BEA WebLogic SIP Server™

Developing SIP Servlets with WebLogic SIP Server

Version 2.1
Revised: December 2, 2005

Contents

1. Overview of SIP Servlets

2. Requirements and Best Practices for WebLogic SIP Server Applications

3. Composing SIP Applications

4. Securing SIP Servlet Resources

5. Developing SIP Servlets Using Eclipse

6. Enabling Access Logging

CHAPTER 1

Overview of SIP Servlets

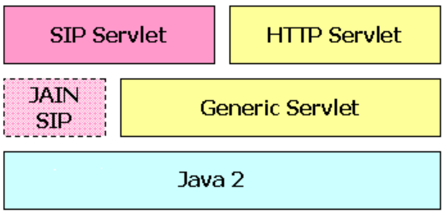
What is a SIP Servlet?

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

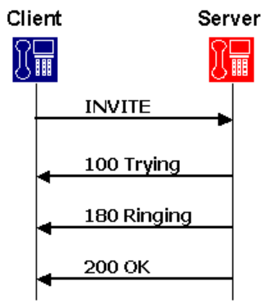
Multiple Responses

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);
```

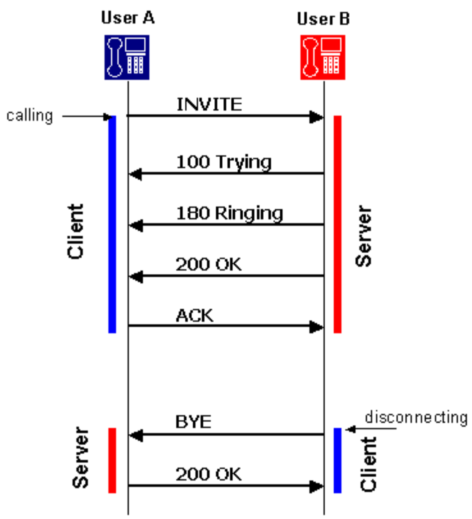
```
protected void doCancel(SipServletRequest req);
protected void doSubscribe(SipServletRequest req);
protected void doNotify(SipServletRequest req);
protected void doMessage(SipServletRequest req);
protected void doInfo(SipServletRequest req);
protected void doPrack(SipServletRequest req);
```

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).

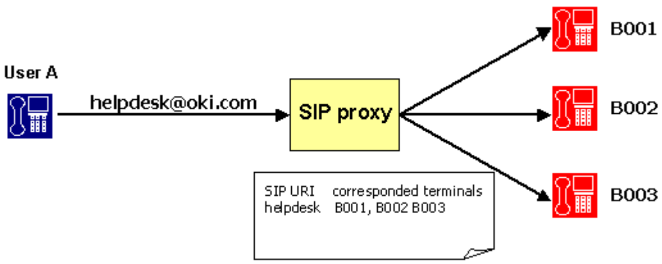
Existence of methods to receive responses indicates that in SIP Servlet requests and responses are independently transmitted an application in different threads. Applications must explicitly manage association of SIP messages. An independent request and response makes the process slightly complicated,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

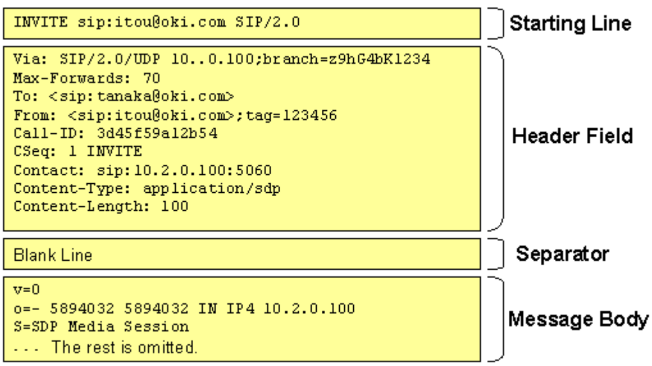


SIP Servlet provides a utility to proxy SIP requests for applications that have proxy functions.

Message Body

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

Figure 1-5 SIP Message Example



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:

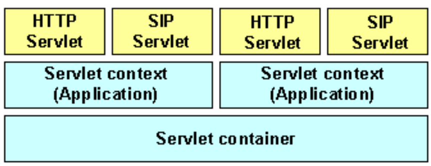
- Application Management—Describes functions such as application management by servlet context, lifecycle management of servlets, application initialization by deployment descriptors.
- SIP Messaging—Describes functions of parsing incoming SIP messages and delivering appropriate SIP servlets, sending 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 received 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.
- 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 <code>Record-Route</code> header in the specified request.
- Parallel/Sequential (parallel)—Determines whether forking is executed in parallel or sequentially.
- stateful—Determines whether proxying is transaction stateful.
- Supervising mode—In the event of the state change of proxying (response receipts), an application reports this.

CHAPTER 2

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 2.1

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.

Avoid Thread Creation

