HP-UX Java SIP Stack Programmer's Guide
HP-UX 11i v3
© Copyright 2007–2008 Hewlett-Packard Development Company, L.P.
Confidential computer software. Valid license from HP required for possession, use or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor’s standard commercial license.

The information contained herein is subject to change without notice. The only warranties for HP products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. HP shall not be liable for technical or editorial errors or omissions contained herein.

UNIX is a registered trademark of The Open Group.

Java is a registered trademark of Sun Microsystems, Inc.
# Table of Contents

**About This Document**

- Intended Audience.................................................................11
- Document Organization..........................................................11
- Typographical Conventions.......................................................12
- Related Information...............................................................12
- HP Encourages Your Comments..............................................15

**1 Introduction**

- SIP Overview...........................................................................17
- SIP Components.......................................................................18
  - User Agent...........................................................................18
  - Proxy Server.......................................................................19
  - Redirect Server...................................................................19
  - Registrar.............................................................................20
  - Back 2 Back User Agent......................................................20
- SIP Architecture.......................................................................20
- SIP Messages...........................................................................22
  - Message Types.....................................................................22
    - A Request Message...........................................................22
    - A Response Message.........................................................24
  - Message Parts.....................................................................28
- Call Flow Scenarios...............................................................29
  - Basic Call Setup and Teardown.............................................29
  - Using the Proxy Server.......................................................30
  - Using the Redirect Server...................................................32
- Implementations of SIP on HP-UX...........................................33
  - Documentation for the C SIP Stack.....................................33
  - Documentation for the Java SIP Stack.................................33

**2 JSR32 JAIN SIP APIs**

- Overview..................................................................................35
- JSR32 Architecture....................................................................36
  - JSR32 SIP Stack.................................................................37
  - JSR32 Packages.....................................................................39
  - JSR32 Objects.....................................................................39
    - SipStack Object............................................................39
    - Dialog Object..................................................................40
    - Transaction Object......................................................41
    - Message Object............................................................41
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>JSR32 Factories</td>
<td>41</td>
</tr>
<tr>
<td>HP-UX JSR32 Features</td>
<td>42</td>
</tr>
<tr>
<td>Event Notification</td>
<td>42</td>
</tr>
<tr>
<td>The REFER Method</td>
<td>43</td>
</tr>
<tr>
<td>SIP for Telephones</td>
<td>43</td>
</tr>
<tr>
<td>Provisional Response Acknowledgement in SIP</td>
<td>43</td>
</tr>
<tr>
<td>E.164 Number Mapping</td>
<td>44</td>
</tr>
<tr>
<td>Parsing Different URIs</td>
<td>44</td>
</tr>
<tr>
<td>Additional SIP Extensions</td>
<td>44</td>
</tr>
<tr>
<td>Persistent Connection</td>
<td>45</td>
</tr>
<tr>
<td>Transport Support</td>
<td>45</td>
</tr>
<tr>
<td>Support for IPv6 Addresses</td>
<td>45</td>
</tr>
<tr>
<td>Configuration Parameters</td>
<td>45</td>
</tr>
<tr>
<td>3 Creating an Application</td>
<td>51</td>
</tr>
<tr>
<td>Creating an Instance of the SIP Stack</td>
<td>51</td>
</tr>
<tr>
<td>Creating Instances of Factory Implementations</td>
<td>52</td>
</tr>
<tr>
<td>Creating an Instance of the SipProvider</td>
<td>52</td>
</tr>
<tr>
<td>Forming a SIP Message</td>
<td>53</td>
</tr>
<tr>
<td>Sending a SIP Message</td>
<td>53</td>
</tr>
<tr>
<td>Processing an Event</td>
<td>54</td>
</tr>
<tr>
<td>4 Compiling and Running Applications Using the HP-UX JSR32 Kit</td>
<td>55</td>
</tr>
<tr>
<td>5 Troubleshooting HP JSR32 APIs</td>
<td>57</td>
</tr>
<tr>
<td>Glossary</td>
<td>59</td>
</tr>
<tr>
<td>Index</td>
<td>61</td>
</tr>
</tbody>
</table>
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-1</td>
<td>SIP Protocol Structure</td>
<td>21</td>
</tr>
<tr>
<td>1-2</td>
<td>Establishing and Terminating a Call</td>
<td>30</td>
</tr>
<tr>
<td>1-3</td>
<td>Call Setup Using a Proxy server</td>
<td>31</td>
</tr>
<tr>
<td>1-4</td>
<td>Call Setup Using a Redirect Server</td>
<td>32</td>
</tr>
<tr>
<td>2-1</td>
<td>JSR32 SIP Architecture</td>
<td>37</td>
</tr>
<tr>
<td>Table</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>-------</td>
<td>------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>1-1</td>
<td>Response Types</td>
<td>25</td>
</tr>
<tr>
<td>1-2</td>
<td>Specific Response Types</td>
<td>26</td>
</tr>
<tr>
<td>1-3</td>
<td>Message Parts</td>
<td>29</td>
</tr>
<tr>
<td>2-1</td>
<td>SIP Stack Parameters</td>
<td>45</td>
</tr>
<tr>
<td>5-1</td>
<td>JSR32 API Exceptions</td>
<td>57</td>
</tr>
</tbody>
</table>
List of Examples

3-1 Setting the TRACE_LEVEL Stack Parameter ................................................................. 52
3-2 Sample Method for Creating a Listening Point .......................................................... 52
3-3 Creating a SipProvider ............................................................................................... 52
4-1 Compiling an Application ......................................................................................... 55
4-2 Running an Application ......................................................................................... 55
About This Document

This document discusses the Session Initiation Protocol (SIP) and the Java™ Specification Request 32 (JSR32) APIs. It describes how to create, compile, and run applications that use the JSR32 APIs on systems running HP-UX 11i v3. It also describes how to troubleshoot exceptions raised by the JSR32 APIs.

In addition to the Java SIP stack, the HP implementation of SIP is also available on the C SIP stack. For information on the HP implementation of the C SIP stack, see “Related Information” (page 12).

The document printing date and part number indicate the document’s current edition. The printing date will change when a new edition is printed. Minor changes may be made at reprint without changing the printing date. The document part number will change when extensive changes are made.

The latest version of the document is available at:

http://www.docs.hp.com

Document updates can be issued between editions to correct errors or document product changes. To ensure that you receive the updated or new edition, subscribe to the appropriate support service.

Contact your HP sales representative for details.

Intended Audience

This document is intended for application developers who write programs using JSR32 APIs. Application developers must be familiar with Java, networking concepts, and operating system concepts. For more information on JSR32, HP recommends that application developers read the relevant JSR32 APIs. This document is not a tutorial.

Document Organization

The HP-UX Java SIP Stack Programmer’s Guide is organized as follows:

Chapter 1 Chapter 1 (page 17) introduces the SIP protocol. It also discusses the SIP protocol architecture, components, message types, call flow scenarios, and implementations on HP-UX.

Chapter 2 Chapter 2 (page 35) introduces the JSR32 SIP APIs. It also discusses the architecture and features of JSR32 APIs.

Chapter 3 Chapter 3 (page 51) describes how to create an application.

Chapter 4 Chapter 4 (page 55) how to compile and run applications that use the JSR32 APIs.

Chapter 5 Chapter 5 (page 57) discusses the exceptions raised by the JSR32 APIs.
Typographical Conventions

This document uses the following typographical conventions:

audit(5)  An HP-UX manpage. The name of the manpage is audit and 5 is the section in the HP-UX Reference. On the web and on the Instant Information CD, it may be a link to the manpage itself. From the HP-UX command line, you can enter “man audit” or “man 5 audit” to view the manpage. See man(1).

Book Title  The title of a book. On the web and on the Instant Information CD, it may be a link to the book itself.

KeyCap  The name of a keyboard key. Note that Return and Enter both refer to the same key.

Emphasis  Text that is emphasized.

Emphasis  Text that is strongly emphasized.

Term  The defined use of an important word or phrase.

ComputerOut  Text displayed by the computer.

UserInput  Commands and other text that you type.

Command  A command name or qualified command phrase.

Variable  The name of a variable that you may replace in a command or function or information in a display that represents several possible values.

[ ]  The contents are optional in formats and command descriptions.

{}  The contents are required in formats and command descriptions. If the contents are a list separated by @, you must choose one of the items

...  The preceding element may be repeated an arbitrary number of times.

|  Separates items in a list of choices.

Related Information

The following related documents are available for the SIP product:

- HP-UX SIP Release Notes at:

- HP-UX C SIP Stack Programmer’s Guide at:

- HP-UX C SIP Stack Reference Guide at:
- **HP-UX C SIP Stack Message Layer Reference Guide at:**

- **Request for Comments (RFC) documents:**
  - RFC 1951 (*DEFLATE Compressed Data Format version 1.3*)
  - RFC 2052 (*A DNS RR for specifying the location services (SRV]*)
  - RFC 2246 (*TLS Protocol Version 1.0*)
  - RFC 2327 (*SDP: Session Description Protocol*)
  - RFC 2617 (*HTTP Authentication: Basic and Digest Access Authentication*)
  - RFC 2806 (*URLs for Telephone Calls*)
  - RFC 2848 (*The PINT Service Protocol: Extensions to SIP and SDP for IP Access to Telephone Call Services*)
  - RFC 2915 (*The Naming Authority Pointer (NAPTR) DNS Resource Record*)
  - RFC 2916 (*E.164 number and DNS*)
  - RFC 2976 (*The SIP INFO Method*)
  - RFC 3087 (*Control of Service Context using SIP-Request URI*)
  - RFC 3258 (*Distributing Authoritative Name Servers via Shared Unicast Addresses*)
  - RFC 3261 (*SIP: Session Initiation Protocol*)
  - RFC 3262 (*Reliability of Provisional Response in the Session Initiation Protocol (SIP]*)
— RFC 3263 (Session Initiation Protocol (SIP): Locating SIP Servers)
  http://www.ietf.org/rfc/rfc3263.txt?number=3263

— RFC 3264 (An Offer/Answer Model with the Session Description Protocol (SDP))

— RFC 3265 (Session Initiation Protocol (SIP) - Specific Event Notification)
  http://www.ietf.org/rfc/rfc3265.txt?number=3265

— RFC 3311 (The Session Initiation Protocol (SIP) UPDATE Method)
  http://www.ietf.org/rfc/rfc3311.txt?number=3311

