Java™ SIP Servlet API Specification

Ajay P. Deo

Kelvin R. Porter

Mark X. Johnson

April 27, 2000
Revision 0.5

ACKNOWLEDGMENTS

We gratefully acknowledge the encouragement and assistance of many people. First, we would like to thank Kenneth J. Fischer and Sami Syed for their continuous encouragement.

We thank the members of MCIWorldcom IP Telephony group for their comments, suggestions, and support including Dean Willis, Robert Sparks, Ben Campbell, Henry Sinnreich, John Truetken, and Teresa Hastings.

This document is based upon the Java™ Servlet Specification, v2.2 from Sun Microsystems and this document makes many references to it. James Duncan Davidson and Danny Coward wrote the Servlet Specification document. We would like to them for providing an excellent example and would like to acknowledge the significant role their specification played in the writing of this document.

REVISION HISTORY

Revision	Date	Authors	Reason for Change
0.1	July 25, 1999	Ajay Deo	Initial Specification.
0.2	September 5, 1999	Ajay Deo	Updated SDP interfaces and defined servlet mapping rules.
0.3	September 8, 1999	Ajay Deo	Added deployment descriptor and defined the associated tel-app.dtd
0.4	April 27, 2000	Kelvin Porter	Multiple modifications.
0.5	May 1-3, 2000	Ajay Deo & Kelvin Porter	Addition of interfaces and supporting material

CONTENTS

1. OVERVIEW

This document defines an architectural framework for rapid application development based on Session Initiation Protocol (SIP). The Session Initiation Protocol is a derivative of Hypertext Transfer Protocol (HTTP) and much of the message syntax and header fields are identical to HTTP/1.1. While HTTP is a protocol for HTML content exchange, SIP is a signaling protocol for creating, modifying and ending sessions for one or more participants.

Since the two protocols share so much in common, SIP can reap the benefits of some of the technological advancements surrounding HTTP; specifically Servlets and Enterprise Java Beans (EJB) based application servers. Both Servlets and EJB frameworks promote rapid application development through their well-defined application programming interfaces.

This specification defines a set of Java application programming interfaces for developing telephony applications, by extending the Java Servlet framework. Since this framework builds upon on the Servlet framework, it intends to appeal to the broad Web development community. Developers already familiar with the Java Servlet framework will be able to develop telephony applications, or set of hybrid applications based on this framework. Further, this framework also intends to appeal to the Servlet Container Providers, because they can leverage their HTTP container development expertise to provide SIP servlet containers.

Rather than repeating the interfaces syntax and semantics of the base Servlet framework, we use [HTTP Servlet X.Y] to refer to Section X.Y of the current Servlet/2.2 specification.

1.1 What is a SIP Servlet?

A SIP Servlet is a telephony component, managed by a container, which executes telephony service logic. A SIP Servlet is a Servlet specialization, just like HTTP Servlets. SIP Servlets interact with web clients via a request, acknowledge, response semantics implemented by the SIP Servlet Container. This request-acknowledge-response model reflects the behavior of the Session Initiation Protocol (SIP).

1.2 What is a SIP Servlet Container?

A SIP servlet container is a servlet container that implements all the interfaces defined by this specification. Please refer to the Servlet Container definition in [HTTP Servlet 1.2].

1.3 Client-Server Interaction

A SIP client (SIP User Agent Client), which could be browser plug-in, or a stand-alone client program initiates a SIP request, accesses a SIP server and makes a session request. This request is handled by the SIP servlet container that runs inside (or in conjunction with) the host SIP server which calls a specific servlet. This SIP servlet in turn sends a response back to the client.

This request is handled by the SIP servlet container that runs inside (or in conjunction with) the host SIP server which calls a specific servlet. This SIP servlet in turn sends a response back to the client.

The SIP Servlet API focuses primarily on enabling server-based services. Developers writing SIP clients need a different set of APIs. The authors will offer this set of client APIs in a separate draft.

1.4 An Example

A SIP client (which could be SIP phone or a program running on a PC) issues an invitation request. The server framework (within the SIP server) hands off the SIP request message to the servlet container for processing. The SIP servlet container determines which servlet to invoke based upon its internal configuration. The SIP servlet container then calls the SIP servlet with a collection of objects representing the request.

The servlet container can run in the same process as the host SIP server, in a different process on the same host, or on a different host from the SIP server for which it processes requests.

The SIP servlet extracts whatever information it needs to process the request from the request object. The servlet can then use its specified algorithm(s) to generate data to send back the response to the client. The servlet sends the data back to the client via one or more response messages.

Once the servlet completes a SIP transaction, the servlet container ensures the proper dispatch of the response and then the servlet container returns control back to the host SIP server.

1.5 Comparing SIP Servlets with Other Technologies

The comparisons between SIP servlets and other technologies are very similar to those for HTTP Servlets and other technologies.

Please refer to the Servlet Container definition in [HTTP Servlet 1.4].

1.6 API Status

This is the initial version of the SIP Servlet specification. This specification is based on Java Servlet API specification version 2.2.

The SIP Servlet Architecture served as the basis for the development of a SIP Servlet based proxy server. A reference implementation of this specification has been successfully tested in two SIP Bakeoffs (3 and 4).

2. TERMS USED

These terms are used throughout the specification.

2.1 Basic Terms

2.1.1 Servlet Definition

A servlet definition is a unique name associated with a fully qualified class name of a class implementing the `Servlet` interface. A set of initialization parameters can be associated with a servlet definition.

Please refer to [HTTP Servlet 2.1.2]

2.1.2 Servlet Mapping

A Servlet Mapping is a Servlet Definition that is associated by a container by forming unique servlet discrimination pattern consisting of field from Request-URI, From header, and the To header. The server directs all requests that match the servlet discrimination pattern to the servlet associated with the Servlet Definition.

