
Juphoon Protocol Framework

Juphoon SIP Stack

Updated: 2007-6-13

Please visit <http://www.juphoon.com> for more information

Juphoon SIP Introduction

Ningbo Juphoon System Software Co., Ltd.

<http://www.juphoon.com>

Tel: +86-574-87287820

Fax: +86-574-87304379

Text Number: 100-000-02-02

Copyright © 2007, Juphoon System Software Corporation.

All rights reserved.

Contents

1. INTRODUCTION	5
1.1 PURPOSE	5
1.2 SCOPE	5
1.3 TERMINOLOGY	5
1.4 REFERENCES	5
1.5 OVERVIEW	6
2. PROTOCOL FEATURES	8
2.1 RFCs	8
2.2 METHODS	9
2.3 CODEC	10
2.4 TRANSPORTS	12
2.5 TRANSACTION	13
2.6 DIALOG PROCESS	13
2.7 URI SCHEMES	13
2.8 MIME TYPES	13
2.9 AUTOMATIC GENERATION OF HEADERS	14
2.10 DNS QUERY	14
2.11 OFFER-ANSWER	14
2.12 RELIABLE PROVISIONAL RESPONSE	14
2.13 SESSION TIMER	15
2.14 SIGCOMP	15
2.15 IPV6 SUPPORT	15
2.16 SSL/TLS SUPPORT	15
2.17 SIP BODIES	15
2.18 SDP INTEGRATION	15
2.19 SIP FRAGMENT	16
2.20 EVENT PACKAGE	16
2.21 IM & PRESENCE	17
2.22 3GPP & IMS	17
2.23 QUALITY OF SERVICE	17
2.24 SECURITY	17
2.25 NAT TRAVERSAL	17
3. IMPLEMENTATIONS OF SIP	18
3.1 SIP PHONE (JPHONE)	18
3.2 SIP IM & PRESENCE	18
3.3 SIP SERVER	19
3.4 IMS PoC CLIENT	19
4. SIP USER INTERFACES	21
4.1 INTERFACE PRIMITIVES	21

4.2	INTERFACE PRIMITIVES	22
4.3	SESSION EVENTS	23
4.4	MESSAGE INTERFACES	24
5.	FUNCTION EXTENSIONS	24
6.	ABOUT THE IMPLEMENTATION OF SIP.....	25
6.1	RELATED MODULES.....	25
6.2	ABOUT THE TEST OF SIP	25
6.3	SIZES OF THE CODES.....	25
7.	APPLICATION DEVELOPMENT	26
7.1	SEND AN INVITE REQUEST	26
7.2	SEND A RESPONSE TO AN INVITE REQUEST	28
7.3	SEND AN ACK REQUEST	29
7.4	CONVERSION OF A SESSION EVENT.....	30
7.5	USAGE OF SIP MESSAGE INTERFACES	32
8.	ABOUT THE DEMO PROGRAM.....	33
8.1	ABOUT THE STARTUP.....	33
8.2	MEMORY MESSAGES.....	34
8.3	FILE MESSAGES.....	35
8.4	PERFORMANCE ANALYSIS.....	36
9.	OTHER PRODUCTS.....	40
9.1	OPERATING SYSTEM SERVICE PLATFORM	40
9.2	PROTOCOL SOFTWARE	40
9.3	COMPILERS	40
9.4	TESTING TOOLS	40

Tables

Table 2-1 RFCs.....	9
Table 2-2 Header fields supported by SIP	12
Table 4-1 Interface primitives.....	23
Table 6-1 Sizes of the codes	25

Figures

Figure 2-1 DNS Query and Cache	14
Figure 3-1 SIP Phone (JPhone) implementations	18
Figure 3-2 A reference of Juphoon SIP IM interface.....	19
Figure 3-3 SIP Server reference scenario	19
Figure 3-4 Example PoC UE endpoint	20
Figure 4-1 Primitive operation flow between layers	21
Figure 4-2 Using SIP interface primitives to send requests and responses.....	22
Figure 8-1 Comparison of codec performance between Juphoon SIP and osip	39

1. Introduction

This document covers the main services provided by Juphoon SIP as well as its usage.

1.1 Purpose

In this document, SIP functions as protocol conformance, interface primitives and part of the usage of SIP are presented. And the introduction on the demo is the concluding section of this document.

1.2 Scope

This document provides several functions of SIP and their usages but not the description about the design and the implementation of SIP. For more details please refer to Juphoon SIP Function Definition and SIP Release Notes.

1.3 Terminology

ZOS Zero Operating System

SIP Session Initiation Protocol

SDP Session Description Protocol

1.4 References

- [1] *RFC2327: Session Description Protocol*
- [2] *RFC4566: Session Description Protocol*
- [3] *RFC3261: Session Initiation Protocol*
- [4] *RFC3262: Reliability of Provisional Response*
- [5] *RFC3263: SIP: Locating SIP Servers*
- [6] *RFC3264: An Offer-Answer Model with Session Description*
- [7] *RFC3265: SIP – Specific Event Notification*
- [8] *RFC3311: The SIP UPDATE method*
- [9] *RFC3312: Integration of Resource Management and SIP*
- [10] *RFC3313 Private SIP Extensions for Media Authorization*
- [11] *RFC3323: A Private mechanism for the Session Initiation Protocol*
- [12] *RFC3325: Private Extensions to the SIP for Asserted Identity within Trusted Networks*
- [13] *RFC3326: The Reason Header*
- [14] *RFC3329: Security Mechanism Agreement for the SIP*
- [15] *RFC3372: SIP for Telephones (SIP-T) Context and Architectures*
- [16] *RFC3428: SIP Extensions for Instant Messaging*
- [17] *RFC3515: The REFER method*
- [18] *RFC2806: URLs for Telephone Calls*
- [19] *RFC2976: The SIP INFO Method*
- [20] *RFC3204: MIME media types for ISUP and QSIG Objects*
- [21] *RFC3581: An Extension to the SIP for Symmetric Response Routing*
- [22] *RFC3903: Session Initiation Protocol (SIP) Extension for Event State Publication*
- [23] *RFC3959: The Early Session Disposition Type for the SIP*

- [24] *RFC3960: Early Media and Ringing Tone Generation in the SIP*
- [25] *RFC3911: The Session Initiation Protocol (SIP) "Join" Header*
- [26] *RFC3455: Private Header (P-Header) Extensions to the Session Initiation Protocol (SIP) for the 3GPP*
- [27] *RFC3327: SIP Extension Header Field for Registering Non-Adjacent Contacts*
- [28] *RFC3608: SIP Extension Header Field for Service Route Discovery During Registration*
- [29] *RFC2368: The mailto URL scheme*
- [30] *RFC4028: The SIP Session Timer*
- [31] *draft-ietf-sip-events-01: SIP-Specific Event Notification*
- [32] *RFC3515: The Refer Method*
- [33] *RFC3891: The Session Initiation Protocol (SIP) "Replaces" Header*
- [34] *draft-ietf-sip-callerprefs: Caller Preferences and Callee Capabilities for the Session Initiation Protocol (SIP)*
- [35] *RFC3323: SIP Extensions for Network-Asserted Caller Identity and Privacy within Trusted Networks*
- [36] *RFC3824: Using E.164 numbers with the Session Initiation Protocol (SIP)*
- [37] *RFC4354: A Session Initiation Protocol (SIP) Event Package and Data Format for Various Settings in Support for the Push-to-Talk over Cellular (PoC) Service*
- [38] *RFC4575:: A Session Initiation Protocol (SIP) Event Package for Conference State*
- [39] *RFC3603: Private Session Initiation Protocol (SIP) Proxy-to-Proxy Extensions for Supporting the Packet Cable Distributed Call Signaling Architecture*
- [40] *RFC3856: A Presence Event Package for the SIP*
- [41] *RFC3857: A Watcher Information Event Template-Package fo SIP*
- [42] *RFC4662: A SIP Event Notification Extension*

1.5 Overview

Juphoon provides professional protocol software featuring:

- **Uniform and clear code architecture**
- **Standard interfaces**
- **High portability**
- **High performance**
- **Good extensibility**

After reading the codes and running the protocol software, users will have an idea about the high quality of the SIP protocol. Because Juphoon has been focusing on the quality of products, all the protocol products are conform to the protocol specifications.