— RFC 3312 (Integration of Resource Management and Session Initiation Protocol (SIP))
  http://www.ietf.org/rfc/rfc3312.txt?number=3312

— RFC 3313 (Private Session Initiation Protocol (SIP) Extensions for Media Authorization)
  http://www.ietf.org/rfc/rfc3313.txt?number=3313

— RFC 3323 (A Privacy Mechanism for the Session Initiation Protocol (SIP))
  http://www.ietf.org/rfc/rfc3323.txt?number=3323

— RFC 3325 (Private Extensions to the Session Initiation Protocol (SIP) for Asserted Identity within Trusted Networks)
  http://www.ietf.org/rfc/rfc3325.txt?number=3325

— RFC 3326 (The Reason Header Field for the Session Initiation Protocol (SIP))
  http://www.ietf.org/rfc/rfc3326.txt?number=3326

— RFC 3327 (Session Initiation Protocol (SIP) Extension Header Field for Registering Non-Adjacent Contacts)

RFC 3428 (Session Initiation Protocol (SIP) Extension for Instant Messaging)

RFC 3515 (The Session Initiation Protocol (SIP) Refer Method)

RFC 3581 (An Extension to the Session Initiation Protocol (SIP) for Symmetric Response Routing)

RFC 3608 (Session Initiation Protocol (SIP) Extension Header Field for Service Route Discovery During Registration)

RFC 3665 (Session Initiation Protocol (SIP) Basic Call Flow Examples)

RFC 3903 (Session Initiation Protocol (SIP) Extension for Event State Publication)
HP Encourages Your Comments

HP encourages your comments concerning this document. We are committed to providing documentation that meets your needs. Send any errors found, suggestions for improvement, or compliments to:

feedback@fc.hp.com

Include the document title, manufacturing part number, and any comment, error found, or suggestion for improvement you have concerning this document.
1 Introduction

With the increase in the mobility of people and the number of mobile devices that they use, it is important to stay connected irrespective of the geographical location of a person and the mobile device used. The Session Initiation Protocol (SIP) provides converged communication between various mobile devices and enables seamless data and voice transfer. This protocol establishes a session between the two end points in the network and the actual data is transferred using protocols, such as Real-Time Transport Protocol (RTP), Real-Time Streaming Protocol (RTSP) and the Media Gateway Control Protocol (MEGACO), and the Session Description Protocol (SDP). SIP also enables personal mobility so that users can maintain a single externally visible identifier irrespective of their geographical location.

This chapter discusses the SIP architecture and the various components in the SIP infrastructure that enable it to facilitate users to remain connected in different network locations.

This chapter addresses the following topics:

- “SIP Overview”
- “SIP Components” (page 18)
- “SIP Architecture” (page 20)
- “SIP Messages” (page 22)
- “Call Flow Scenarios” (page 29)
- “Implementations of SIP on HP-UX” (page 33)

SIP Overview

SIP is an application layer signaling protocol, developed by the Internet Engineering Task Force (IETF), that enables the creation, modification, and termination of sessions that are independent of the underlying transport protocols and session type being established.

Like hypertext transfer protocol (HTTP), SIP is a client-server protocol, in which requests are issued by the clients and responses are managed by the servers. Clients and servers exchange messages that contain the information required for establishing a session.

The SIP infrastructure includes network hosts called the proxy server to which user agents can send registrations, invitations to sessions, and other requests to locate session participants and for other functions. The infrastructure also contains entities such as redirect server, registrar, and location service to redirect requests, alter the contact address, and locate an address. These entities along with the name mapping service in SIP support personal mobility and enable users to maintain a single externally visible identifier, regardless of the network location of the user. SIP also contains a back 2 back user agent (B2BUA) that is similar to the proxy server. However, B2BUA serves as both a user agent client and a user agent server.
SIP supports the following while establishing and terminating sessions between user agents:

- Determines the end system to be used for communication.
- Determines the willingness of the called user to participate in a call.
- Determines the media and media parameters to be used for a particular session.
- Establishes session parameters at both the ends of a session.
- Manages a session by including transfer and termination of sessions, modifying session parameters, and invoking services.

Following are the salient features of SIP:

- Session-agnostic
- Transport independent
- Provides user mobility
- Provides stack configuration capabilities

**SIP Components**

This section discusses the components of the SIP infrastructure. The SIP infrastructure contains the following components:

- User Agent
- Proxy Server
- Redirect Server
- Registrar Server
- Back 2 Back User Agent

This section addresses the following topics:

- “User Agent” (page 18)
- “Proxy Server” (page 19)
- “Redirect Server” (page 19)
- “Registrar” (page 20)
- “Back 2 Back User Agent” (page 20)

**User Agent**

A user agent (UA) is an application that can initiate, receive, and terminate a call in a SIP session. The user agent initiating a call acts as a client while sending the initial INVITE request and acts as a server while receiving a BYE request from the client. In a SIP session, a user agent can be any end-user device, such as a cell phone, multimedia handset, personal computer (PC), personal digital assistant (PDA), or a softphone.
A user agent can function in the following roles:

- **User Agent Client**
  A user agent client (UAC) is a client application that initiates a new SIP request. The application acts as a UAC only for the duration of the transaction when it sends a SIP request. If the application receives a request, UAC assumes the role of a user agent server for processing that transaction. The requests issued by a UAC are INIVITE, ACK, OPTIONS, BYE, CANCEL, and REGISTER.

- **User Agent Server**
  A user agent server (UAS) is a server application that receives a request, processes the request, and generates a response. An application acts as a UAS only for the duration of the transaction when it receives a SIP response. If the application sends a request, UAS assumes the role of a user agent server for processing that transaction.

  A SIP end point can perform the role of both UAC and UAS. However, during a transaction, it can perform only one role. Depending on the UA that initiated the request, an end point functions as a UAC or a UAS.

**Proxy Server**

A proxy server is an intermediate entity that receives SIP requests and forwards them on behalf of the requestor. Proxy servers route requests to the user's current location, authenticate and authorize users for services, and implement provider call-routing policies. It also enforces polices for a user and provides name mapping.

For each new request, a proxy can function either in stateful or stateless mode. When it functions as a stateless proxy server, it acts as a simple forwarding element. A stateless proxy server discards information about a message after the message is forwarded. A stateful proxy server remembers information (specifically, transaction state) about each incoming request and also about requests it sends as a result of processing the incoming request. It uses this information to influence the processing of future messages associated with that request.

SIP proxy servers use presence services to track users and to check the willingness and ability of users to communicate with other users on the network. A presence service is an application of the SIP-specific event notification mechanism (RFC 3265 *(Session Initiation Protocol (SIP) - Specific Event Notification)*). It enables a user to be located regardless of the physical location. It also accepts presence information from users, stores, and distributes the presence information to other interested users.

**Redirect Server**

A redirect server accepts a SIP request, maps the address into zero or more new addresses, and returns these addresses to the client. The redirect server uses the location service to obtain information about a user's possible locations. Unlike a proxy server,
the redirect server does not initiate its own SIP request. Instead, it redirects the request back to the client indicating that the client must attempt a different route to reach the recipient. This method of identifying a client irrespective of its location enables a client to move geographically and still remain contactable with the same SIP identity. Unlike a user agent server, the redirect server does not accept or terminate calls.

Registrar

The registrar server enables clients to alter the address at which they can be contacted. A SIP client can send a REGISTER request to inform the registrar about the change in the address. The registrar accepts the change and records the user's new address. Thus, the information about the client's location is kept updated, even though the client changes locations in the network. The registrar server also supports authentication.

A client must register with the registrar server each time a user turns on the SIP user client, using a SIP IP phone, PC, laptop, or other SIP devices. Typically, the proxy or registration server forwards the registration information to the location or the redirect server. These registrations can occur at any time during a session and are also periodically refreshed so that most recent information is stored in the location or the redirect server. Therefore, a registrar server is co-located with a proxy or redirect server to offer the location services.

Back 2 Back User Agent

A back 2 back user agent (B2BUA) serves both as a user agent server and a user agent client. It is a logical entity that receives a request and processes it as a user agent server. It acts as a user agent client to determine how the request must be handled.

Unlike a proxy server, it maintains the dialog state and participates in all requests sent on the dialogs that it has established. The responsibility of a proxy server is limited to routing messages. However, a B2BUA performs additional tasks, such as topology hiding and network termination of dialogs. The B2BUA can also alter the messages to suit different applications, such as prepaid charging, and customer care applications.

SIP Architecture

This section discusses the SIP architecture and the different layers in the SIP protocol. It also illustrates the SIP protocol structure.

Figure 1-1 illustrates the building blocks in the SIP protocol architecture and the position of SIP in the IP stack.
Figure 1-1 SIP Protocol Structure

- The application layer provides application programs with an interface to communicate and transfer data across the network.
- Following are the SIP protocol layers:
  - A transaction user (TU) can be any SIP entity except a stateless proxy. A transaction user uses transactions to send a request to the peer. It creates a client transaction and sends the request, the destination IP address, port number, and transport service to which the request must be sent.
  - The transaction layer handles application layer retransmissions and timeouts, and matches responses to requests. Any task that a user agent client accomplishes takes place using a series of transactions.
  - All SIP components contain a transport layer to send requests and responses over network transports. For connection-oriented transports, the transport layer determines the type of connection to use for a request or response. The network transport can be Transmission Control Protocol (TCP), User Datagram Protocol (UDP), Transport Layer Security (TLS), or Stream Control Transmission Protocol (SCTP). The transport layer also determines how a client can send requests and responses, and how a server can receive requests and responses over the network.
• The network transport layer is the transport layer in the IP stack. It enables transfer of data between end points by using the services of the network layer. This layer has two primary protocols, TCP and UDP. The TCP supports reliable and sequential packet delivery through error recovery and flow control mechanisms. The UDP is a simple message-based connectionless protocol compared to TCP. The SCTP is yet another transport layer protocol that application developers can use to transmit data between end points.

• The network layer routes data packets from the sender to the receiver in the network. The most common network layer protocol is IP.

SIP Messages

A SIP message is a simple text message that is similar to an HTTP message. It is either a request from a client to a server, or a response from a server to a client. It uses the client-server model to send a request and to receive a response. The request messages are in the form of nouns, for example, INVITE, BYE, and so on. The response messages are always in the numerical form, for example, 180, 200, or 404. Following is the format of a SIP message:

<Header_name> : <Description>

This section briefly describes the different types of SIP messages, parts of a message, and sample messages.