2.1.3 SIP URL

SIP requests use SIP URLs to store different data elements including the originator (From), the current destination (Request-URI), and the final recipient (TO). SIP requests also use SIP URLs to specify redirection addresses (Contact). A SIP URL can also be embedded in web pages or other hyperlinks to indicate that the particular user or the service can be called via SIP. When used as a hyperlink, the SIP URL indicates the use of the INVITE method.

An example SIP URL would be:

```
sip:j.doe@big.com;maddr=239.255.255.1;ttl=15
```

Please refer to the latest IETF draft of the "SIP: Session Initiation Protocol" document, Section 2 "SIP Uniform Resource Locators."

2.1.4 SIP Servlet Definition

A SIP servlet definition is a unique name associated with a fully qualified class name of a class implementing the `SipServlet` interface. A set of initialization parameters can be associated with a SIP servlet definition.

2.1.5 SIP Servlet Mapping

A SIP servlet mapping is a SIP servlet definition that is associated by a SIP servlet container with a SIP URL. The SIP servlet associated with the Servlet Definition handles all requests to that SIP URL.

2.1.6 SIP Application

A SIP application is a collection of SIP servlets, HTTP servlets, JavaServer Pages 2, HTML documents, and other SIP resources that might include image files, compressed archives, and other data. A SIP application may be packaged into an archive or exist in an open directory structure.

All compatible servlet containers must accept a SIP application and perform a deployment of its contents into their runtime. This may mean that a container can run the application directly from a SIP application archive file or it may mean that it will move the contents of a SIP application into the appropriate locations for that particular container.

Please refer to [HTTP Servlet 2.1.4]

2.1.7 SIP Application Archive

A SIP application archive is a single file that contains all of the components of a SIP application. This archive file is created by using standard JAR tools which allow any or all of the SIP components to be signed.

SIP application archive files are identified by the `.sar` extension. A new extension is used instead of `.jar` because that extension is reserved for files which contain a set of class files and that can be placed in the classpath or double clicked using a GUI to launch an application. As the contents of a SIP application archive are not suitable for such use, a new extension was in order.

Please refer to [HTTP Servlet 2.1.5]

2.2 Roles

The following roles are defined to aid in identifying the actions and responsibilities taken by various parties during the development, deployment, and running of a SIP Servlet based application. In some scenarios, a single party may perform several roles; in others, a different party may perform each role.

2.2.1 SIP Application Developer

The SIP Application Developer is the producer of a SIP based application. His or her output is a set of SIP servlet classes, JSP pages, HTML pages, and supporting libraries and files (such as images, compressed archive files, etc.) for the web application. The Application Developer is typically a SIP application domain expert. The SIP developer is required to be aware of the servlet environment and its consequences when programming, including concurrency considerations, and create the SIP application accordingly.

Please refer to [HTTP Servlet 2.2.2]

2.2.2 Application Assembler

The Application Assembler takes the work done by the developer and ensures that it is a deployable unit. The input of the Application Assembler is the servlet classes, JSP pages, HTML pages, other supporting libraries and files for the SIP application. The output of the application assembler is a SIP Application Archive or a SIP Application in an open directory structure.

2.2.3 Deployer

The Deployer takes one or more SIP application archive files or other directory structures by a SIP Application Developer and deploys the application into a specific operational environment.

The operational environment includes a specific servlet container and SIP server. The Deployer must resolve all the external dependencies declared by the developer. To perform his role, the Deployer uses tools provided by the Servlet Container.

The Deployer is an expert in a specific operational environment. For example, the Deployer is responsible for mapping the security roles defined by the Application Developer to the user groups and accounts that exist within the operational environment where the SIP application is deployed.

Please refer to [HTTP Servlet 2.2.3]

2.2.4 System Administrator

The System Administrator is responsible for the configuration and administration of the SIP servlet container and SIP server. The administrator is also responsible for overseeing the well-being of the deployed SIP applications at run time.

Please refer to [HTTP Servlet 2.2.2]

2.2.5 SIP Servlet Container Provider

The SIP Servlet Container Provider is responsible for providing the runtime environment, namely the SIP servlet container and possibly the SIP server, in which a SIP application runs as well as the tools necessary to deploy web applications.

The expertise of the SIP Container Provider is in server level programming. This specification does not specify the interface between the SIP server and the SIP servlet container. It is left to the SIP Container Provider split the implementation of the required functionality between the SIP container and the SIP server. Once future version of EJB specification defines Application Server to Container interfaces, this specification will be updated to support those APIs. At that time, it will be possible to write a SIP Container that compatible with EJB application servers.

Please refer to [HTTP Servlet 2.2.5]

2.3 Security Terms

2.3.1 Principal

A principal is an entity that can be authenticated by an authentication protocol. A principal is identified by a *principal name* and authenticated by using *authentication dat*a. The content and format of the principal name and the authentication data depend on the authentication protocol.

Please refer to [HTTP Servlet 2.3.1]

2.3.2 Security Policy Domain

A security policy domain is a scope over which security policies are defined and enforced by a security administrator of the security service. A security policy domain is also sometimes referred to as a *realm*.

Please refer to [HTTP Servlet 2.3.2]

2.3.3 Security Technology Domain

A security technology domain is the scope over which the same security mechanism, such as Kerberos, is used to enforce a security policy. Multiple security policy domains can exist within a single technology domain.

Please refer to [HTTP Servlet 2.3.3]

3. THE SIP SERVLET INTERFACE

The SipServlet abstract base class is the central abstraction of the SIP Servlet API. It extends the GenericServlet class that implements the Servlet interface. SIP application developers will typically extend SipServlet class for implementing their services.

3.1 Request Handling Methods

The SipServlet class defines the default method called service for handling all SIP requests. This method de-multiplexes each request and is responsible for invoking the appropriate method on the proper instance of a SipServlet.