Juphoon Software Architecture and Standard (JSAS) is architecture for software development. It includes a protocol architecture and protocol specifications. Through JSAS our protocol products can be independent of processors, compilers, and operating systems. They also have good compatibility and can easily be integrated with the customers' products.

All protocol products are programmed using JSAS and the same programming standards. This can be demonstrated by comparing the SDP ABNF and SIP ABNF in the demo. So users only need to

learn one of the protocol designs and the programming standards to easily maintain other Juphoon protocol products. This will save the cost of maintenance and development by a large margin.

2. Protocol features

SIP stack is conform to protocol standards, provides fast encoding and decoding which is also easy to extend and maintain, supports basic transports, and manages transactions and Dialog. At present, Juphoon SIP stack have almost every function that IMS asks.

2.1 RFCs

SIP stack supports the following RFCs:

RFC	Description
RFC 3261	Session Initiation Protocol
RFC 3262	Reliability of Provisional Response
RFC 3263	Locating SIP Servers
RFC 3264	An Offer-Answer Model with Session Description
RFC 3265	Specific Event Notification
RFC 3311	The SIP UPDATE method
RFC 3312	Integration of Resource Management and SIP
RFC 3313	Private SIP Extensions for Media Authorization
RFC 3323	A Private mechanism for the SIP
RFC 3325	Private Extensions to the SIP for Asserted Identity within Trusted Networks
RFC 3326	The Reason Header
RFC 3329	Security Mechanism Agreement for the SIP
RFC 3372	SIP for Telephones (SIP-T): Context and Architectures
RFC 3428	SIP Extensions for Instant Messaging
RFC 3515	The REFER method
RFC 2806	URLs for Telephone Calls
RFC 2976	The SIP INFO Method
RFC 3204	MIME media types for ISUP and QSIG Objects
RFC 3581	An Extension to the SIP for Symmetric Response Routing
RFC 3903	Session Initiation Protocol (SIP) Extension for Event State Publication
RFC 3911	The Session Initiation Protocol (SIP) "Join" Header
RFC 3959	The Early Session Disposition Type for the SIP
RFC 3960	Early Media and Ringing Tone Generation in the SIP

RFC 3455	Private Header (P-Header) Extensions to the Session Initiation Protocol (SIP) for the 3GPP
RFC 3327	SIP Extension Header Field for Registering Non-Adjacent Contacts
RFC 3608	SIP Extension Header Field for Service Route Discovery During Registration
RFC 2368	The mailto URL scheme
RFC 2327	Session Description Protocol
RFC 4028	The SIP Session Timer
draft-ietf-sip-events-01	SIP-Specific Event Notification
RFC 3515	The Refer Method
RFC 3891	The Session Initiation Protocol (SIP) "Replaces" Header
draft-ietf-sip-callerprefs	Caller Preferences and Callee Capabilities for the Session Initiation Protocol (SIP)
RFC 3323	SIP Extensions for Network-Asserted Caller Identity and Privacy within Trusted Networks
RFC 2392	Content-ID and Message-ID Uniform Resource Locators

Table 2-1 RFCs

2.2 Methods

SIP stack supports the following Methods:

- INVITE
- ACK
- OPTIONS
- CANCEL
- BYE
- REGISTER
- PRACK
- SUBSCRIBE
- NOTIFY
- INFO
- UPDATE
- MESSAGE
- REFER
- COMET
- PUBLISH

SIP stack also supports other augmented Methods

2.3 Codec

SIP stack supports precise codec of the head fields in Table 2-2. It can also use the table of head fields to encode or decode part of the head fields.

RFC	Supported SIP header fields
RFC 3261	<ol style="list-style-type: none"> 1. Accept 2. Accept-Encoding 3. Accept-Language 4. Alert-Info 5. Allow 6. Authentication-Info 7. Authorization 8. Call-ID 9. Call-Info 10. Contact 11. Content-Disposition 12. Content-Encoding 13. Content-Language 14. Content-Length 15. Content-Type 16. CSeq 17. Date 18. Error-Info 19. Expires 20. From 21. In-Reply-To 22. Max-Forwards 23. MIME-Version 24. Min-Expires 25. Organization 26. Priority 27. Proxy-Authenticate 28. Proxy-Authorization 29. Proxy-Require 30. Record-Route 31. Reply-To 32. Require 33. Retry-After

	<ul style="list-style-type: none"> 34. Route 35. Server 36. Subject 37. Supported 38. Timestamp 39. To 40. Unsupported 41. User-Agent 42. Via 43. Warning 44. WWW-Authenticate 45. extension-header
RFC 3262	<ul style="list-style-type: none"> 46. RAck 47. RSeq
RFC 3265	<ul style="list-style-type: none"> 48. Event 49. Allow-Events 50. Subscription-State
RFC 3515	<ul style="list-style-type: none"> 51. Refer-To
draft-ietf-sip-events	<ul style="list-style-type: none"> 52. Subscription-Expires
RFC 3892	<ul style="list-style-type: none"> 53. Referred-By
RFC 3891	<ul style="list-style-type: none"> 54. Replaces
RFC 4028	<ul style="list-style-type: none"> 55. Session-Expires 56. Min-SE
RFC 3841	<ul style="list-style-type: none"> 57. Request-Disposition 58. Accept-Contact 59. Reject-Contact
draft-ietf-sip-privacy	<ul style="list-style-type: none"> 60. Anonymity 61. RPID-Privacy 62. Remote-Party-ID
RFC 3903	<ul style="list-style-type: none"> 63. SIP-ETag 64. SIP-If-Match
RFC 3911	<ul style="list-style-type: none"> 65. Join
RFC 3455 for 3GPP	<ul style="list-style-type: none"> 66. P-Associated-URI 67. P-Called-Party-ID 68. P-Visited-Network-ID

	69. P-Access-Network-Info 70. P-Charging-Addr 71. P-Charging-Vector
RFC 3327 for 3GPP	72. Path
RFC 3608 for 3GPP	73. Service-Route
RFC 3325	74. PAssertedID 75. PPreferredID
RFC 3313	76. P-Media-Authorization
RFC 3323	77. Privacy-hdr
RFC 3326	78. Reason
RFC 3329	79. security-client 80. security-server 81. security-verify
RFC 3603	82. P-DCS-Trace-Party-ID 83. P-DCS-OSPS 84. P-DCS-Billing-Info 85. P-DCS-LAES 86. P-DCS-Redirect
RFC 4244	87. History-Info
draft poc-p-headers	88. P-Alerting-Mode 89. P-Answer-State
RFC 4457	90. P-User-Database
RFC 4488	91. Refer-Sub
RFC 2392	92. Message-ID 93. Content-ID

Table 2-2 Header fields supported by SIP

SIP stack also supports other augmented headers.

2.4 Transports

Transports including UDP and TCP are supported now. If users are using transports other than UDP and TCP, we can make our SIP stack support these transports as soon as possible.

SIP stack supports the transports which are described in RFC3261:

If a request is within 200 bytes of the path MTU, or if it is larger than 1300 bytes and the path MTU is unknown, the request must be sent using an RFC2914 congestion controlled transport protocol, such as TCP.