This section addresses the following topics:

• “Message Types” (page 22)
• “Message Parts” (page 28)

Message Types

Following are the SIP message types:

• Request
• Response

The request and response messages use the same basic format. Both types of messages consist of a start-line, one or more header fields, an empty line indicating the end of header fields, and an optional message-body. However, the syntax differs in character set and syntax specifics.

A Request Message

A request is a message sent from the client to the server. Following is a sample request message:

INVITE sip:tpu@hp.com SIP/2.0
Via: SIP/2.0/UDP local.hp.com
From: OC <sip:OpenCall.SIP@hp.com>
To: TPU <sip:tpu@hp.com>
Subject: Confcall
Call-ID: 132059753@local.hp.com
Content-Type: application/sdp
CSeq: 1 INVITE
Contact: <sip:telecom@16.188.155.140>
Content-Length: 187

v=0
o=user1 51633745 1348648134 IN IP4 16.188.155.140
s=Interactive Conference
c=IN IP4 224.2.4.4/127
t=0 0
m=audio 3456 RTP/AVP 0 22
a=rtpmap:22 application/g723.1

The following are the different parts of a request:

First Line
Contains the method name, such as INVITE.

Via
Indicates the transport used for transaction and contains the address at which the caller is expecting responses to the request.

From
Contains a display name, and a SIP or secure SIP (SIPS) uniform resource identifier (URI) that indicates the originator of the request.

To
Contains the display name (TPU) and a SIP or a SIPS URI towards which the request was originally directed.

Subject
Contains the subject line for the call

Call ID
Contains a globally unique identifier for this call generated by the combination of a random string and the end point’s host name or IP address.

Content-Type
Contains a description of the message body.

Command sequence (CSeq)
Contents an integer and a method name.

Contact
Contains a SIP or SIPS URI that represents a direct route to contact the end point. Contact usually contains a username that is a fully qualified domain name.

Content-Length
Contains an octet (byte) count of the message body.

A valid SIP request formulated by a user agent client must contain the following header fields: To, From, CSeq, Call-ID, Max-Forwards, and Via. These header fields are the fundamental building blocks of a SIP message, because they provide most of the critical message routing services including addressing of messages, routing of responses, limiting message propagation, ordering of messages, and the unique identification of transactions.

Following are the SIP Request method names, as defined in RFC 3261 (Session Initiation Protocol):
ACK Confirms that the client has received a final response to an INVITE request.

BYE Indicates to the server that the client wants to release the call.

CANCEL Cancels a pending user agent client request.

INVITE Indicates that the user or service is being invited to participate in a session.

OPTIONS Queries a server with regard to its capabilities.

REGISTER Registers contact information with a SIP server.

JAIN SIP also supports the following request method name extensions:

INFO Carries session-related control information that is generated during a session, as specified in RFC 2976 (The SIP INFO Method).

PRACK Confirms that the client has received a final response to an INVITE. PRACK is similar to ACK, but specific to the reliability of provisional responses. The PRACK functionality is defined in RFC 3262 (Reliability of Provisional Response in the Session Initiation Protocol (SIP)).

UPDATE Enables a client to update parameters of a session without impacting the state of a dialog. This functionality is defined in RFC 3311 (The Session Initiation Protocol (SIP) UPDATE Method).

SUBSCRIBE Provides an extensible framework by which SIP nodes can request notification from remote nodes indicating that certain events have occurred. This functionality is defined in RFC 3265 (Session Initiation Protocol (SIP) - Specific Event Notification).

NOTIFY Provides an extensible framework by which SIP nodes can receive notifications from remote nodes indicating that certain events have occurred. This functionality is defined in RFC 3265 (Session Initiation Protocol (SIP) - Specific Event Notification).

MESSAGE Used for sending instant messages. This functionality is defined in RFC 3428 (Session Initiation Protocol (SIP) Extension for Instant Messaging).

REFER Requests that the recipient must refer to a resource provided in the request. This functionality is defined in RFC 3515 (The Session Initiation Protocol (SIP) Refer Method).

A Response Message

A response is a message sent by the recipient of a request after it has received and interpreted the request. A response contains the following information:

- **Status Code**: Specifies a 3-digit integer that indicates the result of the response.
- **Reason phrase**: Provides a short textual description of the status code.
- **Headers**: Specifies the header information.
- **Message body**: Contains the actual information sent in the response.

Following is a sample response message:
The status line at the beginning of the response message differentiates this message from a request message. This line consists of the protocol version followed by a numeric status-code and its associated textual phrase, with each element separated by a single space (SP) character.

The status line contains the following information: SIP version, SIP status code, and the SIP reason phrase. The first digit of the status code defines the class of response. The last two digits do not carry any specific meaning, but the range of the status code defines the type of response code. For example, if the response code is within the range 100 and 199, it is a 1xx response and this response is called a provisional response.

Table 1-1 discusses the response types.

<table>
<thead>
<tr>
<th>Response Code</th>
<th>Reason Phrase</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1xx</td>
<td>Provisional</td>
<td>Specifies that the request is received and the server is processing the request.</td>
</tr>
<tr>
<td>2xx</td>
<td>Success</td>
<td>Specifies that the action was successfully received, understood, and accepted by the server.</td>
</tr>
<tr>
<td>3xx</td>
<td>Redirection</td>
<td>Specifies that further action needs to be taken in order to complete the request.</td>
</tr>
<tr>
<td>4xx</td>
<td>Client error</td>
<td>Specifies that the request contains a bad syntax or that the request cannot be fulfilled at the server.</td>
</tr>
<tr>
<td>5xx</td>
<td>Server error</td>
<td>Specifies that the server failed to fulfill a valid request.</td>
</tr>
<tr>
<td>6xx</td>
<td>Global failure</td>
<td>Specifies that the request cannot be fulfilled at any server.</td>
</tr>
</tbody>
</table>

Table 1-2 discusses the response codes for each response type.
### Table 1-2 Specific Response Types

<table>
<thead>
<tr>
<th>Response Code</th>
<th>Description</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>1xx (Provisional)</strong></td>
<td>TRYING</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>RINGING</td>
<td>180</td>
</tr>
<tr>
<td></td>
<td>CALL_IS_BEING_FORWARDED</td>
<td>181</td>
</tr>
<tr>
<td></td>
<td>QUEUED</td>
<td>182</td>
</tr>
<tr>
<td></td>
<td>SESSION_PROGRESS</td>
<td>183</td>
</tr>
<tr>
<td><strong>2xx (Success)</strong></td>
<td>OK</td>
<td>200</td>
</tr>
<tr>
<td></td>
<td>ACCEPTED</td>
<td>202</td>
</tr>
<tr>
<td><strong>3xx (Redirection)</strong></td>
<td>MULTIPLE_CHOICES</td>
<td>300</td>
</tr>
<tr>
<td></td>
<td>MOVED_PERMANENTLY</td>
<td>301</td>
</tr>
<tr>
<td></td>
<td>MOVED_TEMPORARILY</td>
<td>302</td>
</tr>
<tr>
<td></td>
<td>USE_PROXY</td>
<td>305</td>
</tr>
<tr>
<td></td>
<td>ALTERNATIVE_SERVICE</td>
<td>380</td>
</tr>
<tr>
<td>Response Code</td>
<td>Description</td>
<td>Specific-Response Type</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------</td>
<td>------------------------</td>
</tr>
<tr>
<td>400</td>
<td>BAD_REQUEST</td>
<td>400</td>
</tr>
<tr>
<td>401</td>
<td>UNAUTHORIZED</td>
<td>401</td>
</tr>
<tr>
<td>402</td>
<td>PAYMENT_REQUIRED</td>
<td>402</td>
</tr>
<tr>
<td>403</td>
<td>FORBIDDEN</td>
<td>403</td>
</tr>
<tr>
<td>404</td>
<td>NOT_FOUND</td>
<td>404</td>
</tr>
<tr>
<td>405</td>
<td>METHOD_NOT_ALLOWED</td>
<td>405</td>
</tr>
<tr>
<td>406</td>
<td>NOT_ACCEPTABLE</td>
<td>406</td>
</tr>
<tr>
<td>407</td>
<td>PROXY_AUTHENTICATION_REQUIRED</td>
<td>407</td>
</tr>
<tr>
<td>408</td>
<td>REQUEST_TIMEOUT</td>
<td>408</td>
</tr>
<tr>
<td>410</td>
<td>GONE</td>
<td>410</td>
</tr>
<tr>
<td>413</td>
<td>REQUEST_ENTITY_TOO_LARGE</td>
<td>413</td>
</tr>
<tr>
<td>414</td>
<td>REQUEST_URI_TOO_LONG</td>
<td>414</td>
</tr>
<tr>
<td>415</td>
<td>UNSUPPORTED_MEDIA_TYPE</td>
<td>415</td>
</tr>
<tr>
<td>416</td>
<td>UNSUPPORTED_URI_SCHEMA</td>
<td>416</td>
</tr>
<tr>
<td>420</td>
<td>BAD_EXTENSION</td>
<td>420</td>
</tr>
<tr>
<td>421</td>
<td>EXTENSION_REQUIRED</td>
<td>421</td>
</tr>
<tr>
<td>423</td>
<td>INTERVAL_TOO_BRIEF</td>
<td>423</td>
</tr>
<tr>
<td>480</td>
<td>TEMPORARILY_UNAVAILABLE</td>
<td>480</td>
</tr>
<tr>
<td>481</td>
<td>CALL_OR_TRANSACTION DOES NOT_EXIST</td>
<td>481</td>
</tr>
<tr>
<td>482</td>
<td>LOOP_DETECTED</td>
<td>482</td>
</tr>
<tr>
<td>483</td>
<td>TOO_MANY_HOPS</td>
<td>483</td>
</tr>
<tr>
<td>484</td>
<td>ADDRESS_INCOMPLETE</td>
<td>484</td>
</tr>
<tr>
<td>485</td>
<td>AMBIGUOUS</td>
<td>485</td>
</tr>
<tr>
<td>486</td>
<td>BUSY_HERE</td>
<td>486</td>
</tr>
<tr>
<td>487</td>
<td>REQUEST_TERMINATED</td>
<td>487</td>
</tr>
<tr>
<td>488</td>
<td>NOT_ACCEPTABLE_HERE</td>
<td>488</td>
</tr>
<tr>
<td>489</td>
<td>BAD_EVENT</td>
<td>489</td>
</tr>
<tr>
<td>Response Code</td>
<td>Description</td>
<td>Code</td>
</tr>
<tr>
<td>---------------</td>
<td>----------------------------</td>
<td>------</td>
</tr>
<tr>
<td>491</td>
<td>REQUEST_PENDING</td>
<td></td>
</tr>
<tr>
<td>493</td>
<td>UNDECIPHERABLE</td>
<td></td>
</tr>
<tr>
<td>500</td>
<td>SERVER_INTERNAL_ERROR</td>
<td></td>
</tr>
<tr>
<td>501</td>
<td>NOT_IMPLEMENTED</td>
<td></td>
</tr>
<tr>
<td>502</td>
<td>BAD_GATEWAY</td>
<td></td>
</tr>
<tr>
<td>503</td>
<td>SERVICE_UNAVAILABLE</td>
<td></td>
</tr>
<tr>
<td>504</td>
<td>SERVER_TIMEOUT</td>
<td></td>
</tr>
<tr>
<td>505</td>
<td>VERSION_NOT_SUPPORTED</td>
<td></td>
</tr>
<tr>
<td>513</td>
<td>MESSAGE_TOO_LARGE</td>
<td></td>
</tr>
<tr>
<td>600</td>
<td>BUSY_EVERYWHERE</td>
<td></td>
</tr>
<tr>
<td>603</td>
<td>DECLINE</td>
<td></td>
</tr>
<tr>
<td>604</td>
<td>DOES_NOT_EXIST_ANYWHERE</td>
<td></td>
</tr>
<tr>
<td>606</td>
<td>SESSION_NOT_ACCEPTABLE</td>
<td></td>
</tr>
</tbody>
</table>