3.2 SIP Specific Request Handling Methods

The SipServlet abstract class defines additional methods for handling SIP requests. These methods correspond to the request methods defined by SIP specification. The service method of the SipServlet class automatically calls these methods for processing a SIP request. The SipServlet interface defines the following methods for request processing:

- **`doInvite`** for handling SIP INVITE requests
- **`doAck`** for handling SIP ACK requests
- **`doOptions`** for handling SIP OPTIONS requests
- **`doBye`** for handling SIP BYE requests
- **`doCancel`** for handling SIP CANCEL requests
- **`doRegister`** for handling SIP REGISTER requests

Typically, SIP application developers will not override the service method. The service method handles standard SIP requests by dispatching them to the handler methods for each SIP request type (the **`doXXX`** methods listed above).

Likewise, developers are not likely to override the **`doOptions`** methods. The service method supports SIP 2.0 OPTIONS requests by dispatching it and **`doOptions`**. Servlets typically run on multithreaded servers, so you must write your servlet to handle concurrent requests and synchronize access to shared resources.

Shared resources include in-memory data such as instance or class variables and external objects such as files, database connections, and network connections.

3.3 Servlet Life Cycle

The SIP Servlet life cycle is similar to the HTTP Servlet life cycle.

Please refer to [HTTP Servlet 3.3]

3.4 Loading and Instantiation

Please refer to [HTTP Servlet 3.3.1]

3.5 Initialization

Please refer to [HTTP Servlet 3.3.2]

3.6 Request Handling

Please refer to [HTTP Servlet 3.3.3]

3.7 End of Service

Please refer to [HTTP Servlet 3.3.4]

4. SIP SERVLET CONTEXT

The SIP Servlet context fulfills a similar role to the HTTP Servlet context.

Please refer to [HTTP Servlet 4]

5. THE SIP REQUEST

Similar to HTTP, messages in the SIP protocol are either: request messages or response messages. A request message consists of method tokens, message headers, and message body. The SIP request object acts like a container encapsulating all information sent by the SIP client.

5.1 Parameters

The parameters made available to a request are encapsulated via the URI of the request message or the To value of the message. This specification provides a SIPURL interface that specifies how a SipServlet can extract information from the string containing the SIP URL.

Unlike the HTTP requests, there is no posted data and no format data.

5.2 Attributes

Attributes within a request or a response are associated with a message header value.

Attributes are objects associated with a request. Attributes may be set by the container to express information that otherwise could not be expressed via the API, or may be set by a servlet to communicate information to another servlet. Attributes are accessed with the following methods of the `ServletRequest` interface:

- `getAttribute`
- `getAttributeNames`
- `setAttribute`

Only one attribute value may be associated with an attribute name.

Attribute names beginning with the prefixes of "`java.`" and "`javax.`" are reserved for definition by the HTTP Servlet Specification (Please refer to [HTTP Servlet]).

Please refer to [HTTP Servlet 5.2].

5.3 Headers

A SIP servlet can access the headers of a SIP request by using the following methods provided by the SipServletRequest interface:

- `SipServletRequest.getHeader`
- `SipServletRequest.getHeaders`
- `SipServletRequest.getHeaderNames`

These methods are syntactically and semantically identical to the methods provided by the HttpServletRequest interface. The getHeader method can be used for accessing singleton headers (Call-Id), whereas the getHeaders method can be used for accessing multiple headers (like Via) with the same name.

SIP headers are categorized into general-headers, request-headers, response-headers and entity-headers. A SIP request can contain general-headers, request-headers and entity-headers only. Entity-headers provide meta-information about the message-body. Request-headers are used by clients for passing additional information about the request that could not by conveyed by the request method. As described in the SIP specification, “these fields act as request modifiers, with semantics equivalent to the parameters of a programming language method invocation.

Additional convenience methods are also provided for frequently accessed headers. The following convenience methods are provided by the SipServletRequest interface for accessing common header fields:

- `SipServletRequest.getCallId`
- `SipServletRequest.getFrom`
- `SipServletRequest.getTo`

6. THE SDP SESSION DESCRIPTION

The Session Description Protocol (SDP) is used by SIP for conveying session information to prospective participants. It is purely a format for session description and not really a protocol. Unlike the H.245 specification, it is not meant for supporting media encoding and session content.

The SDP is more like a markup language for describing session information. It defines tags that can convey the following information related to session formation:

- Session name and purpose
- Time(s) the session is active
- The media comprising the session
- Information to receive those media (addresses, ports, formats, etc).
- Information about the bandwidth to be used by the conference
- Contact information for the person responsible for the session

Please refer to RFC 2327 for a detailed description of SDP.

We define two interfaces for encapsulating all the information related to session description, namely the SessionDescription and MediaDescription interfaces. The server populates these objects with the information elements contained in the SDP portion of the SIP message. SIP container implementers are responsible for providing objects that implement these interfaces.

7. THE SIP RESPONSE

Messages in SIP are (as in HTTP) either request messages or response messages. A response message consists of method tokens, message headers, and message body. The SIP response object encapsulates responses generated by SIP severs and SIP servlets. It acts like a container encapsulating all information to be sent to the SIP client.

7.1 Headers

A SIP servlet can set various headers and response codes via the SipServletResponse interface:

- `SipServletResponse.setHeader`
- `SipServletResponse.appendHeader`

Typically, the servlet writers use these methods for setting optional headers. For example, a mobility servlet may decide to set the Contact and Expires headers as part of a 302 Moved temporarily response.

Both these methods take two parameters, a name and a header value. The setHeader method sets the header specified by the name to the value parameter. It overwrites any previous values associated with the given header. The SipServletResponse.containsHeader method can be used to test the presence of a header before setting its value. The appendHeader method concatenates a header value to the existing header value. If the header specified does not exist, then this method will create a new header entry.