2.5 Transaction

There are four transaction FSMs supported by SIP stack:

- INVITE client transaction
- Non-INVITE client transaction
- INVITE server transaction
- Non-INVITE server transaction

Both transactions of User Agent and Proxy can be processed by Juphoon SIP stack, although they have different requirements in details.

The transaction process will not be totally limited in a dialog because transaction and dialog are loosely coupled.

2.6 Dialog process

Ways of dialog establishment specified by RFC 3261(INVITE), RFC 3265(SUBSCRIBE, NOTIFY), RFC 3428 (REFER) are supported.

Early dialog, establish dialog and multi dialog are supported.

2.7 URI Schemes

SIP stack supports the following SIP URI schemes:

- SIP
- SIPS
- IM
- TEL
- Absolute

Other augmented URI schemes are also supported.

2.8 MIME Types

SIP stack supports the following Content Types:

- multipart/mixed
- application/sdp
- application/isup
- application/qsig
- text/plain
- message/sipfrag
- application/ simple-message-summary
- message/ cpim
- application/reginfo+xml
- application/watcherinfo+xml
- application/pdf+xml
- application/rlmi+xml
- application/simple-filter+xml

Other augmented MIME types are also supported.

2.9 Automatic generation of headers

SIP stack can automatically generate headers during a calling process. This will be helpful for the developers to simplify the development and implement services more quickly.

SIP stack can automatically generate headers including Call-ID, Cseq, Contact, Via, From, To, Content-Length. And it is necessary for the users to write From and To when a Dialog is initiated by a request like INVITE, SUBSCRIBE or REFER. Then SIP stack will write these headers for users in subsequent operations. But the stack will not do so if the users have designated these headers.

2.10 DNS query

The SIP stack supports the DNS query specified by RFC 3261 and RFC 3263.

It also supports several track records as A, SRV, NAPTR and etc. Communications between the stack and the DNS client terminal use loose coupling mechanism (system messages). And the DNS client should provide cached information.

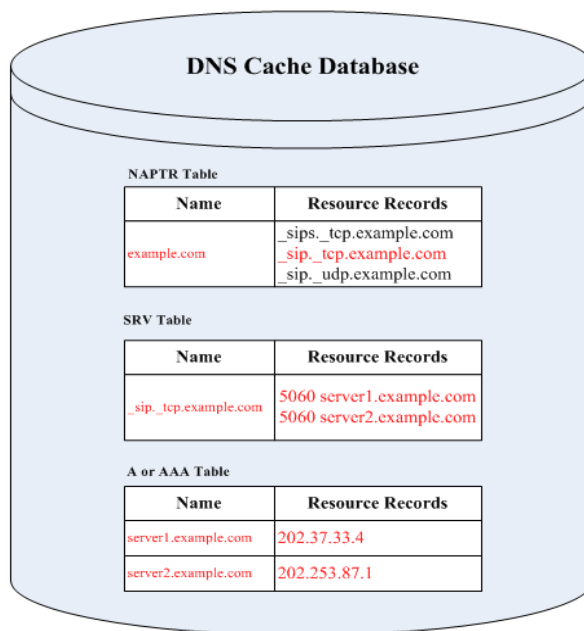


Figure 2-1 DNS Query and Cache

2.11 Offer-Answer

SIP stack supports carrying SDP messages and Offer Answer model specified by RFC3264.

By supporting Alert-Info head and Offer Answer model, SIP stack can support Early Media specified by RFC3960.

2.12 Reliable Provisional Response

SIP stack uses PRACK specified by RFC3262 to send reliable provisional responses.

SIP stack supports RSeq header field and Rack header field and the option tag of 100rel (see enum EN_SIP_OPT_TAG_100REL).

2.13 Session Timer

SIP stack can ask for periodical update of a SIP session by using sending a Re-INVITE request or UPDATE request.

SIP stack supports Session-Expire header field and Min-SE header field specified by RFC4028. It also supports the option tag of timer (see enum EN_SIP_OPT_TAG_TMR).

2.14 SigComp

Juphoon provides an independent SigComp module which can be integrated with the transport layer of SIP stack to perform SIP signaling compression.

SIP stack supports using a URI parameter : comp=sigcomp to indicate that messages are SigComp compressed.

Juphoon SigComp module conforms to the following RFCs:

- RFC 3320 Signaling Compression
- RFC 3321 SigComp - Extended Operations
- RFC 3485 SIP and SDP static Dictionary for SigComp
- RFC 3486 Compressing the SIP
- RFC 4077 A Negative Acknowledgement Mechanism for Signaling Compression
- RFC 4464 SigComp Users Guide
- RFC 4465 SigComp Torture Tests

2.15 IPv6 support

SIP stack supports IPv6 transmission. And this function has been tested in practical environments. SIP stack can be configured by users to support IPv4 transmission or IPv6 transmission.

2.16 SSL/TLS support

SIP stack supports SSL/TLS transport. And this function has been tested in practical environments. SIP stack use OpenSSL library as the default SSL/TLS transport layer.

2.17 SIP Bodies

SIP stack allows a SIP message to carry different SIP bodies.

- MIME messages
- SDP messages
- Mutlitpart messages
- SIP Fragment messages
- XML messages

2.18 SDP integration

SDP stack is an independent module. Considering that many SIP services are well connected with SDP, SIP stack are implicitly integrated with SDP message codec.

SDP stack supports precise decoding of SDP messages. It supports the following RFCs:

- RFC2327 SDP
- RFC4566 SDP
- RFC3266 Support for IPv6 in SDP
- RFC3605 RTCP attribute in SDP.txt
- RFC3556 SDP Bandwidth Modifiers for RTCP Bandwidth
- RFC3890 A Transport Independent Bandwidth Modifier for SDP
- RFC4145 TCP-Based Media Transport in the SDP
- RFC3312 Integration of Resource Management and SDP
- RFC3388 Grouping of Media Lines in SDP
- RFC2733 RTP Payload Format for Generic Forward Error Correction
- RFC2848 Extensions to SIP and SDP for PINT
- RFC2833 RTP Payload for DTMF Digits
- RFC3640 RTP Payload Format for Transport of MPEG-4 Elementary Streams
- RFC3950 Tag Image File Format Fax eXtended (TIFF-FX) - image/tiff-fx MIME Sub-type Registration
- OMA-TS-PoC-ControlPlane-V1_0_1-20061128-A

2.19 SIP Fragment

SIP Fragment supports RFC 3420, i.e. a SIP Body includes a well-formed SIP message. Usually it is used in method REFER to transmit state information about a related request.

SIP Fragment requires Content-Type to support message/sipfrag and a SIP message within a SIP Body like this:

```
sipfrag = [ start-line ]
          *message-header
          [ CRLF [ message-body ] ]
```

2.20 Event Package

SIP stack supports the following Event Packages:

- “presence” Event Package in RFC3856
- “refer” Event Package in RFC3515
- “poc-settings” Event Package in RFC4354
- “conference” Event Package in RFC4575
- “dialog” Event Package in RFC4235
- “kpml” Event Package in RFC4730
- “message-summary” Event Package in RFC3842
- “reg” Event Package in RFC3680
- “ua-profile” Event Package in draft-ietf-sip-xcap-config

SIP stack supports the following Event Template:

- “winfo” Event Template in RFC3857

2.21 IM & Presence

SIP stack supports IM & presence services specified by the following RFCs:

- RFC 3428 Extension for Instant Messaging
- RFC 3856 SIP Extensions for Presence
- RFC 3857 A Watcher Information Event Template-Package (WINFO)
- RFC 4662 A SIP Event Notification Extension

2.22 3GPP & IMS

SIP supports the following augmented 3GPP protocols:

- RFC3455 Private Header (P-Header) Extensions to the Session Initiation Protocol (SIP) for the 3GPP
- RFC3324 Short Term Requirements for Network Asserted Identity
- RFC3608 SIP Extension Header Field for Service Route Discovery During Registration
- RFC3325 Private Extensions to the SIP for Asserted Identity within Trusted Networks
- RFC3603 Private Session Initiation Protocol (SIP) Proxy-to-Proxy Extensions for Supporting the PacketCable Distributed Call Signaling Architecture
- RFC4354 A SIP Event Package and Data Format for Various Settings in Support for the Push-to-Talk over Cellular (PoC) Service
- RFC4083 3GPP Release 5 Requirements
- RFC4032 Update to SIP Preconditions Framework
- Parameters including “+g.poc.talkburst“, “+g.poc.groupad“, “3GPP-GERAN“ and etc.