**Message Parts**

Following are the parts of a SIP message:

- Start line
- Headers
- Message

Table 1-3 discusses parts of a SIP message.
Table 1-3 Message Parts

<table>
<thead>
<tr>
<th>SIP Message Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| Start Line       | Denotes the beginning of a SIP message. A start line can be either a request or a response. A request start line contains a method name, the SIP URI to which the message is sent, and the SIP version number. The response start line contains the SIP version number, the status code, and the description for the response phrase. Following are sample SIP start lines:  
  - **Request start line**  
    INVITE sip:tpu@hp.com SIP/2.0  
  - **Response start line**  
    SIP/2.0 200 OK |
| Header           | Contains fields that provide additional information about a message, such as To, From, Subject, and Via. Certain header fields, such as To, From, and Subject, contain only a single header field value and appear only once in the header. Certain header fields, such as Via, Contact, and Route, can appear multiple times in a header and can contain multiple values separated by a comma. Following is a sample SIP header:  
  From: <sip:hosta@xyz.com>  
  To: TPU <sip:team@xyz.com>  
  Subject: Confcall  
  Call-ID: 132059753@local.xyz.com  
  Content-Type: application/sdp  
  CSeq: 1 INVITE |
| Message Body     | Contains information about the message that must be passed to the receiver. The message can be either a plain text message or a multimedia message. The session description protocol (SDP) describes the session parameters for a multimedia message. The message body is independent of the SIP protocol and can contain any information. |

Call Flow Scenarios

This section describes the interaction between SIP entities in various call flow scenarios. This section addresses the following topics:

- “Basic Call Setup and Teardown”
- “Using the Proxy Server” (page 30)
- “Using the Redirect Server” (page 32)

Basic Call Setup and Teardown

This section discusses the various SIP requests and responses exchanged between user agents during a basic call setup and teardown.
Figure 1-2 illustrates the interaction between user agents while a session is being established and terminated.

**Figure 1-2 Establishing and Terminating a Call**

The following steps describe the sequence of messages exchanged between the user agents:

- The following messages are exchanged while establishing a session:
  1. UA1 sends an INVITE request message to invite UA2 to participate in the session.
  2. UA2 responds with a 100 - Trying response message to indicate that the request has been received.
  3. UA2 sends the 180-Ringing response to indicate that UA2 is ringing.
  4. UA2 sends the 200-OK response to inform that the user at UA2 has picked up the call.
  5. UA1 sends the ACK request to confirm receipt of the 200-OK response.
- The following messages are exchanged while terminating a session:
  6. When the user at UA1 ends the call, a BYE request message is sent to UA2.
  7. UA2 responds with the 200-OK message and informs the user at UA2 that the session has ended.

**Using the Proxy Server**

A proxy server exists between the users. This server handles requests and responses between the users. Figure 1-3 illustrates the interaction between users that use the proxy server.
The following messages are exchanged while UA1 uses a proxy server to connect to UA2:

1. UA1 sends an INVITE request to UA2. This request is first sent to the proxy server and then to UA2.
2. The proxy server responds with a 100–Trying message to UA1.
3. The proxy server looks up user2’s current location in a location service.
4. The location service returns user2’s current location.
5. The proxy server decides to proxy the call and creates a new INVITE transaction based on the original INVITE message, but with the Request-URI in the start line changed to user2@alp.sev.com. The proxy server sends this request to UA2.
6. UA2 responds with a 100–Trying message to the proxy server.
7. The UA2 responds with a 180–Ringing response to the proxy server.
8. The proxy server forwards the 180–Ringing response back to UA1.
9. UA2 sends a 200–OK response, when the call is accepted by the user (for example, by picking up the handset).
10. The proxy server forwards the `200-OK` response back to UA1.
11. UA1 sends an `ACK` directly to UA2.

Using the Redirect Server

This section discusses the steps involved in establishing a call that uses a redirect server. Figure 1-4 illustrates the interaction between the user agents, redirect server, and the location service while establishing a call.

Figure 1-4 Call Setup Using a Redirect Server

The following steps describe how a call is established using a redirect server:
1. UA1 sends an `INVITE` message to `host1@seq.com`. UA1 notices the redirect server `red.seq.com` in the signalling path.
2. The redirect server looks up host1’s current location in the location service.
3. The location service provides host1’s current location (`87604309@sev.alp.com`).
4. The redirect server sends host1’s current location to UA1 using a `302` response.
5. UA1 acknowledges the response by sending an `ACK` message to the redirect server.
6. UA1 directly sends an `INVITE` message to the `sev.alp.com` domain.
7. The `sev.alp.com` domain identifies the call to host1. The `host1` answers the call.
8. UA2 sends a `200-OK` response to UA1.
9. UA1 acknowledges the response with an `ACK` message.
Implementations of SIP on HP-UX

The HP implementation of SIP is available on the following stacks:

- C SIP
- Java SIP

The following sections discuss the documentation available for these stacks.
This section addresses the following topics:

- “Documentation for the C SIP Stack”
- “Documentation for the Java SIP Stack” (page 33)

Documentation for the C SIP Stack

HP uses the C SIP stack from RADVISION Ltd. to support SIP on the C stack in the HP-UX operating system. For information on the C SIP stack and the relevant APIs, see the following documents at http://www.docs.hp.com:

- HP-UX C SIP Stack Programmer’s Guide
- HP-UX C SIP Stack Reference Guide
- HP-UX C SIP Stack Message Layer Reference Guide

Documentation for the Java SIP Stack

HP uses the Java Specification Request 32 (JSR32) APIs for Integrated Networks (JAIN) to support the Java stack in the HP-UX operating system. The HP-UX Java SIP Stack Programmer’s Guide describes the JSR32 architecture, and explains how to configure, compile, and troubleshoot applications that are written using the JSR32 JAIN APIs. This document is also available at:

http://www.docs.hp.com
2 JSR32 JAIN SIP APIs

This chapter discusses the JSR32 stack architecture, the main objects in the stack, the packages and factories available in the stack, and the JSR32 features.

This chapter addresses the following topics:

• “Overview”
• “JSR32 Architecture” (page 36)
• “HP-UX JSR32 Features” (page 42)

Overview

The SIP protocol, as defined in RFC 3261 (Session Initiation Protocol), is accepted as the industry standard for establishing sessions between users in a network. In a heterogeneous network, applications on different types of stacks must interact with each other for exchanging or passing information. Though the SIP protocol specification ensures interoperability between different stacks, it does not address interoperability between applications in different types of stacks. JAIN SIP (JSIP) fulfils this requirement by using the Java programming language to define an API specification that ensures interoperability between stacks and also interoperability of applications across stacks.

The Java APIs for Integrated Networks (JAIN) is an activity within the Java Community Process (JCP), that develops APIs for telephony (voice and data) services. JSR32, the JSIP API specification, defines APIs that enable rapid development of IP telephony services and applications using Java. The JSIP APIs provide a standard portable interface to share information between SIP clients and SIP servers, through call control elements and by enabling converged-network applications. These APIs enable application developers to have powerful access to the SIP protocol.

JSIP offers the following benefits:

• Standardizes the interface to the stack.
• Standardizes the message interface.
• Standardizes events and event semantics
• Provides application portability that enables interoperability of applications across stacks.

The JSIP supports the SIP protocol described in RFC 3261. It can be used in a user agent, proxy server, redirect server, or registrar.

JSIP APIs provide the following functions:

• Provide methods to format SIP messages.
• Enable an application to send and receive SIP messages.
• Parse incoming messages.
• Enable applications to access fields through a standardized Java interface.
- Invoke appropriate application handlers for message arrivals and protocol-specific events, such as transaction timeouts.
- Provide transaction support, and manage transaction state and lifetime on behalf of a user application.
- Provide dialog support, and manage dialog state and lifetime on behalf of a user application.
- Handle retransmissions that reduces complexity for applications acting as user agents.

The JSIP enables voice over IP gateways, client end points, PBXs, and other communication systems to interface with each other using the SIP protocol. It addresses call setup, teardown, presence, and instant messaging over IP-based networks. However, JSIP does not include a mechanism for media streaming between the caller and the called party. The JSIP API provides an industry standard interface for proprietary SIP protocol stacks.