7.2 Convenience Methods

SIP support resource mobility via redirection response. SIP Servlet writers need to frequently generate redirection responses. The SipServletResponse interface provides the following convenience methods for generating error and redirection responses:

- `SipServletResponse.sendRedirect`
- `SipServletResponse.sendError`

The sendRedirect method is used for generating 3xx (Redirection) responses. This method will set the appropriate headers and content body for redirecting SIP clients to a different location.

The sendError method emits the appropriate headers and content body in the response message. It can be used for generating 4xx (Client-Error), 5xx (Server-Error), and 6xx (Global-Failure) responses.

7.3 Proxy Method

Like HTTP, SIP supports proxying functionality. For rapidly developing proxy services, the SipServletResponse interface provides the following convenience methods for proxying requests:

- `SipServletResponse.proxyStateless`
- `SipServletResponse.proxyStatefull`

The proxyStateless and proxyStatefull methods can be used for directing the server to carryout SIP proxy operation. These methods take two flags, ***fork***, and ***recurse***. The only difference between proxyStateless and proxyStatefull operation is that the proxyStatefull operation processes Record-Route, Route, and Session-Expires headers according to their draft specifications; the proxyStateless operation does not does not process these headers.

8. SESSIONS

The SIP servlet API provides a simple session tracking interface that allows SIP servlet containers to track a SIP session. SIP session tracking is straight forward, unlike the session tracking techniques in HTTP.

8.1.1 Session Tracking Mechanism

Since SIP is state-oriented protocol by design, it has build in capability for identifying a series of unique requests from a remote client. Every SIP request and response contains a Call-ID generic header field that is used for uniquely identifying a session.

8.1.2 Session Timeouts

The SIP sessions should timeout according to the default timers specified by the SIP specifications, regardless of the underlying transport protocol used (TCP or UDP). All SIP servlet containers must follow the default timeout and retry semantics specified by the SIP specification. A SIP servlet may specify a default timeout value as part of their deployment information, or may programmatically set the timeout values.

9. DISPATCHING REQUESTS

The ability to chain together SIP servlets when serving requests is under study.

Input is welcomed.

Please refer to [HTTP Servlet 8].

10. SIP APPLICATIONS

SIP Servlets can be used to implement several kinds of services. They include the following services:

- Location Manager
- Redirection Server
- Proxy Server, both call statefull and stateless servers
- Treatment Server

These services are discussed below.

10.1 Location Manager

A location manager tracks the contact information for users (supplied via REGISTER requests). The location manager's functionality can enhance the functionality of a redirection server and a proxy server, by enabling the users with a means to control redirection and proxying via the SIP protocol.

10.2 Redirection Server

A redirection server redirects incoming requests to another SIP UAS. The server can decide whether to accept and to redirect the request based upon some combination of stored data and algorithm(s) specified by a SIP servlet.

10.3 Proxy Server

A proxy server proxies each incoming requests to another SIP UAS. The server can decide whether to accept and to proxy a service request based upon some combination of stored data and algorithm(s) specified by a SIP servlet.

10.4 Announcement/Treatment Server

An announcement or treatment server is a server that can accept calls. The server connects the caller to a media connection that does not initiate the completion of the call. The caller can disconnect at anytime. This is useful for connecting callers to announcements or other types of sessions where interaction is not required.

11. MAPPING REQUESTS TO SERVLETS

Application developers can use SIP Servlets API for developing redirect, proxy, or termination side services. A SIP Servlet engine needs to de-multiplex incoming requests and dispatch them to the proper servlet.

The HTTP Servlet mapping scheme is based on the request URL paths. However, a SIP-URL does not contain a path and is used for identifying users instead of files. Further, since SIP Servlets API can be used for developing originating, terminating, and redirect services, a simple path based mapping scheme is not adequate.

SIP Servlet engines can use Request-URI (current destination), From header (the originator), and To header (final recipient of the request) to map requests to servlets. The container can use any or all of these three fields for mapping request to servlets. However, the current servlet mapping rules are based only on the Request-URI field and they are as follows:

- The servlet container will try to map by looking for an exact match on the user and hostport fields on the Request-URI. I.e., the current destination of the request. The search is terminated once an exact match is found and no further rules are attempted.
- The container will then try to map by looking for an exact match on the user and host fields of the Request-URI.
- The container will then try to map by looking for an exact match on the hostport filed of the Request-URI.
- The container will then try to map by looking for an exact match on the host filed of the Request-URI.
- The container will then try to map by looking for an exact match on the user field of the Request-URI.
- If none of the rules result in a servlet match, the container will emit a 5xx response.

Note: These mapping rules are preliminary and are subject to change based on the feedback and reference implementation experiences.

12. SECURITY

Security issues are under investigation.

Please refer to [HTTP Servlet 11].

13. APPLICATION PROGRAMMING INTERFACE

This section lists the set of interfaces, classes, and exceptions that compose the SIP Servlet API. The accompanying JavaDoc documentation contains a more detailed description of all these.

Table 1: Servlet and SIP Servlet Package Summary

Package javax.servlet	Package com.wcom.sip.servlet
RequestDispatcher	SipServletRequest
Servlet	SipServletResponse
ServletConfig	SipSession
ServletContext	SipSessionBindingListener
ServletRequest	SipServlet
ServletResponse	SIPURL
SingleThreadModel	MediaDescription
GenericServlet	SessionDescription
ServletInputStream	SipUtils
ServletException	Contact
UnavailableException	ContactAddress
	To

13.1 Package javax.servlet

The javax.servlet package is listed here only for completeness. This package is defined and controlled by JavaSoft.