2.23 Quality of Service

SIP supports the following QoS protocol:

- RFC 3313 Private Extensions for Media Authorization

2.24 Security

SIP supports the following protocols of security mechanisms:

- RFC3323 A Privacy mechanism for the Session Initiation Protocol
- RFC3329 Security Mechanism Agreement

2.25 NAT Traversal

SIP supports the parameters of NAT traversal:

Parameter “received” in Via header field specified by RFC 3261

Parameter “rport” specified by RFC 3581

3. Implementations of SIP

3.1 SIP Phone (JPhone)

JPhone was developed by Juphoon as a solution for SIP Phone. It employs several techniques created by Juphoon, including Fast ABNF, Object Fsm, RTimer and etc. JPhone can realize a wide range of traditional services independent of Call Server and many original Value-Added services. JPhone works with 3G, NGN, P2P, INTERNET and etc.

The JPhone solution allows development and tests on Windows.



Figure 3-1 SIP Phone (JPhone) implementations

3.2 SIP IM & Presence

Juphoon SIP Stack conforms to the following RFCs to support implementation of instant messages based on SIP and endpoints that implement services:

RFC3428, Session Initiation Protocol (SIP) Extension for Instant Messaging

RFC3856, A Presence Event Package for the Session Initiation Protocol (SIP)

This solution also allows development and tests on Windows that brings users a wide range of Look & Feel.

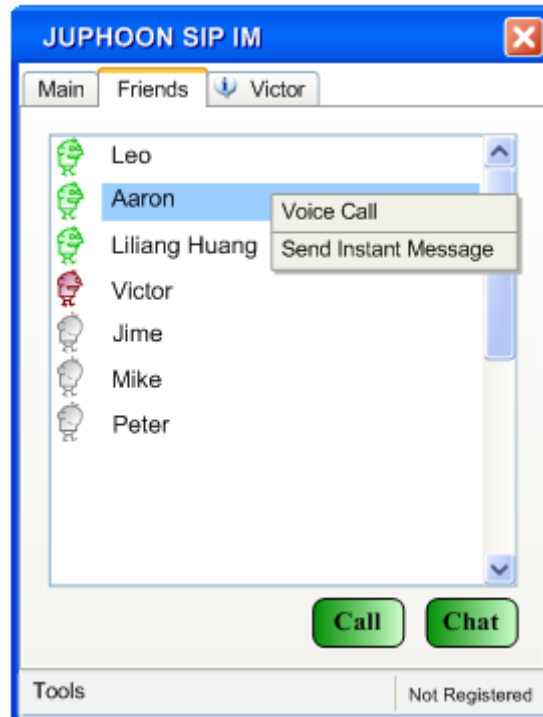


Figure 3-2 A reference of Juphoon SIP IM interface

3.3 SIP Server

Juphoon SIP Stack realizes various features of SIP in the same way of implementing a SIP Server. These features include supporting multi-SUA endpoint and multi-task processing. It also supports servers including SIP Call Server, SIP Proxy (state & stateless) and etc.

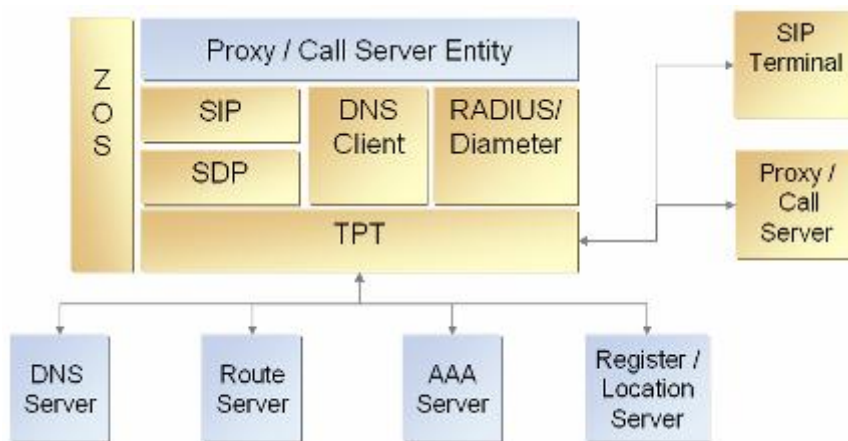


Figure 3-3 SIP Server reference scenario

3.4 IMS PoC client

IMS PoC client was developed with Juphoon SIP Stack and Juphoon PoC Client Framework. It conforms to IETF/OMA/3GPP:

3GPP/3GPP2/IETF & OMA PoC 1.0 Compliant

IETF SIMPLE Compliant

IETF SIP (RFC3261, RFC3262, RFC3265...etc)

IETF SDP (RFC2327, RFC4566...etc)

IETF HTTP (RFC2616, OMA-TS-XDM_Core-V1_0_1, 3GPP TS 24.109 etc)

IETF XCAP (draft-ietf-simple-xcap-12, draft-ietf-simple-xcap-list-usage ... etc)

IETF XML (W3C XML 1.0, 1.1, DOM Level-1, 2, 3... etc)

IETF SigComp (RFC3320, RFC3321, RFC3485, RFC4464 ... etc)

IETF RTP/RTCP (RFC1889, RFC1890 RFC3267, RFC3711... etc)

It has the following functions:

- One-to-one talk
- ad-hoc, pre-arranged and chat
- User list/group management
- User registration
- PoC Session management
- Talk Burst
- Presence
- Instant Personal Alert
- Application Setting and Account Setting management

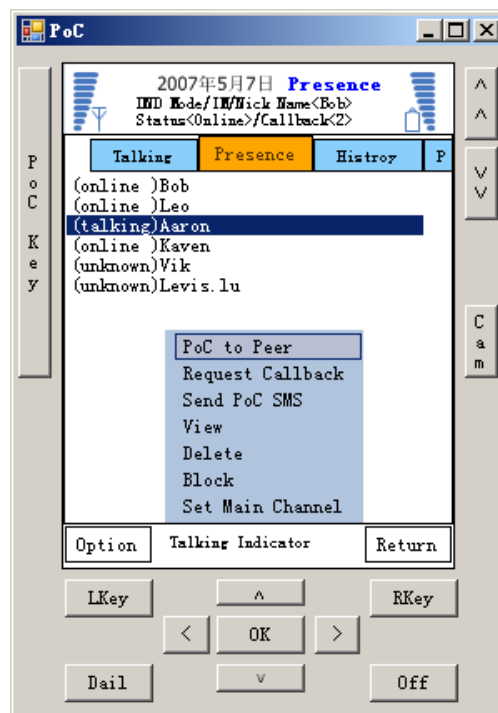


Figure 3-4 Example PoC UE endpoint

4. SIP User Interfaces

The SIP stack provides user primitive interfaces to help users in application development.