The JSR32 SIP stack conforms to the following RFC standards:

- RFC 2806 (URLs for Telephone Calls)
- RFC 2976 (The SIP INFO Method)
- RFC 3258 (Distributing Authoritative Name Servers via Shared Unicast Addresses)
- RFC 3261 (Session Initiation Protocol)
- RFC 3262 (Reliability of Provisional Response in the Session Initiation Protocol (SIP))
- RFC 3265 (Session Initiation Protocol (SIP) - Specific Event Notification)
- RFC 3311 (The Session Initiation Protocol (SIP) UPDATE Method)
- RFC 3326 (The Reason Header Field for the Session)
- RFC 3327 (Session Initiation Protocol (SIP) Extension Header Field for Registering Non-Adjacent Contacts)
- RFC 3428 (Session Initiation Protocol (SIP) Extension for Instant Messaging)
- RFC 3515 (The Session Initiation Protocol (SIP) Refer Method)
- RFC 3581 (An Extension to the Session Initiation Protocol (SIP) for Symmetric Response Routing)
- RFC 3608 (Session Initiation Protocol (SIP) Extension Header Field for Service Route Discovery During Registration)
- RFC 3665 (Session Initiation Protocol (SIP) Basic Call Flow Examples)
- RFC 3903 (Session Initiation Protocol (SIP) Extension for Event State Publication)

**JSR32 Architecture**

This section discusses the JSR32 SIP stack architecture, and the different objects, packages, and factories in the JSR32 SIP stack.
This section addresses the following topics:
- “JSR32 SIP Stack” (page 37)
- “JSR32 Packages” (page 39)
- “JSR32 Objects” (page 39)
- “JSR32 Factories” (page 41)

**JSR32 SIP Stack**

This section discusses the different layers in the JSR32 SIP stack.

Figure 2-1 (page 37) illustrates the different layers and the interaction between them in the JSR32 SIP architecture.

**Figure 2-1 JSR32 SIP Architecture**

![JSR32 SIP Architecture Diagram]

Network (RAW bytes)
The JSR32 SIP architecture includes the following components:

- **SipListener**
  This interface represents the application view to a SIP stack and defines the application's communication channel to the SIP stack. It also defines the methods required by an application to receive and process events that are generated by an object implementing the `SipProvider` interface. An application can accept one of the following event types:

  - RequestEvent
    Specifies request messages emitted as events by the `SipProvider`.
  - ResponseEvent
    Specifies response messages emitted as events by the `SipProvider`.
  - TimeoutEvent
    Specifies timeout notifications emitted as events by the `SipProvider`.
  - IOExceptionEvent
    Specifies input/output (I/O) exception notifications emitted as events by the `SipProvider`.
  - TransactionTerminatedEvent
    Specifies transaction terminated notifications emitted as events by the `SipProvider`.
  - DialogTerminatedEvent
    Specifies dialog terminated notifications emitted as events by the `SipProvider`.

- **SipProvider**
  This interface represents the messaging entity of a SIP stack and defines the messaging and transactional view of the SIP stack.

- **ListeningPoint**
  This interface represents a unique IP network listening point, which consists of port, transport and IP address. A `ListeningPoint` is a Java representation of the socket that a `SipProvider` messaging entity uses to send and receive messages.

- **JSR32 SIP stack**
  The JSR32 SIP stack contains the following components:

  - Dialog
    Represents a peer-to-peer SIP relationship between two user agents. Dialog facilitates sequencing of messages between user agents and routes requests between the user agents.
  - Transaction
    Specifies a request sent by a client transaction to a server transaction, along with all responses to that request sent from the server transaction back to the client transactions.
  - Request
    Specifies a message sent from the client to the server.
  - Response
    Specifies a message sent from the server to the client.
  - Parser
    Specifies an interface that accepts raw bytes from the network, parses it according to the grammar specified in RFC 3261 (Session 38 JSR32 JAIN SIP APIs)
Initiation Protocol, and converts to a format that an application can interpret and process.

Encoder Specifies an interface that encodes the message from the application and sends them over the network as raw bytes.

- Network
  Contains raw bytes that is sent to the JSR32 SIP stack for further processing.

**JSR32 Packages**

HP provides the following packages for the Java SIP stack:

- **General Package** (`javax.sip`)
  This package contains the main interfaces in the JSIP architecture, the transaction, dialog interfaces, and the event objects.

- **Address Package** (`javax.sip.address`)
  This package contains interfaces that represent the addressing components of the SIP protocol.

- **Header Package** (`javax.sip.header`)
  This package contains all the headers interfaces.

- **Message Package** (`javax.sip.message`)
  This package contains the interfaces that represent SIP messages.

**JSR32 Objects**

The JSIP JSR32 specification includes the following main objects:

- The `SipStack` Object
- The `Dialog` Object
- The `Transaction` Object
- The `Message` Object

**SipStack Object**

The `SipStack` object defines the management and architectural view of the SIP stack. It contains methods to represent and modify the properties of a SIP stack.
An application can have multiple SipStack objects. However, only a single SipStack can be associated with an IP address. The SipStack object defines the following methods to control the architecture and setup of the SIP stack:

- Creation or deletion of SipProviders. A SipProvider represents messaging objects that can be used by an application to send request and response messages statelessly or statefully using client and server transactions.
- Creating or deletion of ListeningPoints. A ListeningPoint is an interface that represents a unique IP networking listening point, which consists of a port, transport, and IP address.

SipStack is instantiated by the SipFactory interface, ensuring that the pathname for the implementation of the stack (com.hpl.net for HP JSR32 implementation) is set and initialized with a set of static configuration properties. An application can use these configuration properties to modify the SIP stack properties. For information on the different configuration properties, see “Configuration Parameters” (page 45). An application must set these properties before initializing the SIP stack. If the properties of a SIP stack require modification after the stack is initialized, the SipStack must be deleted and recreated.

**Dialog Object**

The Dialog object facilitates sequencing of messages between user agents and proper routing of requests between them. A dialog is not directly created by an application. It is established by the dialog creating transactions, such as INVITE or SUBSCRIBE, and managed by the SIP stack. A single dialog can contain many transactions.

A dialog is identified by the dialog ID, which contains the following information:

- **Call ID**
  - Represents the value of the Call ID for all messages that belong to a SIP session.

- **Local Tag**
  - Represents the local tag for a dialog. For a UAC, the local tag is set to the tag in the FROM field of the message and for a UAS, the local tag is set to the tag in the TO field of the message. A local tag for a user agent is identical to the remote tag at the peer user agent. The tags are opaque tokens that facilitate the generation of unique dialog IDs.

- **Remote Tag**
  - Represents the remote tag for a dialog. For a UAC, the remote tag is set to the tag in the TO field of the message and for a UAS, the remote tag is set to the tag in the FROM field of the message.

A dialog contains the following data for further message transmission within the dialog:

- **Dialog ID**
  - Identifies the dialog.

- **Local Sequence Number**
  - Orders requests from the user agent to its peer.

- **Remote Sequence Number**
  - Orders requests from its peer to the user agent.

- **Local URI**
  - Specifies the address of the local party.

- **Remote URI**
  - Specifies the address of the remote party.
Remote Target Specifies the address from the contact header field of the request or response, or refresh request or response.
secure Determines whether the dialog is secure.
Route Set Specifies an ordered list of URIs.

Transaction Object
The transaction object represents a generic transaction interface that defines the methods that are common to client and server transactions. Transaction is a fundamental component of SIP. The transaction is a request sent by a client transaction to a server transaction, along with all responses to that request sent from the server transaction back to the client transactions. User agents and stateful proxies contain a transaction layer, whereas stateless proxies do not contain a transaction layer.

Message Object
A message object represents a request from a client to a server, or a response from a server to a client. Both types of messages consist of a method name, address, and protocol version, one or more header fields that describe the routing of the message, and an optional message-body.

The Message object contains the following common elements of request and response:
- Generic accessor functions to headers.
- Convenience accessor to the expiration value of the message.
- Convenience header accessor methods for the body content type, language, disposition, and length.
- Accessor methods to the body content.

JSR32 Factories
A factory interface provides methods that enable an application to create the factory-specific objects. The JSIP JSR32 specification defines the following factories:
- **SipFactory**
  This interface defines methods to create new stack objects and other factory objects. It is a single access point to obtain proprietary implementations of the JSR32 specification.
- **AddressFactory**
  This interface defines methods to create address objects, URIs, SIP URIs (SipURIs), and telephone URLs (TelURLs).
• HeaderFactory
  This interface defines methods to create new headers objects.

• MessageFactory
  This interface defines methods to create new request and response objects.

HP-UX JSR32 Features

This section discusses the features in the HP implementation of the JSR32 APIs.
This section addresses the following topics:

• “Event Notification” (page 42)
• “The REFER Method” (page 43)
• “SIP for Telephones” (page 43)
• “Provisional Response Acknowledgement in SIP” (page 43)
• “E.164 Number Mapping” (page 44)
• “Parsing Different URIs” (page 44)
• “Additional SIP Extensions” (page 44)
• “Persistent Connection” (page 45)
• “Transport Support” (page 45)
• “Support for IPv6 Addresses” (page 45)
• “Configuration Parameters” (page 45)

Event Notification

In a SIP infrastructure, a SIP node requests event notifications from the remote nodes to indicate that certain events have occurred. Hence, it is important for the end nodes to co-operate with each other in providing appropriate event notifications. The SIP services that request event notification include automatic callback services, buddy lists, message waiting indications, presence, and public switched telephone network (PSTN) and internetworking.

The SIP stack supports the following SIP methods to subscribe for events and to notify events:

• SUBSCRIBE
  The SUBSCRIBE method is used to request asynchronous notification of an event or a set of events.

• NOTIFY
  The NOTIFY method is used to notify a SIP node that an event requested by the previous SUBSCRIBE method has occurred. The NOTIFY method also provides further details about the event.

For more information on these methods, see RFC 3265 (SIP Specific Event Notification).
The REFER Method

The SIP stack enables the implementation of transfer services using the REFER method, as defined in RFC 3515 (The Session Initiation Protocol (SIP) Refer Method). The REFER method can be used in call transfer applications. It indicates that the recipient must contact a resource using the contact information provided in the request. The protocol for emitting and responding to a REFER request is identical to that of a BYE request.

A user agent accepting a REFER request must request approval from the user that is sending the request. If the user grants approval, the user agent must contact the resource identified by the URI in the Refer-To header field in the request. The NOTIFY mechanism is used to inform the user that is sending the REFER method about the status of the reference.

SIP for Telephones

SIP for Telephones (SIP-T), as defined in RFC 3372 (Session Initiation Protocol for Telephones (SIP-T): Context and Architectures), is a specification that defines how to interwork SIP with PSTN. The SIP-T specification does not define any new SIP extension, but uses the existing extension (such as PRACK, INFO, and so on) and other advanced SIP features (such as Multipart MIME and 183 response request) to translate ISDN user part (ISUP) and SIP messages, and in some cases, the tunnel ISUP in SIP.

