System.out.println("IMSUserState updated: " +  
ps.getDocumentSelector());  
    }  
}
```

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 6-1](#)
- [“Using the Content Indirection API” on page 6-2](#)

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).

WebLogic SIP 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 WebLogic SIP 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://bgp.potaroo.net/ietf/ids/draft-ietf-sip-content-indirect-mech-05.txt>.

See also the [WebLogic SIP Server JavaDoc](#) for additional documentation about the content indirection API.

Securing SIP Servlet Resources

The following sections describe how to apply security constraints to SIP Servlet resources when deploying to WebLogic SIP Server 2.2:

- [“Overview of SIP Servlet Security” on page 7-1](#)
- [“WebLogic SIP Server Role Mapping Features” on page 7-2](#)
- [“Using Implicit Role Assignment” on page 7-3](#)
- [“Assigning Roles Using security-role-assignment” on page 7-4](#)
- [“Assigning run-as Roles” on page 7-7](#)
- [“Role Assignment Precedence for SIP Servlet Roles” on page 7-8](#)
- [“Debugging Security Features” on page 7-9](#)
- [“weblogic.xml Deployment Descriptor Reference” on page 7-9](#)

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 WebLogic Server 8.1 SP5 documentation:

- [J2EE Security Model](#) 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 7-1](#), “[Declarative Security Constraints in sip.xml](#),” on page 7-4.

WebLogic SIP 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. WebLogic SIP Server 2.2 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 7-7](#).

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 7-3](#).

The sections that follow describe WebLogic SIP Server role assignment in more detail.

Using Implicit Role Assignment

With implicit role assignment, WebLogic SIP Server assigns a `security-role` name in `sip.xml` to a role of the exact same name, which should be configured in the WebLogic SIP 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 used for mapping `run-as` roles.

When no role mapping elements are available in `weblogic.xml`, WebLogic SIP 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. WebLogic SIP Server displays 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 WebLogic SIP 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 WebLogic SIP Server 2.2

If you specify a `security-role-assignment` element in `weblogic.xml`, WebLogic SIP Server 2.2 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`, WebLogic SIP Server 2.2 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 7-3](#).

For example, [Listing 7-1](#) shows a portion of a `sip.xml` deployment descriptor that defines a security constraint with the role, `roleadmin`. [Listing 7-2](#) shows that a `security-role-assignment` element has been defined in `weblogic.xml` to assign principals and roles to `roleadmin`. In WebLogic SIP Server 2.2, this Servlet *must* be deployed with a `web.xml` deployment descriptor that also defines the `roleadmin` role, as shown in [Listing 7-3](#).

If the `web.xml` contents were not available, WebLogic SIP 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 7-1 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 7-2 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 7-3 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 WebLogic SIP Server security realm. If the `security-role` is used in combination with the `run-as` element in `sip.xml`, WebLogic SIP Server assigns the first principal or role name specified in the `security-role-assignment` to the `run-as` role.

[Listing 7-2, “Example security-role-assignment in weblogic.xml,” on page 7-5](#) shows an example `security-role-assignment` element. This example assigns three users to the `roleadmin` role defined in [Listing 7-1, “Declarative Security Constraints in sip.xml,” on page 7-4](#). 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 BEA suggested convention for security role names is that they be singular.

[Listing 7-4](#) shows an example of using the `externally-defined` element with the `roleadmin` role defined in [Listing 7-1](#), “[Declarative Security Constraints in sip.xml](#),” on [page 7-4](#). To assign existing principals and roles to the `roleadmin` role, the Administrator would use the WebLogic SIP Server Administration Console.

See [Security Roles](#) in the WebLogic Server 8.1 SP5 documentation for information about adding and modifying security roles using the Administration Console.

Listing 7-4 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 7-4](#) 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 7-9 for more information about individual `weblogic.xml` descriptor elements. See also “[Role Assignment Precedence for SIP Servlet Roles](#)” on page 7-8 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

WebLogic SIP 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: WebLogic SIP Server 2.2 also requires a role definition in `web.xml` in order to use a `security-role-assignment`. See “[Important Requirement for WebLogic SIP Server 2.2](#)” on page 7-4.

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: WebLogic SIP Server 2.2 also requires a role definition in `web.xml` in order to assign roles with `run-as-principal-name`. See “[Important Requirement for WebLogic SIP Server 2.2](#)” on page 7-4.

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: WebLogic SIP Server 2.2 also requires a role definition in `web.xml` in order to assign roles with `run-as-role-assignment`. See “[Important Requirement for WebLogic SIP Server 2.2](#)” on page 7-4.

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

Note: WebLogic SIP Server 2.2 also requires a role definition in `web.xml` in order to use a `security-role-assignment`. See [“Important Requirement for WebLogic SIP Server 2.2”](#) on page 7-4.

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 WebLogic SIP Server. Using this debug option causes WebLogic SIP 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 <http://www.bea.com/servers/wls810/dtd/weblogic810-web-jar.dtd> for the complete DTD. See also [weblogic.xml Deployment Descriptor Elements](#) in the WebLogic Server 8.1 SP5 documentation.

Developing SIP Servlets Using Eclipse

The following sections describe how to use Eclipse to develop SIP Servlets for use with WebLogic SIP Server:

- [“Overview” on page 8-1](#)
- [“Setting Up the Development Environment” on page 8-2](#)
- [“Building and Deploying the Project” on page 8-6](#)