There are four kinds of primitives:

- Request
- Confirm
- Indication
- Response

The interface primitives can be classified by the functions of SIP messages:

- Session associated messages, such as INVITE, re-INVITE, BYE, ACK and etc, used for session establishment, session modification and session deletion.
- Session status messages, such as re-INVITE, UPDATE, and etc, used in sessions but unrelated with session establishment, session modification and session deletion.
- Dialog associated messages, such as SUBSCRIBE, REFER, NOTIFY, involved in establishment and deletion of a dialog.
- Call independent messages, such as REGISTER, INFO, OPTION MESSAGE and etc. not belonging to any dialog.

4.1 Interface Primitives

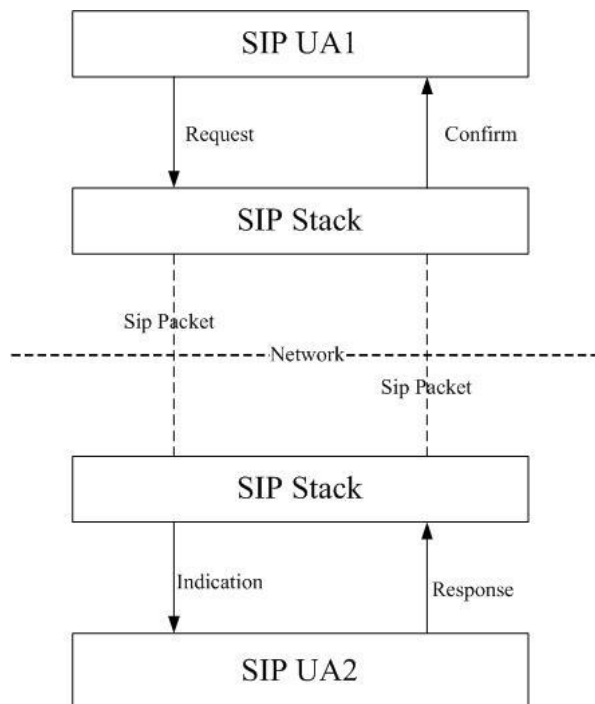


Figure 4-1 Primitive operation flow between layers

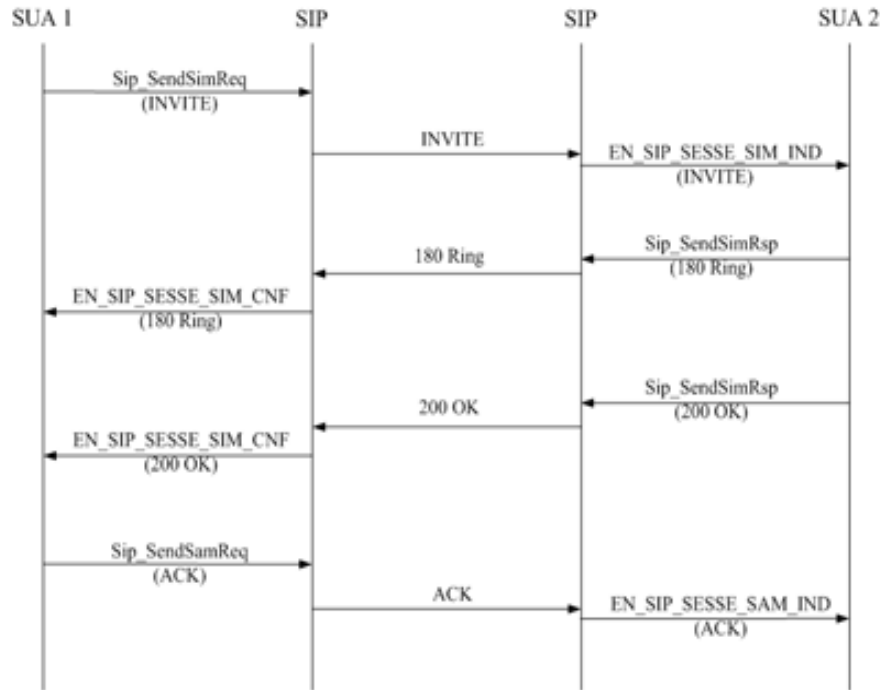


Figure 4-2 Using SIP interface primitives to send requests and responses

4.2 Interface Primitives

Interface primitives for users are listed as the followings:

Interface	Description	Operation flow
Sip_SendSimReq	Sends an INVITE request.	SUA → SIP
Sip_SendSimRsp	Sends a response to an INVITE request.	SUA → SIP
Sip_SendSsmReq	Sends a session status request. The request may be PRACK, INFO, UPDATE, OPTIONS, REGISTER, MESSAGE, COMET, or PUBLISH	SUA → SIP
Sip_SendSsmRsp	Sends a response to a session status request. The request may be PRACK, INFO, UPDATE, OPTIONS, REGISTER, MESSAGE, COMET, or PUBLISH.	SUA → SIP
Sip_SendSamReq	Sends an ACK request.	SUA → SIP
Sip_SendScmReq	Sends a CANCEL request.	SUA → SIP
Sip_SendSmmReq	Sends a re-INVITE request.	SUA → SIP
Sip_SendSmmRsp	Sends a response to a re-INVITE request.	SUA → SIP
Sip_SendStmReq	Sends a BYE request.	SUA → SIP
Sip_SendDamReq	Sends a dialog-related request. The request maybe SUBSCRIBE, REFER, or NOTIFY.	SUA → SIP

Sip_SendDamRsp	Sends a response to a dialog-related request. The request may be SUBSCRIBE, REFER, or NOTIFY.	SUA → SIP
Sip_SendCimReq	Sends a call independent request. The request may be OPTIONS, REGISTER, MESSAGE, COMET, or PUBLISH.	SUA → SIP
Sip_SendCimRsp	Sends a response to a call independent request. The request may be OPTIONS, REGISTER, MESSAGE, COMET, or PUBLISH.	SUA → SIP

Table 4-1 Interface primitives

4.3 Session events

Event	Description	Operation flow
EN_SIP_SESSE_ERR_IND	A session error has occurred.	SIP → SUA
EN_SIP_SESSE_SIM_CNF	A response to an INVITE request has been sent.	SIP → SUA
EN_SIP_SESSE_SIM_IND	An INVITE has been received.	SIP → SUA
EN_SIP_SESSE_SSM_CNF	A response to a PRACK, INFO, UPDATE, OPTIONS, REGISTER, MESSAGE, COMET, or PUBLISH has been sent.	SIP → SUA
EN_SIP_SESSE_SSM_IND	A PRACK, INFO, UPDATE, OPTIONS, REGISTER, MESSAGE, COMET, or PUBLISH request has been received.	SIP → SUA
EN_SIP_SESSE_SAM_IND	An ACK request has been received.	SIP → SUA
EN_SIP_SESSE_SCM_IND	A CANCEL request has been received	SIP → SUA
EN_SIP_SESSE_SMM_CNF	A response to a Re-INVITE request has been sent.	SIP → SUA
EN_SIP_SESSE_SMM_IND	A Re-INVITE has been received	SIP → SUA
EN_SIP_SESSE_STM_IND	A BYE request has been received.	SIP → SUA
EN_SIP_SESSE_DAM_CNF	A response to a SUBSCRIBE, REFER, or NOTIFY request has been sent.	SIP → SUA
EN_SIP_SESSE_DAM_IND	A SUBSCRIBE, REFER, or NOTIFY message has been received.	SIP → SUA
EN_SIP_SESSE_CIM_CNF	A response to a OPTIONS, REGISTER, MESSAGE, COMET, or PUBLISH request has been sent.	SIP → SUA

EN_SIP_SESSE_CIM_IND	An OPTIONS, REGISTER, MESSAGE, COMET, or PUBLISH request has been received.	SIP → SUA
----------------------	---	-----------

4.4 Message interfaces

SIP stack provides almost 200 message interfaces, such as Sip_CreateMsgHdr, Sip_DeleteMsgHdr, Sip_FindMsgHdr and etc., for session establishment, deletion, search, copy, comparison and header field duplication.