The SIP stack provides all the necessary building blocks required by an application to be compliant with SIP-T, for example, soft switches and PSTN gateways.

Provisional Response Acknowledgement in SIP

The SIP product contains two types of responses, namely provisional and final. Final responses convey the result of the request processing. These responses are sent reliably. Provisional responses provide information about the progress of the request processing. These responses are not sent reliably. Provisional response acknowledgement (PRACK) is an IETF SIP extension that enables reliable transmission of provisional responses, as specified in RFC 3262 (Reliability of Provisional Responses in the Session Initiation Protocol (SIP)). The PRACK message is used in opening one-way media sessions before a call is established and for quality of service (QoS) negotiation before completing an INVITE transaction.

Each provisional response contains a sequence number that is stored in the RSeq header field in the provisional response. The RSeq header is used to transmit the provisional responses reliably. The PRACK messages contain an Rack header field, which indicates the sequence number of the provisional response that is being acknowledged. This header field is sent in a PRACK request to support reliable transmission of provisional responses.
E.164 Number Mapping

In voice over IP (VoIP) networks, an endpoint is identified by an IP address and a telephone number. When a user calls a SIP remote endpoint from a PSTN network, only the telephone number is known. The SIP uniform resource identifiers (URI) is unknown. Therefore, it is important to translate the telephone number into the SIP URI for the call to be established. While the SIP user agents or proxy servers can be statically provisioned by mapping the destinations corresponding to particular telephone numbers or telephone number ranges, considering the size and complexity of a complete mapping, it is preferable if SIP user agents can query a destination that is appropriate for a particular telephone number.

E.164 Number Mapping (ENUM) is a standard that enables the translation of telephone numbers to SIP URIs. The ENUM feature performs the translation using the existing Internet domain name server (DNS) query mechanism. The ENUM mechanism is used in cellular IP Multimedia Subsystem (IMS) networks, and also for fixed and mobile operators.

Parsing Different URIs

A SIP address can be described in different URI formats. A tel URI describes a SIP end point that is using a phone number. A pres URI is a SIP address used for presence user agents (UAs). An im URI is used for instant messaging. Each URI can appear in many variations in a SIP message. Therefore, parsing these URIs is important. The SIP stack parses the different URIs and provides the data to the application in a friendly data structure.

Additional SIP Extensions

The SIP stack supports a variety of features to implement a number of extensions to SIP. The following extensions are supported:

- General uniform resource locator (URL) scheme support – This extension is a general framework to support any type of URL scheme, such as im: and tel:.
- INFO method – This extension is used for mid-call information exchange without changing the call state. For more information about this extension, see RFC 2976 (The SIP INFO Method).
- MESSAGE method – This extension is used for instant messaging. For more information about this extension, see draft-ietf-sip-message-0x (SIP Extensions for Instant Messaging).
- UPDATE method – This extension is used to enable a client to update the parameters of a session (such as the set of media streams and their codecs) without changing the dialog state. For more information about this extension, see RFC 3311 (The Session Initiation Protocol (SIP) UPDATE Method).
Persistent Connection

The SIP stack uses a single TCP or TLS connection for different messages, transactions, or dialogs. However, the frequent opening and closing of TCP connections is not desirable, because of the extra messaging overhead in TCP and TLS connections. Hence, SIP supports connection persistency, which is the reuse of an open connection. The SIP stack enables connection reuse by specifying that a message can be sent on an existing open connection.

Transport Support

HP-UX JSR32 supports SIP over different transport protocols, such as UDP, TCP, and TLS. You can configure the JSR32 SIP stack to listen on several UDP, TCP, and TLS local addresses.

Support for IPv6 Addresses

HP-UX JSR32 supports IP version 6 (IPv6) addresses. You can configure the JSR32 SIP stack to listen on IPv6 and IPv4 addresses simultaneously, and also to receive and send SIP messages using IPv6 packets.

Configuration Parameters

Table 2-1 lists the SipStack properties that enable you to configure the JSR32 SIP stack.

**Table 2-1 SIP Stack Parameters**

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>javax.sip.STACK_NAME</td>
<td>Sets a user friendly name to identify the underlying stack implementation. The value of the stack name must not contain blank spaces. This property is mandatory.</td>
</tr>
<tr>
<td>javax.sip.OUTBOUND_PROXY</td>
<td>Sets the outbound proxy of the SIP Stack. The format of the outbound proxy parameter must be ipaddress:port/transport, for example, 129.1.22.333:5060/UDP. This property is optional.</td>
</tr>
</tbody>
</table>
| javax.sip.ROUTER_PATH        | Sets the fully qualified class path to the application-supplied router object that determines how to route messages, when the stack cannot make a routing decision pertaining to non-SIP URIs. This property is optional.  
<pre><code>                    | An application-defined router object must implement the javax.sip.Router interface.                                                            |
</code></pre>
<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>javax.sip.EXTENSION_METHODS</code></td>
<td>Informs the underlying implementation about supported extension methods that create new dialogs. This configuration flag must be used for dialog creating extension methods; other extension methods that do not create dialogs can be used using the method parameter on request, assuming that the implementation understands the method. If more than one method is supported in this property, each extension must be separated with a colon, for example FOO:BAR. This property is optional.</td>
</tr>
<tr>
<td><code>javax.sip.RETRANSMISSION_FILTER</code></td>
<td>Prevents an application from handling retransmission ACK requests, 1xx, and 2xx responses for INVITE transactions. Instead, SipProvider handles the retransmissions. The RETRANSMISSION_FILTER option is useful for hiding protocol retransmission semantics from higher level programming environments. This property is optional and the default value is OFF.</td>
</tr>
<tr>
<td><code>javax.sip.AUTOMATIC_DIALOG_SUPPORT</code></td>
<td>Provides the flexibility to toggle the dialog support. The option to turn off the dialog support is motivated by dialog free server (such as proxy servers) that do not want to pay for the overhead of the dialog layer and user agents that require to create multiple dialogs for a single INVITE message. The default value is ON.</td>
</tr>
<tr>
<td><code>javax.sip.FORKABLE_EVENTS</code></td>
<td>Specifies a comma-separated list of events that results in forked SUBSCRIBE dialogs. Each element in this list must be of the syntax packagename.eventname. This property is optional.</td>
</tr>
<tr>
<td><code>javax.sip.USE_ROUTER_FOR_ALL_URIS</code></td>
<td>Specifies the router that the stack must consult for routing decisions. If this property is set to true, the application installed router is consulted for all routing decisions (that is, both dialog SIP and non-SIP request URIs). If this property is set to false, the user installed router is consulted for routing of non-SIP URIs. This property is optional. The default value for this property is true.</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>-------------</td>
</tr>
<tr>
<td>com.hp.net.javax.sip.SERVER_LOG</td>
<td>Logs valid incoming messages. If SERVER_LOG is not specified (that is, it contains a null value) and the value of TRACE_LEVEL is more than 16, the error message are printed to standard output.</td>
</tr>
<tr>
<td>com.hp.net.javax.sip.DEBUG_LOG</td>
<td>Specifies the location for storing the debug log messages.</td>
</tr>
<tr>
<td>com.hp.net.javax.sip.TRACE_LEVEL</td>
<td>Sets tracing and debugging using the TRACE and DEBUG strings. If you set TRACE_LEVEL to a value greater or equal to 16, the incoming valid messages are logged in the SERVER_LOG. If you set TRACE_LEVEL to 32 and specify a DEBUG_LOG, that trace information is dumped in the specified DEBUG_LOG file.</td>
</tr>
<tr>
<td>com.hp.net.javax.sip.LOG_MESSAGE_CONTENT</td>
<td>Specifies a boolean value that captures the message content into a log file. HP does not recommend you to log content if you are using SIP to push a lot of information through TCP. The default value is false.</td>
</tr>
<tr>
<td>com.hp.net.javax.sip.MAX_MESSAGE_SIZE</td>
<td>Specifies the maximum size of data that a TCP connection can read. The minimum value is 4 KB. MAX_MESSAGE_SIZE prevents Denial of Service (DoS) attacks that are launched by writing to a TCP connection until the server chokes. The default value is infinity, that is, no limit.</td>
</tr>
<tr>
<td>com.hp.net.javax.sip.CACHE_SERVER_CONNECTIONS</td>
<td>Specifies a boolean value that closes the server socket after a server transaction enters the TERMINATED state, if CACHE_SERVER_CONNECTIONS is set to false. This enables a server to protect against TCP-based DoS attacks launched by clients that initiate multiple transactions. If CACHE_SERVER_CONNECTIONS is set to true, the stack retains the socket in open state to maximize the performance of the stack at the expense of the thread and memory resources, and exposes itself to DoS attacks. The default value is true.</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>com.hp.net.java.sip.CACHE_CLIENT_CONNECTIONS</td>
<td>Specifies whether the listener is re-entrant. If the listener is re-entrant, the stack manages a thread pool and simultaneously calls the listener from the same thread that reads the message. Multiple transactions can concurrently receive messages. As a result, multiple threads are active in the listener at the same time. The default value is false.</td>
</tr>
<tr>
<td>com.hp.net.java.sip.THREAD_POOL_SIZE</td>
<td>Specifies concurrency control from the number of simultaneous active threads. If THREAD_POOL_SIZE is not specified and the listener is re-entrant, each event delivered to the listener is run in the context of a new thread. If THREAD_POOL_SIZE is specified and the listener is re-entrant, the stack runs the listener using a thread from the thread pool. This enables you to manage the level of concurrency to a fixed maximum. Threads are pre-allocated when the stack is instantiated. If THREAD_POOL_SIZE is specified and the listener is not re-entrant, the stack uses the thread pool thread from this pool, to parse and manage the state machine but runs the listener in its own thread. The default value is infinity.</td>
</tr>
<tr>
<td>com.hp.net.java.sip.MAX_CONNECTIONS</td>
<td>Specifies the maximum number of simultaneous TCP connections handled by the stack.</td>
</tr>
<tr>
<td>com.hp.net.java.sip.MAX_SERVER_TRANSACTIONS</td>
<td>Specifies the maximum size of the server transaction table. The low water mark is 80% of the high water mark. Requests are selectively dropped from the low water mark to the high water mark range. If the server transaction table is smaller than the low water mark, requests are unconditionally accepted by the stack. The default value for the high water mark is 5000.</td>
</tr>
<tr>
<td>com.hp.net.java.sip.CACHE_CLIENT_CONNECTIONS</td>
<td>Specifies a boolean value that closes the server socket after a client transaction enters the TERMINATED state, if CACHE_CLIENT_CONNECTIONS is set to false. This enables a client to release buffer threads and socket connections that are associated with a client transaction after the transaction terminates, at the expense of the performance of the stack. The default value is true.</td>
</tr>
</tbody>
</table>