13.1.1 Request Dispatcher

```
public interface RequestDispatcher
public void forward(ServletRequest request, ServletResponse
response);
public void include(ServletRequest request, ServletResponse
response);
```

13.1.2 Servlet

```
public interface Servlet

public void init(ServletConfig config) throws ServletException;
public ServletConfig getServletConfig();
public void service(ServletRequest req, ServletResponse res)
  throws ServletException, IOException;
public String getServletInfo();
public void destroy();
```

13.1.3 ServletConfig

```
public interface ServletConfig

public ServletContext getServletContext();
public String getInitParameter(String name);
public Enumeration getInitParameterNames();
```

13.1.4 ServletContext

```
public interface ServletContext

public ServletContext getContext(String uripath);
public int getMajorVersion();
public String getMimeType(String file);
public int getMinorVersion();
public URL getResource(String path) throws MalformedURLException;
public InputStream getResourceAsStream(String path);
public RequestDispatcher getRequestDispatcher(String urlpath);
public Servlet getServlet(String name) throws ServletException;
public Enumeration getServlets();
public Enumeration getServletNames();
public void log(String msg);
public void log(Exception exception, String msg);
public void log(String message, Throwable throwable);
public String getRealPath(String path);
public String getServerInfo();
public Object getAttribute(String name);
public Enumeration getAttributeNames();
public void setAttribute(String name, Object object);
public void removeAttribute(String name);
```

13.1.5 ServletRequest

```
public interface ServletRequest

public Object getAttribute(String name);
public Enumeration getAttributeNames();
public String getCharacterEncoding ();
public int getContentLength();
public String getContentType();
public ServletInputStream getInputStream() throws IOException;
public String getParameter(String name);
public Enumeration getParameterNames();
public String[] getParameterValues(String name);
public String getProtocol();
public String getScheme();
public String getServerName();
public int getServerPort();
public BufferedReader getReader () throws IOException;
public String getRemoteAddr();
public String getRemoteHost();
public void setAttribute(String key, Object o);
```

13.1.6 ServletResponse

```
public interface ServletResponse
```

```
public String getCharacterEncoding ();
public ServletOutputStream getOutputStream() throws IOException;
public PrintWriter getWriter() throws IOException;
public void setContentLength(int len);
public void setContentType(String type);
SingleThreadModel
public interface SingleThreadModel

// No Methods
```

13.1.7 GenericServlet

```
public abstract class GenericServlet
    implements Servlet, ServletConfig, java.io.Serializable

public GenericServlet ();
public void destroy();
public String getInitParameter(String name);
public Enumeration getInitParameterNames();
public ServletConfig getServletConfig();
public ServletContext getServletContext();
public String getServletInfo();
public void init(ServletConfig config) throws ServletException;
public void init() throws ServletException;
public void log(String msg);
public void log(String message, Throwable t);
public abstract void service(ServletRequest req, ServletResponse
res)
  throws ServletException, IOException;
```

13.1.8 ServletInputStream

```
public abstract class ServletInputStream extends InputStream

protected ServletInputStream ();
public int readLine(byte[] b, int off, int len) throws IOException;
```

13.1.9 ServletOutputStream

```
public abstract class ServletOutputStream extends OutputStream

protected ServletOutputStream ();
public void print(String s) throws IOException;
public void print(boolean b) throws IOException;
public void print(char c) throws IOException;
public void print(int i) throws IOException;
public void print(long l) throws IOException;
public void print(float f) throws IOException;
public void print(double d) throws IOException;
public void println() throws IOException;
public void println(String s) throws IOException;
public void println(boolean b) throws IOException;
public void println(char c) throws IOException;
public void println(int i) throws IOException;
public void println(long l) throws IOException;
public void println(float f) throws IOException;
```

```
public void println(double d) throws IOException;
```

13.1.10 ServletException

```
public class ServletException extends Exception

public ServletException();
public ServletException(String message);
public ServletException(String message, Throwable rootCause);
public ServletException(Throwable rootCause);
public Throwable getRootCause();
```

13.1.11 UnavailableException

```
public  class UnavailableException extends ServletException

public UnavailableException (Servlet servlet, String msg);
public UnavailableException (int seconds, Servlet servlet, String
msg);
public boolean isPermanent ();
public Servlet getServlet ();
public int getUnavailableSeconds ();
```

13.2 Package com.wcom.sip.servlet

```
interface SipServletRequest
interface SipServletResponse
interface SipSession
interface SipSessionBindingListner
interface Contact
interface ContactAddress
interface To

class SipServlet
class URI
class SessionDescription
class MediaDescription
class SipUtils
class SIPURL
```

13.2.1 SipServletRequest

```
public interface SipServletRequest extends ServletRequest

public SIPURL getCalledParty();
public SIPURL getFromURL();
public SIPURL getToURL();
public String getAuthType();
public String getCallId();
public long getDateHeader(String name);
public String getHeader(String name);
public int getIntHeader(String name);
public String getMethod();
public SessionDescription getSessionDescription();
public String getRemoteUser();
public String getRequestURI();
public String getServletInvocationPattern();
```

```
public SipSession getSession (boolean create);
public SipSession getSession();
public boolean isRequestedSessionIdValid ();
public String getBody();
```