5. Function Extensions

The SIP stack codec module supports easy extension. There are three kinds of codec extensions:

- Header extension
- Type extension
- Parameter extension

Parsing of SIP message headers totally or partially is optional. In some scenarios, for example SIP proxy, this is very important because only few headers, such as FROM, TO, VIA and RECORD-ROUTE, should be parsed to help processing more efficiently.

ZOS operating system platform allows Juphoon's software products to be compatible with multiple operating systems. ZOS stands for Zero Operating System, consisting of a set of public service interfaces such as task management, memory management, message queue, data buffer and timer management. SIP stack uses interfaces provided by ZOS to communicate with other modules. ZOS platform supports operation systems including Windows, Linux, VxWorks, Solaris. For example if you want ZOS to supports Window you only need to set the macro to **ZPLATFORM_WIN32** like **ZPLATFORM=ZPLATFORM_WIN32** in the demo.

6. About the implementation of SIP

The main codes of SIP consist of ZOS platform, SDP stack and SIP stack.

6.1 Related modules

There three modules connected with SIP stack:

- ZOS
- ABNF
- SDP

6.2 About the test of SIP

SIPit Interoperability Testing and RFC4475 SIP Torture Testing have been used to test SIP.

SIP stack can couple with servers as Asterisk, SER, OpenSER (open SIP server), CISCO SIP server as well as SIP phones as SJPhone, EyeBeam(X-Lite), LinPhone.

6.3 Sizes of the codes

Sizes of ZOS, SDP, SIP, and RTP stacks vary in different compiling environments:

OS	Compiler	Binary Library
Windows	VC6.0	sip.lib 1810k, zos.lib 561k, sdp.lib 357k, rtp.lib 102k
Linux	GCC 3.2.2	libsip.a 722k libzos.a 264k, libsdp.a 127k, librtp.a 53k

Table 6-1 Sizes of the codes

7. Application development

SIP stack is based on SIP Session Event and primitive interfaces to send and received messages. Here is an example of sending an INVITE request and a response to the request, and an ACK request by using primitives including Sip_SendSimReq, Sip_SendSimRsp, and Sip_SendSamReq.

7.1 Send an INVITE request

```
/* sip ua send INVITE message */
ZINT Sua_SipSendInvite(ST_SUA_MSG_EVT *pstMsgEvt)
{
    ZULONG dwSessUserId, dwDlgUserId, dwTransUserId;
    ST_SIP_TPT_ADDR *pstTptAddr;
    ST_SUA_CP_BODY *pstBody;
    ST_ZOS_DBUF *pstMemBuf;
    ST_SUA_CALL *pstCall;
    ST_SUA_FLOW *pstFlow;
    ST_SIP_MSG *pstMsg;

    /* get call and flow */
    pstCall = pstMsgEvt->pstCall;
    pstFlow = pstMsgEvt->pstFlow;
    pstBody = pstFlow->pstBody;
    pstMemBuf = pstBody ? pstBody->pstMemBuf : ZNULL;

    /* generate invite message */
    if (Sua_GenSipMsg(pstMemBuf, ZTRUE, &pstMsg) != ZOK)
        return ZFAILED;
}
```

```
/* set session event info */
stSessEvt.ucMsgType = EN_SIP_EMT_REQ;
stSessEvt.ucMethodType = EN_SIP_METHOD_INVITE;
stSessEvt.pstMsg = pstMsg;
pstMsg->ucReqPres = ZTRUE;

/* add user uri */
SUA_ADD_USRURI(pstMsg, EN_SIP_METHOD_INVITE, pstFlow);

/* add Supported header */
SUA_ADD_HDR_SUPTED(pstMsg, g_stSuaCfg.dwSuptFlag);

/* add Require header */
SUA_ADD_HDR_REQUIRE(pstMsg, pstFlow->dwRequireFlag);

/* add contact header */
SUA_ADD_HDR_CONTACT(pstMsg, pstFlow);

/* add Allow header */
SUA_ADD_HDR_ALLOW(pstMsg, g_stSuaCfg.dwAllowFlag);

/* add sip body */
SUA_ADD_SIP_BODY(pstMsg, pstBody);

/* set session event */
Sua_SipFlowAddAuthor(pstMsgEvt, EN_SIP_METHOD_INVITE, pstMsg);
Sua_SipFlowGetId(pstMsgEvt, &dwSessUserId, ZNULL, &dwDlgUserId, ZNULL,
                 &dwTransUserId, ZNULL);
Sua_SipFlowGetAddr(pstMsgEvt, ZTRUE, &pstTptAddr);

/* send sim request */
if (Sip_SendSimReq(dwSessUserId, dwDlgUserId, dwTransUserId,
                  pstTptAddr, pstMsg) != ZOK)
{
    Sip_MsgDelete(pstMsg);
    return ZFAILED;
}
```

```
    return ZOK;
}
```

7.2 Send a response to an INVITE request

```
/* sip ua send INVITE response message */
ZINT Sua_SipSendInviteRsp(ST_SUA_MSG_EVNT *pstMsgEvt,
                          ZULONG dwStatCode)
{
    ZULONG dwSessUserId, dwSessId, dwDlgUserId, dwDlgId;
    ZULONG dwTransUserId, dwTransId;
    ST_SUA_CP_EVNT *pstCpEvt;
    ST_SUA_CP_BODY *pstBody;
    ST_ZOS_DBUF *pstMemBuf;
    ST_SUA_FLOW *pstFlow;
    ST_SIP_MSG *pstMsg;

    /* get call flow from message event */
    pstCpEvt = pstMsgEvt->pstCpEvt;
    pstFlow = pstMsgEvt->pstFlow;
    pstBody = pstCpEvt ? pstCpEvt->pstBody : ZNULL;
    pstMemBuf = pstBody ? pstBody->pstMemBuf : ZNULL;

    /* generate invite message */
    if (Sua_GenSipMsg(pstMemBuf, ZFALSE, &pstMsg) != ZOK)
        return ZFAILED;

    /* add Supported header */
    SUA_ADD_HDR_SUPTED(pstMsg, g_stSuaCfg.dwSuptFlag);

    /* add Allow header */
    SUA_ADD_HDR_ALLOW(pstMsg, g_stSuaCfg.dwAllowFlag);

    /* add contact header */
    SUA_ADD_HDR_CONTACT(pstMsg, pstFlow);
}
```

```
/* add sip body */
SUA_ADD_SIP_BODY(pstMsg, pstBody);

/* get the id */
Sua_SipFlowGetId(pstMsgEvt, &dwSessUserId, &dwSessId, &dwDlgUserId,
                 &dwDlgId, &dwTransUserId, &dwTransId);

/* send sim response */
if (Sip_SendSimRsp(dwSessUserId, dwSessId, dwDlgUserId, dwDlgId,
                  dwTransUserId, dwTransId, dwStatCode, pstMsg) != ZOK)
{
    SUA_MSG_DELETE(pstMsg);
    return ZFAILED;
}

return ZOK;
}
```

7.3 Send an ACK request

```
/* sip ua send ACK message */
ZINT Sua_SipSendAck(ST_SUA_MSG_EVT *pstMsgEvt)
{
    ZULONG dwSessUserId, dwSessId, dwDlgUserId, dwDlgId, dwTransUserId;
    ST_SIP_TPT_ADDR *pstTptAddr;

    /* get the id and address */
    Sua_SipFlowGetId(pstMsgEvt, &dwSessUserId, &dwSessId, &dwDlgUserId,
                    &dwDlgId, &dwTransUserId, ZNULL);
    Sua_SipFlowGetAddr(pstMsgEvt, ZTRUE, &pstTptAddr);

    /* send sam request */
    if (Sip_SendSamReq(dwSessUserId, dwSessId, dwDlgUserId, dwDlgId,
                     dwTransUserId, pstTptAddr, pstMsg) != ZOK)
        return ZFAILED;

    return ZOK;
}
```

```
}
```

7.4 Conversion of a session event

When the SIP User Agent(SUA) task receives a message from the SIP stack, it will first convert the SIP session event using Sua_CpEvtTypeInit and then invoke the call FSM.