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.

In most cases, you should avoid application designs that require creating new threads in server-side modules such as SIP Servlets:

- 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.

In rare situations, creating threads may be appropriate in spite of these warnings. If you must use threads in your application code, create a finite pool of threads so that you can control the number of threads your application creates. Understand where your threads can deadlock and handle the deadlocks when they occur. Review your design carefully to ensure that your threads do not compromise the security system.

To avoid undesirable interactions with WebLogic SIP Server threads, do not let your threads call into WebLogic Server modules. For example, do not use Enterprise JavaBeans or Servlets from threads that you create. Application threads are best used for independent, isolated tasks, such as conversing with an external service or, with proper locking, reading or writing to files. A short-lived thread that accomplishes a single purpose and ends (or returns to the thread pool) is less likely to interfere with other threads.

Avoid creating daemon threads in modules that are packaged in applications deployed on WebLogic Server. When you create a daemon thread in an application module such as a Servlet, you will not be able to redeploy the application because the daemon thread created in the original deployment will remain running.

Test all multithreaded code under increasingly heavy loads, adding clients even to the point of failure. Observe the application performance and WebLogic SIP Server behavior and then add checks to prevent failures from occurring in production.

Servlets Must Be Non-Blocking

SIP and HTTP Servlets must not block threads in the body of a SIP method. 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 Persist All Changes to Session State

Use the SIP Session's `setAttribute` method to change attributes in a session object. If you set attributes in a session object with `setAttribute`, the object and its attributes are replicated in the data tier cluster. If you use other set methods to change objects within a session, WebLogic SIP Server cannot replicate those changes. Any time a change is made to an object that is in the session, call `setAttribute` to update that object across the data tier cluster. Likewise, use `removeAttribute` to remove an attribute from a session object.

Also note that the WebLogic SIP Server container does not modify the call state *after* a Servlet makes a call to `setAttribute`. For example, in the following code sample the call to `modifyState()` does not persist call state data in the data tier:

```
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 before calling `setAttribute`, as in:

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

foo.modifyState();

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

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.

Composing SIP Applications

The following sections describe how to use WebLogic SIP Server 2.1 application composition features:

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. WebLogic SIP Server provides support for an application composition model that enables applications to create and maintain a logical composition of multiple SIP applications. By using this programming model, you can easily define an order list of applications that should process a given initial SIP request, and the WebLogic SIP Server container ensures that each application remains on the call path for all subsequent requests.

Application Composition Model

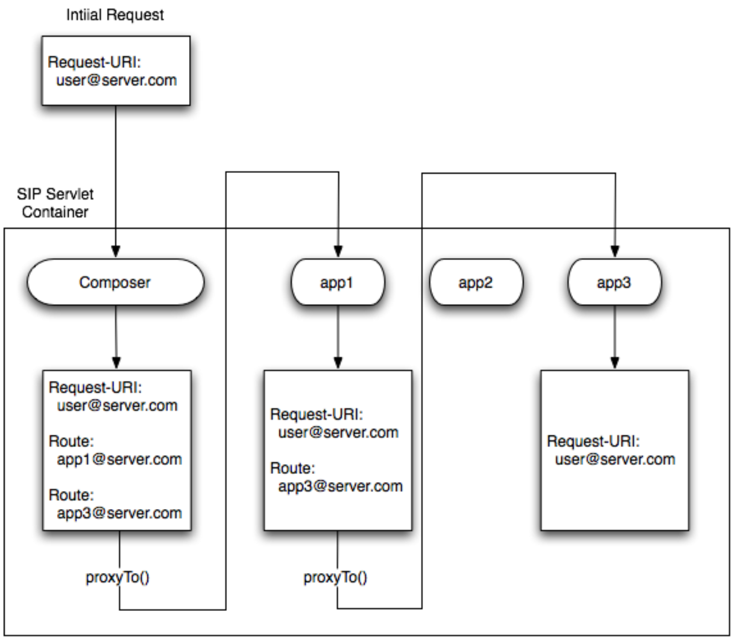
The basic WebLogic SIP Server application composition model involves creating a main "composer" application that examines initial SIP requests to determine which deployed applications should process the request, and in what order. (For example, a composer application may examine the user specified in the Request-URI header and select applications based on the user's subscription level.) The composer application then inserts 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, `proxyapp1@mycompany.com`). The order of the Route headers in the message 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 calls `proxyTo()` to proxy the message using the original Request-URI. The WebLogic SIP Server container examines the contents of the initial Route header in the request to determine if the user portion of the address refers to an application name. If the user name matches a deployed application name and the address matches a configured server address, then the server ignores any configured Servlet mapping rules and instead delivers the request to the named application.

Figure 3-1 Composed Application Model



After processing a request, applications that are part of a composed application path should remove the first Route header and `proxyTo()` the original Request-URI so that WebLogic SIP Server can direct the request to any additional applications specified in Route headers.

In this manner, WebLogic SIP Server honors the configured chain of applications that were defined by the composer application using Route headers. Figure 3-1 shows a summary of the application composition model.

Managing Proxied Requests

In addition to chaining applications for an initial SIP request, WebLogic SIP Server preserves the composed application chain for subsequent requests proxied to other servers. 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. WebLogic SIP Server examines the session IDs to 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;
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 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*.

Securing SIP Servlet Resources

The following sections describe how to apply security constraints to SIP Servlet resources when deploying to WebLogic SIP Server 2.1:

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 4-1, "Declarative Security Constraints in sip.xml," on page 4-4.

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 2.1 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 4-7.

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 4-3.

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 use 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 display a warning message anytime it uses implicit role assignment. For example, if you use the "everyone" role in `sip.xml` but you do not explicitly assign the role in `weblogic.xml`, the server displays the warning:

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

You can ignore the warning message if the corresponding role has been defined in the 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 2.1

If you specify a `security-role-assignment` element in `weblogic.xml`, WebLogic SIP Server 2.1 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 2.1 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 4-3.

For example, Listing 4-1 shows a portion of a `sip.xml` deployment descriptor that defines a security constraint with the role, `roleadmin`. Listing 4-2 shows that a `security-role-assignment` element has been defined in `weblogic.xml` to assign principals and roles to `roleadmin`. In WebLogic SIP Server 2.1, this Servlet *must* be deployed with a `web.xml` deployment descriptor that also defines the `roleadmin` role, as shown in Listing 4-3.

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 4-1 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 4-2 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 4-3 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 4-2, "Example security-role-assignment in weblogic.xml," on page 4-5 shows an example security-role-assignment element. This example assigns three users to the roleadmin role defined in Listing 4-1, "Declarative Security Constraints in sip.xml," on page 4-4. 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 4-4 shows an example of using the `externally-defined` element with the `roleadmin` role defined in Listing 4-1, "Declarative Security Constraints in sip.xml," on page 4-4. 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 4-4 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 4-4 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 4-9 for more information about individual weblogic.xml descriptor elements. See also "Role Assignment Precedence for SIP Servlet Roles" on page 4-8 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 2.1 also requires a role definition in web.xml in order to use a security-role-assignment. See "Important Requirement for WebLogic SIP Server 2.1" on page 4-4.

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 2.1 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 2.1" on page 4-4.