- [“Debugging SIP Servlets” on page 8-6](#)

Overview

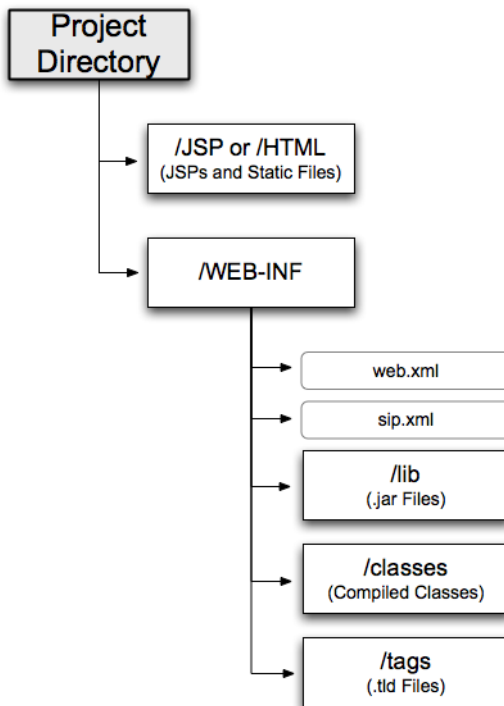
This document provides detailed instructions for using the Eclipse IDE as a tool for developing and deploying SIP Servlets with WebLogic SIP Server 2.2. The full development environment requires the following components, which you must obtain and install before proceeding:

- WebLogic SIP Server 2.2
- JDK 1.4.2
- Ant (installed with WebLogic SIP Server 2.2)
- 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 8-1](#).

Figure 8-1 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 WebLogic SIP 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 WebLogic SIP Server Domain

In order to deploy and test your SIP Servlet, you need access to a WebLogic SIP Server domain that you can reconfigure and restart as necessary. Follow the instructions in [Creating a New WebLogic SIP Server Domain](#) 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 WebLogic SIP 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/jdk142_08 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 WebLogic SIP 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 `BEA_HOME/wlss220/server/lib/weblogic.jar` file to your project.
 - c. Click Add External JARs... once again.
 - d. Use the JAR selection dialog to add the `BEA_HOME/wlss220/telco/auxlib/sipservlet.jar` file to your project.
 - e. (Optional.) If your application needs to access WebLogic SIP Server MBeans using JMX, also use the JAR selection dialog to add `BEA_HOME/wlss220/telco/lib/wcp_sip_core.jar` 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 8-1, “SIP Servlet WAR Contents,”](#) on page 8-2.

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 8-1](#), substituting your domain name and application name for `myDomain` and `myApplication`.

Listing 8-1 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 WebLogic SIP Server 2.2 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. WebLogic SIP 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 WebLogic SIP Server. Follow these steps to add the required debug options to the script used to start WebLogic SIP 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. Save the file and use the script to restart WebLogic SIP Server.

Enabling Access Logging

The following sections describe how to use WebLogic SIP Server access logging features on a development system:

- [“Overview” on page 9-1](#)
- [“Enabling Access Logging” on page 9-2](#)
- [“Specifying Content Types for Unencrypted Logging” on page 9-5](#)
- [“Example Access Log Configuration and Output” on page 9-6](#)

Overview

Access logging records all SIP messages (both requests and responses) received by WebLogic SIP Server. You can use the access 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 access log, you can also check SIP invocations from HTTP servlets and so forth.

WARNING: The access 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 *Configuring and Managing WebLogic SIP Server*.

When you enable access logging, WebLogic SIP Server records access log records in the Managed Server log file associated with each engine tier server instance.

Enabling Access Logging

You enable and configure access logging by adding a `message-debug` element to the `sipserver.xml` configuration file. WebLogic SIP 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 access logging functionality. See also the [Engine Tier Configuration Reference \(sipserver.xml\)](#) in *Configuring and Managing WebLogic SIP Server* for a full reference to the `sipserver.xml` file contents.

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 9-1](#) shows a configuration entry that specifies the `full` logging level.

Listing 9-1 Sample Accessing Logging Level Configuration in `sipserver.xml`

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


Customizing Log Records

WebLogic SIP Server also enables you to customize the exact content and order of each access 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`, WebLogic SIP 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 9-1](#) describes the nested elements used in the `format` element.

Table 9-1 Nested format Elements

param-name	param-value Description
pattern	Specifies the pattern used to format an access log entry. The format is defined by specifying one or more integers, bracketed by “{” and “}”. Each integer represents a token defined later in the <code>format</code> definition.
token	A string token that identifies a portion of the SIP message to include in a log record. Table 9-2 provides a list of available string tokens. You can define multiple <code>token</code> elements as needed to customize your log records.

[Table 9-2](#) describes the string `token` values used to specify information in an access log record:

Table 9-2 Available Tokens for Access 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
%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

Table 9-2 Available Tokens for Access Log Records

Token	Description	Example or Type
%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@bea.com;tag=438943
%from_addr	The address portion of the From header.	foo@bea.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@bea.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@bea.com
%status	The response status.	200
%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@bea.com;tag=438943

Table 9-2 Available Tokens for Access Log Records

Token	Description	Example or Type
%to_addr	The address portion of the To header.	foo@bea.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@bea.com

See [“Example Access Log Configuration and Output” on page 9-6](#) for an example `sipserver.xml` file that defines a custom log record using two tokens.

Specifying Content Types for Unencrypted Logging

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

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

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

[Listing 9-2](#) 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 9-2 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 Access Log Configuration and Output

[Listing 9-3](#) shows a sample access log configuration in `sipserver.xml`. [Listing 9-4](#), “Sample Access Log Output,” on [page 9-6](#) shows sample output from the Managed Server log file.

Listing 9-3 Sample Access Log Configuration in `sipserver.xml`

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

Listing 9-4 Sample Access 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