Here is an example of how function Sua_CpEvtTypeInit in SUA converting a SIP event.

```
/* sip ua cp event type init */
ZCHAR * Sua_CpEvtTypeInit(ST_SUA_MSG_EVNT *pstEvt)
{
    ZULONG dwStatusCode, dwEvtType;
    ST_SIP_SESS_EVNT *pstSessEvt;
    ST_SUA_CP_EVNT *pstCpEvt;
    ST_SUA_TMR *pstTmr;
    ZUCHAR ucMethodType;

    if (pstEvt->ucEvtOwner == EN_SUA_EOT_SM)
    {
        pstSessEvt = pstEvt->pstSessEvt;
        dwEvtType = pstSessEvt->ucEvtType;
        ucMethodType = pstSessEvt->ucMethodType;
        dwStatusCode = pstSessEvt->dwStatusCode;
    }
}
```

```
switch (dwEvtType)
{
case EN_SIP_SESSE_ERR_IND:
    pstEvt->ucEvtType = EN_SUA_CPE_MINOR_SM_RECV_ERR;
    return "ERR IND";
case EN_SIP_SESSE_SIM_IND:
    pstEvt->ucEvtType = EN_SUA_CPE_MINOR_SM_RECV_IVT;
    return "SIM IND";
case EN_SIP_SESSE_SIM_CNF:
    if (dwStatusCode > 100 && dwStatusCode <= 199)
    {
        pstEvt->ucEvtType = EN_SUA_CPE_MINOR_SM_RECV_IVT_1XX;
        return "SIM CNF 1XX";
    }
    else if (dwStatusCode >= 200 && dwStatusCode <= 299)
    {
        pstEvt->ucEvtType = EN_SUA_CPE_MINOR_SM_RECV_IVT_2XX;
        return "SIM CNF 2XX";
    }
    else if (dwStatusCode >= 300 && dwStatusCode <= 699)
    {
        pstEvt->ucEvtType = EN_SUA_CPE_MINOR_SM_RECV_IVT_3XX;
        return "SIM CNF 3456XX";
    }
    break;
case EN_SIP_SESSE_SAM_IND:
    pstEvt->ucEvtType = EN_SUA_CPE_MINOR_SM_RECV_ACK;
    return "SAM IND";
case EN_SIP_SESSE_SCM_IND:
    pstEvt->ucEvtType = EN_SUA_CPE_MINOR_SM_RECV_CANCEL;
    return "SCM IND";
case EN_SIP_SESSE_STM_IND:
    pstEvt->ucEvtType = EN_SUA_CPE_MINOR_SM_RECV_BYE;
    return "STM IND";
default:
    return ZNULL;
}
```

```
}  
  
return ZNULL;  
}
```

7.5 Usage of SIP message interfaces

SIP stack provides interfaces to create or set header fields. Here is an example of how to create a CSeq header field:

```
ST_SIP_HDR_CSEQ *pstCseq;  
  
/* create cseq header */  
pstCseq = Sip_CreateMsgHdr(pstEvt->pstMsg, EN_SIP_HDR_CSEQ);  
pstCseq->dwCseqVal = 100;  
pstCseq->stMethod.ucPres = ZTRUE;  
pstCseq->stMethod.ucType = EN_SIP_METHOD_INVITE;
```

8. About the Demo Program

The demo program is mainly used to test the codec and codec related performance of the SIP stack. There are two versions of the demo program, Debug and Release which can help users to make a comparison of performance between the two versions.

In order to make users focus on the testing itself, there are many header descriptions removed from the demo program. For example, sip_abnf_decode.h only provides the prototype of function Sip_DecodeMsg.

In addition, the demo program does not provide options on codec. It will make a complete codec of all headers. So you must pay special attention to this when you make a compare. As for the SDP messages, all their fields will be encoded and decoded completely.

At last it should be pointed out that this demo program is not the optimized version of our SIP stack. All the data do not show the best performance of the stack. The current version of SIP stack is implemented according to the syntax specified by RFC3261. If the message parsing is wrong please check whether the syntax of the testing message is right.

The demo program is a project of Visual C++6.0. Users can modify the project settings to select the DEBUG version or RELEASE version.

8.1 About the Startup

The command used to start the demo program is included in the main function in sipt_msg.c.

By selecting Sipt_FmsgRun, Sipt_MmsgRun or Sipt_MmsgPerformRun(30000), you can choose what to test.

The main steps to start the demo program in the main function are:

- Start ZOS system, Zos_SysInit
- Initialization of ABNF in SDP, Sdp_AbnfInit
- Initialization of ABNF in SIP, Sip_AbnfInit
- Run the testing program
- Shut down ZOS system
- Exit the demo program

```
ZINT main()
{
    /* system init */
    if (Zos_SysInit() != ZOK)
    {
        return -1;
    }

    /* sdp abnf init */
    if (Sdp_AbnfInit() != ZOK)
    {
        Zos_SysDestroy();
        return -1;
    }

    /* abnf init */
    if (Sip_AbnfInit() != ZOK)
    {
        Sdp_AbnfDestroy();
        Zos_SysDestroy();
        return -1;
    }

    /* run file message test */
    Tsip_FmsgRun();

    /* run memory message test
    Tsip_MmsgRun(); */

    /* run memory message perform test
    Tsip_MmsgPerformRun(30000); */

    getchar();

    /* system destroy */
    Zos_SysDestroy();

    return 0;
}
```

8.2 Memory Messages

In the demo program, we provide fourteen kinds of testing messages including six request messages, seven response messages and a message carrying several message bodies. Some of the testing messages also carry SDP messages with them.

Users can add a message by modifying code in `sipt_mmsg.c`. For example:

```
ZCHAR *m_acTsipMemMsgRequest1 =
{
    "ACK sip:009198400000@218.115.159.20:5060;user=phone
SIP/2.0\r\n"
    "Via: SIP/2.0/UDP 218.115.159.156:5060;"
    "branch=z9hG4bK8ca5821c3fdf4cb3ea2b6b921fb68949\r\n"
    "From:
<sip:312345678@218.115.159.106:5060>;tag=003500521121\r\n"
    "To: <sip:009198400000@218.115.159.86>;tag=1434385583\r\n"
    "Call-ID: 851982_10244-1@218.115.159.156\r\n"
    "CSeq: 2 ACK\r\n"
    "Max-Forwards: 68\r\n"
    "Content-Length: 0\r\n\r\n"
};
```

Modification of the record of the configuration management of the testing messages:

```
/* sipit test files list */
ZCHAR **m_appcTsipMemMsgs[] =
{
    &m_acTsipMemMsgRequest1,
    &m_acTsipMemMsgRequest2,
    &m_acTsipMemMsgRequest3,
    &m_acTsipMemMsgRequest4,
    ...
    &m_acTsipMemMsgOtherMsg1,
    ZNULL
};
```

`Sipt_MmsgRun` is a function used to test the file messages.

Users can modify code in the function `ZINT Sipt_Mmsg(ZCHAR *pcMemMsg)` to decide whether to carry out encoding after decoding. Users can also decide how to print the testing message.

8.3 File Messages

Testing messages of SIP interoperability test events, are provided in the demo program. These testing messages are located in the directory `msg/sipit` which is included in the same directory as where the demo program is.