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 2.1 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 2.1" on page 4-4.

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 2.1 also requires a role definition in `web.xml` in order to use a `security-role-assignment`. See "Important Requirement for WebLogic SIP Server 2.1" on page 4-4.

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.

CHAPTER 5

Developing SIP Servlets Using Eclipse

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

Overview

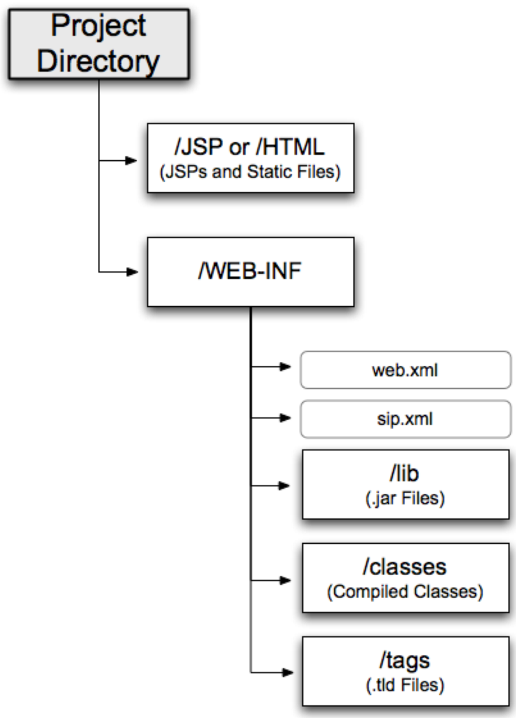
This document provides detailed instructions for using the Eclipse IDE as a tool for developing and deploying SIP Servlets with WebLogic SIP Server 2.1. The full development environment requires the following components, which you must obtain and install before proceeding:

- WebLogic SIP Server 2.1
- JDK 1.4.2
- Ant (installed with WebLogic SIP Server 2.1)
- 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 5-1.

Figure 5-1 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*`/wlss21/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*`/wlss21/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*`/wlss21/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 5-1, "SIP Servlet WAR Contents," on page 5-2.

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 5-1, substituting your domain name and application name for *`myDomain`* and *`myApplication`*.

Listing 5-1 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 2.1 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

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 6-1 shows a configuration entry that specifies the `full` logging level.

Listing 6-1 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 must omit the `level` element described in

"Specifying a Predefined Logging Level" on page 6-2 and instead provide a `format` element that defines a log record `pattern` and one or more `tokens` to log in each record.

Note: If you specify both a `level` element and a `format` element in `message-debug`, WebLogic SIP Server uses the specified `level` and ignores the `format` entry.

Table 6-1 describes the nested elements used in the `format` element.

Table 6-1 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 `format` definition.
`token`	A string token that identifies a portion of the SIP message to include in a log record. Table 6-2 provides a list of available string tokens. You can define multiple `token` elements as needed to customize your log records.

Table 6-2 describes the string `token` values used to specify information in an access log record:

Table 6-2 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
%exception	The class name of the exception occurred when calling the AP. Detailed information is recorded to the run-time log.	NullPointerException

Token	Description	Example or Type
%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
%to_addr	The address portion of the To header.	foo@bea.com
%to_port	The port number portion of the To header.	7002

Token	Description	Example or Type
%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 6-5 for an example `sipserver.xml` file that defines a custom log record using two tokens.

Example Access Log Configuration and Output

Listing 6-2 shows a sample access log configuration in `sipserver.xml`. Listing 6-3, "Sample Access Log Output," on page 6-5 shows sample output from the Managed Server log file.

Listing 6-2 Sample Access Log Configuration in sipserver.xml

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

Listing 6-3 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: sip:sipp@10.32.5.230:5061
```

```
CSeq: 1 INVITE

Call-ID: 59.3170.10.32.5.230@sipp.call.id

From: sipp <sip:sipp@10.32.5.230:5061> <mailto:sip:sipp@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@sipp.call.id

Via: SIP/2.0/UDP 10.32.5.230:5061

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

Server: BEA WebLogic SIP Server 2.1.0.0

 >
```