```

Enabling Access Logging

```
From: user <sip:user@10.32.5.230:5061> <mailto:sip:user@10.32.5.230:5061>  
;tag=59  
Server: BEA WebLogic SIP Server 2.2.0.0  
>
```

Generating SNMP Traps from Application Code

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

- [“Overview” on page 10-1](#)
- [“Requirement for Accessing SipServletSnmpTrapRuntimeMBean” on page 10-2](#)
- [“Obtaining a Reference to SipServletSnmpTrapRuntimeMBean” on page 10-3](#)
- [“Generating a SNMP Trap” on page 10-5](#)

See [Configuring SNMP](#) in *Configuring and Managing WebLogic SIP Server* for information about configuring SNMP in a WebLogic SIP Server domain.

Overview

WebLogic SIP Server 2.2 introduces a new runtime MBean, `SipServletSnmpTrapRuntimeMBean`, that enables applications to easily generate SNMP traps. The WebLogic SIP 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 `SipServletSnmptTrapRuntimeMBean` 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, WebLogic SIP Server also automatically transmits the Servlet name, application name, and WebLogic SIP Server instance name associated with the calling Servlet.

Requirement for Accessing `SipServletSnmptTrapRuntimeMBean`

In order to obtain a `SipServletSnmptTrapRuntimeMBean`, the calling SIP Servlet must be able to perform MBean lookups from the Servlet context. To enable this functionality, you must assign a WebLogic SIP Server administrator `role-name` entry to the `security-role` and `run-as` role elements in the `sip.xml` deployment descriptor. [Listing 10-1](#) shows a sample `sip.xml` file with the required role elements highlighted.

Listing 10-1 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 10-2](#) shows the sample code for a method to obtain the MBean.

Listing 10-2 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 10-2](#), [Listing 10-3](#) demonstrates how a SIP Servlet would use the MBean instance to generate an SNMP trap in response to a SIP INVITE.

Listing 10-3 Generating a SNMP Trap

```
public class MyServlet 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());

    }

}
```

Generating SNMP Traps from Application Code