SIP interoperability test events are gatherings of SIP implementors to test interoperability of their creations.

For detailed information please see: <http://www.cs.columbia.edu/sip/sipit/>

For online messages please see: <http://www.cs.columbia.edu/sip/sipit/testmsg.html>

Note that there are a couple of messages in sipit having severe syntax errors. We have corrected some of them. In the final version, we will consider to support the message parsing of these syntax errors.

Testing codes in sipit_fmsg.c:

```
/* sipit test files list */
ZCHAR *m_apcSipitFiles[] =
{
    "../..msg/sipit/test1",
    "../..msg/sipit/test2",
    ...
    "../..msg/sipit/test40",
    "../..msg/sipit/test41",
    "../..msg/sipit/test42",
    ZNULL
};
```

Sipt_FmsgRun is the function used to test the file messages.

Users can modify the code in ZINT Sipt_Fmsg(FILE *pstFile) to decide whether to carry out encoding after decoding. Users can also decide how to print the testing message.

8.4 Performance Analysis

In the demo program, we provide a program to test the performance of SIP messages. One way is to encode and decode a message repeatedly

Example: A testing message is defined in sipit_mmsg.c:

```
ZCHAR *m_acTsipMemMsgPerformInvite1 =
{
    "INVITE sip:172.19.19.61:5063 SIP/2.0\r\n"
    "To: sip:172.19.19.61:5063\r\n"
    "From: sip:caller@university.edu\r\n"
    "Call-ID: 0ha0isndaksdj@10.0.0.1\r\n"
    "CSeq: 8 INVITE\r\n"
    "Via: SIP/2.0/UDP 172.19.19.61:5060\r\n"
    "Content-Type: application/sdp\r\n"
    "Content-Length: 162\r\n"
    "\r\n"
    "v=0\r\n"
    "o=mhandley 29739 7272939 IN IP4 126.5.4.3\r\n"
    "s=-\r\n"
    "c=IN IP4 135.180.130.88\r\n"
    "t=0 0\r\n"
    "m=audio 49210 RTP/AVP 0 12\r\n"
    "m=video 3227 RTP/AVP 31\r\n"
    "a=rtpmap:31 LPC/8000\r\n\r\n"
};
```

By calling `ZINT Sipt_MmsgPerformRun(ZINT iTimes)` and inputting the test times, the testing program is able to run.

```
/* sip test run memory message perform test */
ZINT Tsip_MmsgPerformRun(ZINT iTimes)
{
    struct timeb stStart, stEnd;
    ZINT i, iSecond, iMicroSecond, iErrorNum = 0;

    /* get start time */
    ftime(&stStart);

    for (i = 0; i < iTimes; i++)
    {
        if (Tsip_Mmsg(m_acTsipMemMmsgPerformInvite1) != ZOK)
        {
            iErrorNum++;
            Zos_Printf("test fail message no %d.\r\n", i);
        }
    }

    /* get end time */
    ftime(&stEnd);

    iSecond = stEnd.time - stStart.time;
    iMicroSecond = stEnd.millitm - stStart.millitm;
    if (iMicroSecond < 0)
    {
        iSecond--;
        iMicroSecond += 1000;
    }

    Zos_Printf("sip perform test times: %d cost time: %ds:%dms\r\n",
              iTimes, iSecond, iMicroSecond);

    return ZOK;
}
```

For the purpose of comparison, we use the same equipment (Celeron mobile 1.3G, 384 RAM) **to send the same SIP messages and** Juphoon SIP stack has been set to completely parse all the header field and the SDP message body.

We have tested thirteen messages including INVITE, 180, 183, 200, ACK, BYE, INFO, PRACK , INVITE(ISUP), 200(BYE), 200(INFO) , 200(PRACK1), and 200(PRACK2).

We made **Juphoon SIP** ABNF Codec and **osip** ABNF Codec to perform the encoding and decoding for 30,000 times (Codec DV, Debug codec; Decode DV, Debug codec; Codec RV,

Release codec; Decode RV, Release code). The figure below shows the lengths of time Juphoon SIP ABNF Codec and osip ABNF Codec used to complete the 30,000 times:

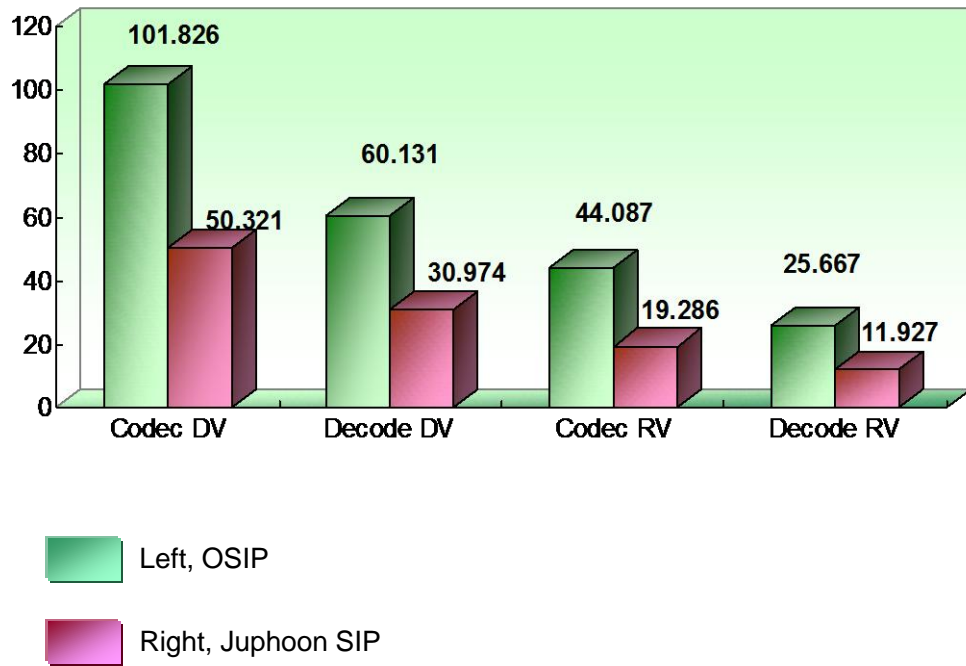


Figure 8-1 Comparison of codec performance between Juphoon SIP and osip

9. Other Products

Juphoon also provides other software products, including operating system service platforms, protocol software, ABNF compiler, tools for testing protocol software. For more detailed information please visit our website <http://www.juphoon.com>.

9.1 Operating System Service Platform

ZOS (Zero Operating System) is an operating system service platform provided by Juphoon. It has a set of unified abstract interfaces that make it be able to support many operating systems. Through ZOS, software can be independent of specified processors, compilers, operating systems and etc. In addition, ZOS strengthens the system services, providing functions as task management, message queues, timer management, memory management, data buffers, log management. It also abstract many protocol related service interfaces, such as ABNF and ASN.1 codec libraries.

9.2 Protocol Software

At present, we provide VoIP and 3G protocol stacks including MGCP, H.248, H.323 and RTP/RTCP. We also help our customers to develop protocol applications and upper business software.

Juphoon also provides application software based on protocols, such as SIP User Agent, RTSP User Agent as well as professional middleware as SIP Phone Framework, IMS PoC Client Framework .

9.3 Compilers

Juphoon provides a compiler which can generate source code in C language automatically according to ABNF syntax. Meanwhile, we are developing a compiler based on ASN.1.

9.4 Testing tools

Juphoon provides testing tools including:

- ABNF2TEST which can automatically produce lots of complicated testing tools according to ABNF
- ABNF2HTML which can produce tools in other language according to ABNF
- Call Engine for the tests of call flows of protocols
- CUNIT
- ASM2BCODE which can produce SigComp byte codes