13.2.2 SipServletResponse

```
public interface SipServletResponse extends ServletResponse

public static final int SC_TRYING;
public static final int SC_RINGING;
public static final int SC_CALL_BEGIN_FORWARDED;
public static final int SC_CALL_QUEUED;
public static final int SC_OK;
public static final int SC_MULTIPLE_CHOICES;
public static final int SC_MOVED_PERMANENTLY;
public static final int SC_MOVED_TEMPORARILY;
public static final int SC_SEE_OTHER;
public static final int SC_USE_PROXY;
public static final int SC_ALTERNATIVE_SERVICE;
public static final int SC_BAD_REQUEST;
public static final int SC_UNAUTHORIZED;
public static final int SC_PAYMENT_REQUIRED;
public static final int SC_BAD_FORBIDDEN;
public static final int SC_NOT_FOUND;
public static final int SC_METHOD_NOT_ALLOWED;
public static final int SC_PROXY_AUTHENTICATION_REQUIRED;
public static final int SC_REQUEST_TIMEOUT;
public static final int SC_CONFLICT;
public static final int SC_GONE;
public static final int SC_LENGTH_REQUIRED;
public static final int SC_REQUEST_ENTITY_TOO_LARGE;
public static final int SC_REQUEST_URI_TOO_LONG;
public static final int SC_UNSUPPORTED_MEDIA_TYPE;
public static final int SC_BAD_EXTENSION;
public static final int SC_TEMPORARLY_UNAVAILABLE;
public static final int SC_CALL_LEG_DNE;
public static final int SC_LOOP_DETECTED;
public static final int SC_TOO_MANY_HOPS;
public static final int SC_ADDRESS_INCOMPLETE;
public static final int SC_AMBIGUOUS;
public static final int SC_BUSY_HERE;
public static final int SC_SERVER_INTERNAL_ERROR;
public static final int SC_NOT_IMPLEMENTED;
public static final int SC_BAD_GATEWAY;
public static final int SC_SERVICE_UNAVAILABLE;
public static final int SC_GATEWAY_TIMEOUT;
public static final int SC_VERSION_NOT_SUPPORTED;
public static final int SC_BUSY_EVERYWHERE;
public static final int SC_DECLINE;
public static final int SC_DOES_NOT_EXIT_ANYWHERE;
public static final int SC_NOT_ACCEPTABLE;

public SIPURL getCalledParty();
public SIPURL getFromURL();
public SIPURL getToURL();
public String getAuthType();
```

```
public String getCallId();
public long getDateHeader(String name);
public String getHeader(String name);
public int getIntHeader(String name);
public String getMethod();
public SessionDescription getSessionDescription();
public String getRemoteUser();
public String getRequestURI();
public String getServletInvocationPattern();
public SipSession getSession (boolean create);
public SipSession getSession();
public boolean isRequestedSessionIdValid ();
public String getBody();
```

13.2.3 SipSession

```
public interface SipSession

public long getCreationTime ();
public String getId ();
public long getLastAccessedTime ();
public int getMaxInactiveInterval();
public Object getValue (String name);
public String [] getValueNames();
public void invalidate ();
public boolean isNew ();
public void putValue (String name, Object value);
public void removeValue (String name);
public void setMaxInactiveInterval(int interval);
```

13.2.4 SipSessionBindingListener

```
public interface SipSessionBindingListener extends EventListener

public void valueBound (SipSessionBindingEvent event);
public void valueUnbound (SipSessionBindingEvent event);
```

13.2.5 SipServlet

```
public abstract class SipServlet extends GenericServlet
    implements java.io.Serializable

public SipServlet ();
protected void doInvite (SipServletRequest req, SipServletResponse
resp) throws ServletException, IOException;
protected long getLastModified (SipServletRequest req);
protected void doAck (SipServletRequest req, SipServletResponse
resp) throws ServletException, IOException;
protected void doBye (SipServletRequest req, SipServletResponse
resp) throws ServletException, IOException;
protected void doCancel (SipServletRequest req, SipServletResponse
resp) throws ServletException, IOException;
protected void doRegister (SipServletRequest req,
SipServletResponse resp) throws ServletException, IOException;
protected void doOptions (SipServletRequest req, SipServletResponse
resp) throws ServletException, IOException;
```

```
protected void service (SipServletRequest req, SipServletResponse
resp) throws ServletException, IOException;
public void service(ServletRequest req, ServletResponse res)
  throws ServletException, IOException;
```

13.2.6 SipSessionBindingEvent

```
public class SipSessionBindingEvent extends EventObject

public SipSessionBindingEvent (SipSession session, String name);
public String getName ();
public SipSession getSession ();
```

13.2.7 SipUtils

```
public class SipUtils

public SipUtils();
static public Hashtable parseQueryString(String s);
static public Hashtable parsePostData(int len, ServletInputStream
in);
static private String parseName(String s, StringBuffer sb);
public static StringBuffer getRequestURL (SipServletRequest req);
```

13.2.8 MediaDescription

```
public interface MediaDescription

public final String getMediaType();
public final void setMediaType(String mediaType);
public final int getMediaPort();
public final void setMediaPort(int port);
public final int getPortCount();
public final void setPortCount(int portCount);
public final String getProtocol();
public final void setProtocol(String protocol);
public final Iterator getMediaFormatIter();
public final void addMediaFormat(String mediaFormat);
public final void clearMediaFormats();
public final String getMediaDescription();
public final void setMediaDescription(String mediaInfo);
public final Iterator getConnectionFieldIter();
public final void addConnectionField(ConnectionField
connectionField);
public final void clearConnectionFields();
public final Iterator getBandwidthFieldIter();
public final void addBandwidthField(String bandwidth);
public final void clearBandwidthFoelds();
public final String getKeyField();
public final void setKeyField(String keyField);
public final Iterator getAttributeIter();
public final void addAttribute(AttributeField attribute);
public final void clearAttributes();
```