Table 2-1 SIP Stack Parameters (continued)
<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.hp.net.jasper.sip.PASS_INIVTE_NON2XX_ACK_TO_LISTENER</td>
<td>If this parameter is set to true, the listener views the ACK for the non-2xx responses. The default value is false.</td>
</tr>
<tr>
<td>com.hp.net.jasper.sip.MAX_LISTENER_RESPONSE_TIME</td>
<td>Specifies the maximum time (in seconds) before sending a response to a server transaction. The stack deletes this transaction if a server does not send a response within this period of time. The default value is infinity, that is, if the listener does not respond, the stack hangs on to a reference for the transaction. It results in a memory leak.</td>
</tr>
<tr>
<td>com.hp.net.jasper.sip.USE_TLS_ACCELERATOR</td>
<td>Permits the delegation of TLS encryption or decryption to an external non-SIP-aware TLS accelerator hardware. The default value is false.</td>
</tr>
<tr>
<td>com.hp.net.jasper.sip.DELIVER_TERMINATED_EVENT_FOR_ACK</td>
<td>Notifies the termination of the ACK transactions (pseudo-transaction) to the application so that all the server transactions are handled uniformly in the user code during cleanup. The default value is false.</td>
</tr>
<tr>
<td>com.hp.net.jasper.sip.READ_TIMEOUT</td>
<td>Defines the timeout (in milliseconds) between successive reads after the first byte of a SIP message is read by the stack. This is relevant for incoming TCP connections to prevent starvation at the server. All the SIP headers must be delivered within this interval and each successive buffer must be of the content delivered in this interval. The default value is -1.</td>
</tr>
<tr>
<td>com.hp.net.jasper.sip.ADDRESS_RESOLVER</td>
<td>Specifies the fully qualified classpath for an implementation of the AddressResolver interface. The AddressResolver enables you to support lookup schemes for addresses that are not directly resolvable to IP addresses using getHostByName(). You can customize address lookup by specifying your own address resolver. The default address resolver is a pass-through address resolver, that is, it returns the input string without performing a resolution.</td>
</tr>
<tr>
<td>com.hp.net.jasper.sip.AUTO_GENERATE_TIMESTAMP</td>
<td>Automatically generates a getTimeOfDay timestamp for a retransmitted request if the original request contained a timestamp.</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>-------------</td>
</tr>
<tr>
<td>com.hp.net.javasip.LOG_FACTORY</td>
<td>Specifies the fully qualified classpath for an implementation of the MessageLogFactory. The SIP stack calls the MessageLogFactory functions to log the messages that are received or sent. The MessageLogFactory enables you to log auxiliary information related to the application or environmental conditions into the log stream.</td>
</tr>
<tr>
<td>com.hp.net.javasip.RFC_2543_SUPPORT_ENABLED</td>
<td>Specifies whether RFC 2543 (SIP: Session Initiation Protocol) is fully supported in the stack. If RFC 2543 is fully supported, check whether the To tag is disabled. As a result, forked dialogs do not work properly. If dialog forking is not important for an application, you can set this flag to true.</td>
</tr>
<tr>
<td>com.hp.net.javasip.THREAD_AUDIT_INTERVAL_IN_MILLISECS</td>
<td>Specifies the frequency (in milliseconds) with which an application can audit the SIP stack about the health of its internal threads. The audit of the SIP stack enables an application to detect failures, such as internal thread terminating caused by an exception or a deadlock condition. Such events result in the stack becoming inoperable and, therefore, requires immediate action from the application layer. The immediate action can be an alarm, trap, reboot, or failover. Thread audits are disabled by default.</td>
</tr>
</tbody>
</table>
This chapter describes how to create an application. Creating an application involves the following tasks:

1. Creating an instance of the SIP stack
2. Creating instances of factory implementations
3. Creating an instance of the SipProvider
4. Forming a SIP message
5. Sending a SIP message
6. Processing an event

The subsequent sections in this chapter describe these tasks in detail.

This chapter addresses the following topics:

- “Creating an Instance of the SIP Stack” (page 51)
- “Creating Instances of Factory Implementations” (page 52)
- “Creating an Instance of the SipProvider” (page 52)
- “Forming a SIP Message” (page 53)
- “Sending a SIP Message” (page 53)
- “Processing an Event” (page 54)

Creating an Instance of the SIP Stack

This section describes how to create an instance of the SIP stack.

NOTE: You must create an instance of the SIP stack only after configuring the SIP stack properties.

Follow this procedure to create an instance of the SIP stack:

1. Create an instance of the SipFactory using the following method:

   ```java
   SipFactory sipFactory=SipFactory.getInstance();
   ```

2. Initialize the path name using the following method:

   ```java
   sipFactory.setPathName("com.hp.net");
   ```

3. Configure the stack parameters using the following method:

   ```java
   Properties prop = new Properties();
   prop.setProperty("prop_name", "val");
   ```
   
   where:
   
   - `prop_name` specifies the name of the stack parameter.
   - `val` specifies the value for the property.
Example 3-1 Setting the TRACE_LEVEL Stack Parameter

To set the TRACE_LEVEL stack parameter, set the property as follows:
prop.setProperty("com.hp.net.javax.sip.TRACE_LEVEL", "32");

4. Configure the SIP stack using the following method:
sipStack sipStack=sipFactory.createSipStack(prop);

Creating Instances of Factory Implementations

You must create instances of the factory implementations before you can create a SIP message.

To create an instance of the address, header, and message factory implementations, use the following methods:
- HeaderFactory headerFactory = sipFactory.createHeaderFactory();
- AddressFactory addressFactory = sipFactory.createAddressFactory();
- MessageFactory messageFactory = sipFactory.createMessageFactory();

Creating an Instance of the SipProvider

Before creating an instance of a SipProvider, you must create a listening point that represents a particular port, address, and transport. You can create a SipProvider for a particular listening point.

Follow this procedure to create an instance of the SipProvider:

1. To create a listening point, use the following method:
sipStack.createListeningPoint("ipaddress", port, "transport");

Example 3-2 Sample Method for Creating a Listening Point

Following is a sample method for creating a listening point:
ListeningPoint udpListeningPoint
    =sipStack.createListeningPoint("127.0.0.1", 5060, "udp");

2. To create a SipProvider on a particular listening point, use the following method:
sipStack.createSipProvider(ListeningPoint);

Example 3-3 Creating a SipProvider

Following is a sample method to create a SipProvider:
SipProvider provider=sipStack.createSipProvider(udpListeningPoint);
Forming a SIP Message

You can use factories to create a SIP message. Therefore, you must create the factories before forming a SIP message.

Follow this procedure to create factories:

1. To create an address, use the AddressFactory interface, as follows:
   
   ```
   Address address=addressFactory.createAddress(URI);
   Address address=addressFactory.createAddress(String);
   Address address=addressFactory.createAddress(String, URI);
   ```

2. To create a header, use the HeaderFactory interface, as follows:
   
   ```
   AcceptHeader accept=headerFactory.createAcceptHeader
   (String, String);
   ```

3. To create a request message or response message using a message factory, use the MessageFactory interfaces, as follows:
   
   ```
   Request request=messageFactory.createRequest(<parameters>);
   Response response=messageFactory.createResponse(<parameters>);
   ```

Sending a SIP Message

You can send a SIP message in the following scenarios:

- Stateless
- Transaction-Stateful
- Dialog-Stateful

The following methods enable you to send messages in different scenarios:

- To send a message in a stateless scenario, send the request or response to the transport using the obtained SipProvider object, specified as follows:
  
  ```
  — provider.sendResponse(Response);
  — provider.sendRequest(Request);
  ```

- Follow this procedure to send a message in a transaction-stateful scenario:
  
  1. To create a new client or server transaction from the SipProvider, use the following methods:
     
     ```
     — ClientTransaction ctr=provider.getNewClientTransaction();
     — ServerTransaction str=provider.getNewServerTransaction();
     ```
To use an existing client or server transaction formed by the arrival of request or response events, use the following methods:

- ServerTransaction str=RequestEvent.getServerTransaction();
- ClientTransaction ctr=ResponseEvent.getClientTransaction();

2. Send the request or response using the obtained transaction object, specified as follows:
   
   ctr.sendRequest();
   str.sendResponse(Response);

- Follow this procedure to send a message in a dialog-stateful scenario:
  
  1. To create a new dialog from the SipProvider, use the following method:
     
     Dialog dialog=provider.getNewDialog();
  
  To use an existing dialog formed by the arrival of events, use the following method:

     Dialog dialog=ResponseEvent.getDialog();
     Dialog dialog=RequestEvent.getDialog();

  2. To send the request using the obtained dialog object, specify the following methods:

     dialog.sendRequest(ClientTransaction);

Processing an Event

The receipt of a request message or response message is notified to an application as events. Processing an event involves receiving and parsing the SIP message. This section describes how to receive and parse SIP messages.

Follow this procedure to process an event:

1. To receive a message, use the getRequest() or onResponse() API using an instance of the RequestEvent or ResponseEvent, specified as follows:

   Request req=RequestEvent.getRequest();
   Response resp=ResponseEvent.getResponse();

2. To parse a message, use the appropriate get* APIs, such as getHeader() or getContentLength() of the Message interface.

3. To set values, use the appropriate set* APIs, such as setHeader() or setContentLength() of the Message interface.
This chapter describes how to compile and run applications that use the JSR32 APIs. Follow this procedure to compile and run applications that use JSR32 APIs:

1. Set the PATH variable to the specific Java Development Kit (JDK) version, as specified in the following example:
   ```bash
   export PATH=$PATH:/opt/java1.5/bin
   ```

2. Set the class path to the location of the following jar files:
   - JainSipApi1.2.jar
   - JainSipRi1.2.jar
   - concurrent.jar
   - log4j-1.2.8.jar

3. To compile an application, enter the following command:
   ```bash
   javac -classpath <jarfile1;jarfile2;.....> <app_name>
   ```
   where:
   - jarfile Specifies the relative path to the jar files.
   - app_name Specifies the application.