13.2.9 SessionDescription

```
public interface SessionDescription

public int getSDPVersion();
public void setSDPVersion(int version);
public OriginField getOriginField();
public void setOriginField(OriginField origin);
public String getSessionName();
public void setSessionName(String name);
public String getSessionDescription();
public void setSessionDescription(String description);
public String getURI();
public void setURI(String uri);
public String getEmail();
public void setEmail(String email);
public String getPhoneNumber();
public void setPhoneNumber(String phone);
public ConnectionField getConnectionField();
public void setConnectionField(ConnectionField connectionField);
public TimeField getTimeField();
public void setTimeField(TimeField time);
public String getKeyField();
public void setKeyField(String keyField);
public String getBandwidthField();
public void setBandwidthField(String bandwidth);
public void addAttribute(AttributeField attribute);
public final void clearAttributes();
public Iterator getAttributsIter();
public void addMediaDescription(MediaDescription media);
public final void clearMediaDescriptions();
public Iterator getMediaDescriptionIter();
```

13.2.10 SIPURL

```
public class SIPURL

public final void setUser(String u);
public final String getUser();
public final boolean isUser();
public final boolean isPhone();
public final void setPhone();
public final void setPassword(String p);
public final String getPassword();
public final void setHost(String h);
public final String getHost();
public final void setPort(String p);
public final String getPort();
public final int getPortAsInt();
public final void addTransParam(String p);
public final String getTransParam(int i);
public final void addUserParam(String u);
public final String getUserParam(int i);
public final void addMethodParam(String m);
public final String getMethodParam(int i);
public final void addTtlParam(String t);
public final String getTtlParam(int i);
```

```
public final void addMaddrParam(String m);
public final void setMaddrParam(int i,String m);
public final String getMaddrParam(int i);
public final void addOtherParam(String key, String value);
public final void setOtherParams(Hashtable ht);
public final String getOtherParam(String m);
public final Hashtable getOtherParams();
public final void addHeader(String name, String value);
public final String getHeaderValue(String name);
```

13.2.11 Contact

```
public class Contact

public Vector getContactAddresses();
public void addContactAddress(ContactAddress a);
public void removeContactAddress(ContactAddress a);
```

13.2.12 ContactAddress

```
public class ContactAddress

public String getDisplayName();
public void setDisplayName(String dn);
public SIPURL getSIPURL();
public SIPURL setSIPURL(SIPURL url);
public String getQParam();
public void setQParam(String q);
public String getAction();
public void setAction(String a);
public String getExpire();
public void setExpire(String e);
public boolean isExpired();
```

13.2.13 To

```
public class To

public SIPURL getSIPURL();
public void setSIPURL(SIPURL url);
public String getTag();
public void setTag(String tag);
public boolean hasTag ();
public To getToWithoutTag ();
public String getDisplayName();
public void setDisplayName(String dn);
public void setParams(Hashtable params);
public Hashtable getParams();
```

14. DEPLOYMENT DESCRIPTOR

The deployment descriptor specifies how the container is configured.

14.1 Deployment Descriptor Elements

The following types of configuration and deployment information exist in the telephony application deployment descriptor:

- ServletContext Initialization parameters
- Localized Content
  - Local specific announcements
  - Local specific error messages and status messages.
- Session Configuration
- Servlet Definitions
- Servlet Mappings
- Mime Type Mappings
- Error Pages
- Security

See the accompanying DTD for further description of all the tags and their comments.

14.2 DTD

```
<!—
A tel-app is the root element of the deployment descriptor for a
telephony application (currently only SIP). It is a schema for
describing meta-information about a telephony application.
-->

<!ELEMENT tel-app (icon?,display-name, description?, distributable?,
context-param*, servlet*, servlet-mapping*, session-config?,
mime-mapping*, welcome-file-list?, error-message*,
resource-ref*, env-entry*, ejb-ref*)>

<!—
A telephony-application contains a small-icon and a large-icon element
which specific the location within the web application for a small and
large image used to represent the telephony application in a GUI tool
-->

<!ELEMENT icon (small-icon?, large-icon?)>

<!—
The small-icon element contains the location within the web application
of a file containing a small (16x16 pixel) icon image.  The image must
be either GIF or JPEG format and the filename must end with the
extension of ".gif" or ".jpg"
```

```
-->
<!ELEMENT small-icon (#PCDATA)>

<!—
The large-icon element contains the location within the web application
of a file containing a small (32x32 pixel) icon image.  The image must
be either GIF or JPEG format and the filename must end with the
extension of ".gif" or ".jpg"
-->

<!ELEMENT large-icon (#PCDATA)>

<!—
The display-name element contains a short name that is intended to be
displayed by
GUI tools
-->

<!ELEMENT display-name (#PCDATA)>

<!—
The description element is used to provide descriptive text about the
parent element
-->

<!ELEMENT description (#PCDATA)>

<!—
The distributable element, by its presence is a telephony application
deployment descriptor, indicates that this web application is
programmed appropriately to be deployed into a distributed servlet
container
-->

<!ELEMENT distributable EMPTY>

<!—
The context-param element contains the declaration of a telephony
application's servlet context initialization parameters.
-->

<!ELEMENT context-param (param-name, param-value, description?)>

<!—
The name of a configuration parameter
-->

<!ELEMENT param-name (#PCDATA)>

<!—
The value of a configuration parameter
-->

<!ELEMENT param-value (#PCDATA)>

<!—
```

```
The servlet element contains the declarative data of a servlet. A
telephony application may be composed out of serveral servlets.
-->

<!ELEMENT servlet (icon?, servlet-name, display-name, decscription?,
servlet-class, init-param*, load-on-startup?)>

<!—
The canonical name of the servlet
-->
<!ELEMENT servlet-name (#PCDATA)>

<!—
The class name of a servlet.
-->

<!ELEMENT servlet-class (#PCDATA)>

<!—
Contains a name/value pair as an initialization param of the servlet.
-->

<!ELEMENT init-param (param-name, param-value, description?)>

<!—
The presence of this tag indicates that this servlet should be loaded
on startup. The optional contents of these element must be an integer
indicating the order in which the servlet should be loaded.
-->

<!ELEMENT load-on-startup (#PCDATA)>

<!—
This tag defines a mapping between a servlet and the Request-URI.
-->

<!ELEMENT servlet-mapping (servlet-name, url-pattern)>

<!—
Defines the actual mapping.
-->

<!ELEMENT url-pattern (#PCDATA)>

<!—
Defines the session parameters for this web application.
-->

<!ELEMENT session-config (session-timeout)>

<!—
Defines the default session timeout interval for all sessions created
in this web application.  The default units are measured in minutes.
-->

<!ELEMENT session-timeout (#PCDATA)>
```

```
<!—
Defines a mapping between and extension and a mime type.
-->

<!ELEMENT mime-mapping (extension, mime-type)>

<!—
An extension.
-->

<!ELEMENT extension (#PCDATA)>

<!—
A defined mime type.
-->

<!ELEMENT mime-type (#PCDATA)>

<!—
A ordered list of greetings.
-->

<!ELEMENT welcome-file-list (welcome-file+)>

<!—
The welcome-file element contains file name to use as a default
welcome file, such as trying.sip.
-->

<!ELEMENT welcome-file (#PCDATA)>

<!--
Contains a mapping between an error code or exception type to the path
of a resource in the telephony application.
-->

<!ELEMENT error-message ((error-code | exception-type), location)>

<!—
The SIP error code, example: 405.
-->

<!ELEMENT error-code (#PCDATA)>

<!—
A Java exception type.
-->

<!ELEMENT exception-type (#PCDATA)>

<!—
The location of the resource in the telephony application.
-->

<!ELEMENT location (#PCDATA)>

<!—
```

```
The optional resource-ref element contains a declaration of a telephony
application reference to an external resource.
-->

<!ELEMENT resource-ref (description?, res-ref-name, res-type, res-
auth)>

<!—
Specifies the name of the resource factory reference name.  This value
is the name of the context-param whose value contains the JNDI name of
the data source.
-->

<!ELEMENT resource-ref-name (#PCDATA)>

<!—
Specifies the (Java class) type of the data source.
-->

<!ELEMENT resource-type (#PCDATA)>

<!—
The res-auth element indicates whether the application component
code performs resource signon programmatically or whether the
container signs onto the resource based on the principle mapping
information supplied by the deployer. Must be CONTAINER or SERVLET.
-->

<!ELEMENT res-auth (#PCDATA)>

<!—
The env-entry element contains the declaration of an
application's environment entry. This element is required to be
honored on in J2EE compliant servlet containers.
-->

<!ELEMENT env-entry (description?, env-entry-name, env-entry-value?,
    env-entry-type)>

<!—
The env-entry-name contains the name of an application's
environment entry.
-->

<!ELEMENT env-entry-name (#PCDATA)>

<!—
The env-entry-value element contains the value of an
application's environment entry.
-->

<!ELEMENT env-entry-value (#PCDATA)>

<!—
The env-entry-type element contains the fully qualified Java type
of the environment entry value that is expected by the application
code. The following are the legal values of env-entry-type:
```

```
java.lang.Boolean, java.lang.String, java.lang.Integer,
java.lang.Double, java.lang.Float.
-->

<!ELEMENT env-entry-type (#PCDATA)>

<!—
The ejb-ref element is used to declare a reference to an
enterprise bean.
-->

<!ELEMENT ejb-ref (description?, ejb-ref-name, ejb-ref-type, home,
    remote, ejb-link?)>

<!—
The ejb-ref-name element contains the name of an EJB
reference. This is the JNDI name that the servlet code uses to get a
reference to the enterprise bean.
-->

<!ELEMENT ejb-ref-name (#PCDATA)>

<!—
The ejb-ref-type element contains the expected Java class type of
the referenced EJB.
-->

<!ELEMENT ejb-ref-type (#PCDATA)>

<!—
The ejb-home element contains the fully qualified name of the
EJB's home interface.
-->

<!ELEMENT home (#PCDATA)>

<!—
The ejb-remote element contains the fully qualified name of the
EJB's remote interface.
-->

<!ELEMENT remote (#PCDATA)>

<!—
The ejb-link element is used in the ejb-ref element to specify
that an EJB reference is linked to an EJB in an encompassing Java2
Enterprise Edition (J2EE) application package. The value of the
ejb-link element must be the ejb-name of and EJB in the J2EE
application package.
-->

<!ELEMENT ejb-link (#PCDATA)>
```

14.3 Example

The following example illustrates an instantiation of the above DTD.

```
<tel-app>
  <display-name>A Simple Mobility Application</display-name>
  <session-timeout>30</session-timeout>
  <servlet>
     <servlet-name>mobility</servlet-name>
     <servlet-class>
        com.wcom.sip.servlet.examples.SimpleMobilityServlet
     </servlet-class>
     <init-param>
        <param-name>foo</param-name>
        <param-value>foo</param-value>
     </init-param>
  </servlet>
  <servlet-mapping>
     <servlet-name>mobility</servlet-name>
     <urlpatterns>
        <invite>adeo@sip.wcom.com</invite>
     </urlpatterns>
  </servlet-mapping>
</tel-app>
```

15. FUTURES

Contributors have made many excellent suggestions for additions to this specification. The authors will attempt to incorporate these suggestions in later versions of this document.

16. AUTHORS' ADDRESSES

Ajay P. Deo
MCIWorldCom, Inc
901 International Parkway
Dept 1731/678
Richardson, TX 75081
USA
Electronic mail: adeo@nettaxi.com

Kelvin R. Porter
MCIWorldCom, Inc
901 International Parkway
Dept 1731/678
Richardson, TX 75081
USA
Electronic mail: kelvin_porter@yahoo.com

Mark Johnson
MCIWorldCom, Inc
901 International Parkway
Dept 1731/678
Richardson, TX 75081
USA
Electronic mail: mark.x.johnson@wcom.com