   **Example 4-1 Compiling an Application**
   
   To compile an application named `appl.java`, enter the following command if the jar files are located in the current working directory:
   ```bash
   javac -classpath JainSipApi1.2.jar:JainSipRi1.2.jar:
   concurrent.jar:log4j-1.2.8.jar appl.java
   ```

4. To run an application, enter the following command:
   ```bash
   java -classpath <jarfile1;jarfile2;.....> app_name <argument>
   ```
   where:
   - jarfile Specifies the relative path to the jar files.
   - app_name Specifies the application.
   - argument Specifies arguments, if any.

   **Example 4-2 Running an Application**
   
   Following is a sample command to run an application named `appl.java`:
   ```bash
   java -classpath JainSipApi1.2.jar:JainSipRi1.2.jar:
   concurrent.jar:log4j-1.2.8.jar appl
   ```
This chapter discusses the exceptions raised by the JSR32 APIs. Table 5-1 lists the exceptions raised by the JSR32 APIs and their causes.

### Table 5-1  JSR32 API Exceptions

<table>
<thead>
<tr>
<th>Exception Name</th>
<th>Cause</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>The javax.sip Exceptions</strong></td>
<td></td>
</tr>
<tr>
<td>InvalidArgument Exception</td>
<td>An application passes an invalid argument, such as an invalid numerical value.</td>
</tr>
<tr>
<td>ObjectInUseException</td>
<td>A method is unable to delete a specified object, because the object is still in use by the underlying implementation.</td>
</tr>
<tr>
<td>PeerUnavailableException</td>
<td>The implementation of the JAIN SIP interface could not be created.</td>
</tr>
<tr>
<td>SipException</td>
<td>A general SIP exception is encountered. The specialized exceptions defined in the JSR32 specification cannot handle an error.</td>
</tr>
<tr>
<td>TransactionAlreadyExistsException</td>
<td>An application attempts to get a transaction to handle a message when the transaction is already handling the message.</td>
</tr>
<tr>
<td>TransactionDoesNotExistException</td>
<td>An application attempts to reference a client or server transaction that does not exist in the SipProvider.</td>
</tr>
<tr>
<td>TransactionUnavailableException</td>
<td>The implementation cannot create a transaction</td>
</tr>
<tr>
<td>TransportNotSupportedException</td>
<td>A specific transport is not supported by an implementation of this specification.</td>
</tr>
<tr>
<td>TransportAlreadySupportedException</td>
<td>A specific transport is already supported by a SipProvider through its listening points.</td>
</tr>
<tr>
<td><strong>The javax.sip.header Exception</strong></td>
<td></td>
</tr>
<tr>
<td>TooManyHopsException</td>
<td>An application attempts to decrement the hop count when the message has already reached its maximum number of forwards.</td>
</tr>
</tbody>
</table>
## Glossary

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dialogue</td>
<td>A dialog is a peer to peer association between communicating SIP end points.</td>
</tr>
<tr>
<td>Java Community Process</td>
<td>Java Community Process (JCP) is the home of the international developer community whose charter is to develop and evolve Java technology specifications, reference implementations, and technology compatibility kits.</td>
</tr>
<tr>
<td>Java Specification Request</td>
<td>A Java specification request (JSR) is a document submitted to the Program Management Office (PMO) by one or more members to propose the development of a new Specification or significant revision to an existing specification.</td>
</tr>
<tr>
<td>JCP</td>
<td>See Java Community Process.</td>
</tr>
<tr>
<td>JSR</td>
<td>See Java Specification Request.</td>
</tr>
<tr>
<td>National Institute of Standards and Technology</td>
<td>The National Institute of Standards and Technology (NIST), Advanced Networking Technologies Division provides a public domain implementation of the JAVA Standard for SIP (JAIN SIP), which serves as a reference implementation for the standard. The JSIP stack can work in proxy server or user agent scenarios and has been used in numerous commercial and research projects. It supports RFC 3261 and a number of extension RFCs including RFC 3265 (Subscribe/Notify) and RFC 3262 (Provisional Reliable Responses).</td>
</tr>
<tr>
<td>NIST</td>
<td>See National Institute of Standards and Technology.</td>
</tr>
<tr>
<td>Route Set</td>
<td>The route set is the list of servers that need to be traversed to send a request to the peer.</td>
</tr>
<tr>
<td>Session</td>
<td>Internet telephone calls, multimedia distribution, and multimedia conferences, with one or more participants</td>
</tr>
<tr>
<td>Stateful Proxy</td>
<td>A SIP transaction consists of a single request and any number of responses to that request.</td>
</tr>
<tr>
<td>TLS accelerator</td>
<td>TLS accelerator is a separate hardware that takes care of TLS functionality.</td>
</tr>
<tr>
<td>Transaction</td>
<td>Transactions are a fundamental component of SIP. A transaction is a request sent by a client transaction (using the transport layer) to a server transaction, along with all responses to that request sent from the server transaction back to the client.</td>
</tr>
</tbody>
</table>
Index

A
ACK method, 24
AddressFactory, 41
application layer, 21

B
BYE method, 24

C
C SIP stack, 33
call ID, 40
CANCEL method, 24
cancel error response code, 27
command sequence, 23
compiling an application, 55
  example, 55
configure
  stack parameter, 51
configuring
  SIP stack, 52
creating
  factory, 53
  header, 53
listening point, 52
request message, 53
response message, 53
SipProvider, 52
CSeq (see command sequence)

D
dialog, 38
data types, 40
dialog ID, 40
dialog-stateful, 54
DialogTerminatedEvent, 38
DNS (see domain name server)
domain name server, 44

E
E.164 number mapping, 44
encoder, 39
ENUM (see E.164 number mapping)
exceptions, 57

G
global error response code, 28

H
header, 24
  sample, 29
  header field, 29
HeaderFactory, 42
HTTP (see HyperText Transfer Protocol)
HyperText Transfer Protocol, 17

I
IETF (see Internet Engineering Task Force)
IMA (see IP multimedia subsystem)
INFO method, 24, 44
Internet Engineering Task Force, 17
INVITE method, 24
IOExceptionEvent, 38
IP multimedia subsystem, 44
ISDN user part, 43
ISUP (see ISDN user part)

J
JAIN (see Java APIs for Integrated Networks)
JAIN SIP, 35
  benefit, 35
  function, 35
Java APIs for Integrated Networks, 35
Java Community Process, 35
Java Development Kit, 55
Java Specification Request 32, 33
JCP
  Java Community Process, 35
JDK (see Java Development Kit)
JSIP (see JAIN SIP)
JSR32 (see Java Specification Request 32)
JSR32 SIP architecture
  component, 38
JSR32 SIP stack
  components, 38

L
listening point
  creating, 52
  example, 52
ListeningPoint, 38
local sequence number, 40
local tag, 40
local URL, 40
location service, 19

M
Media Gateway Control Protocol, 17
MEGACO (see Media Gateway Control Protocol)
message body, 24, 29
MESSAGE method, 24, 44
MessageFactory, 42

N
network layer, 22
network transport layer, 22
NOTIFY method, 24

O
OPTIONS method, 24

P
parser, 38
parsing
   SIP message, 54
PRACK method, 24
PRACK response, 43
presence service, 19
provisional response code, 26
proxy server
   presence service, 19
   stateful, 19
   stateless, 19
PSTN (see public switched telephone network)
public switched telephone network, 42, 43

Q
QoS (see quality of service)
quality of service, 43

R
RAck field, 43
Real-Time Streaming Protocol, 17
Real-Time Transport Protocol, 17
reason phrase, 24
redirection response code, 26
REFER method, 24
REGISTER method, 24
remote sequence number, 40
remote tag, 40
remote target, 41
remote URI, 40
request, 38
request message
   call ID field, 23
   command sequence field, 23
   contact field, 23
   content-length field, 23
   content-type field, 23
   first line field, 23
   form of, 22
   from field, 23
   sample, 22
   subject field, 23
   to field, 23
   via field, 23
RequestEvent, 38
response, 38
response code
   client error, 27
   global error, 28
   provisional, 26
   redirection, 26
   server error, 28
   success, 26
response message
   form of, 22
ResponseEvent, 38
RFC 3265, 19
route set, 41
RSeq header, 43
RTP (see Real-Time Transport Protocol)
RTSP, 17
   (see also Real-Time Streaming Protocol)
running an application, 55
   example, 55

S
SCTP (see stream control transmission protocol)
SDP (see Session Description Protocol)
secure SIPS, 23
server error response code, 28
Session Description Protocol, 17
SIP
   building block, 20
   infrastructure, 17
   protocol layer, 21
   transaction layer, 21
   transaction user, 21
SIP for telephones, 43
SIP message
   dialog-stateful, 54
   stateless, 53
   transaction-stateful, 53
SIP request method, 23
   ACK, 24
   BYE, 24
   CANCEL, 24
   INFO, 24
   INVITE, 24
   MESSAGE method, 24
   NOTIFY, 24
   OPTIONS, 24
   PRACK, 24
   REFER, 24
   REGISTER, 24
   SUBSCRIBE, 24
   UPDATE, 24
SIP response code, 25
SIP response message
   headers, 24
   message body, 24
   reason phrase, 24
sample, 24
status code, 24
SIP-T (see SIP for telephones)
SipFactory, 41, 51
SipListener, 38
SipProvider, 38, 52
creating, 52
example, 52
SIPS (see secure SIPS)
SipStack object, 40
start line, 29
sample line, 29
stateful proxy server, 19
stateless proxy server, 19
stateless scenario, 53
status code, 24
status line
detail, 25
stream control transmission protocol, 21
SUBSCRIBE method, 24
success response code, 26

T
TCP (see transmission control layer)
TimeoutEvent, 38
TLS (see transport layer security)
transaction, 38
transaction layer, 21
transaction user, 21
transaction-stateful, 53
TransactionTerminatedEvent, 38
transaction control layer, 21
transport layer security, 21
TU (see transaction user)

U
UA (see user agent)
UAC (see user agent client)
UAC request, 19
UAS (see user agent server)
UDP (see user datagram protocol)
uniform resource identifier, 23, 44
uniform resource locator, 44
UPDATE method, 24, 44
URI (see uniform resource identifier)
URI format
im, 44
pres, 44
tel, 44
URL (see uniform resource locator)
URL scheme, 44
user agent, 18
user agent client, 19
user agent server, 19
user datagram protocol, 21

V
voice over IP, 44
VoIP (see voice over IP)