Abstract

SIP networks use signalling monitoring tools to diagnose user reported problem and for regression testing if network or client software is upgraded. As networks grow and become interconnected, including connection via transit networks, it becomes impractical to predict the path that SIP signalling will take between clients, and therefore impractical to monitor SIP signalling end-to-end.

This draft describes requirements for adding an indicator to the SIP protocol which can be used to mark signalling as of interest to logging. Such marking will typically be applied as part of network testing controlled by the network operator and not used in regular client signalling. However, such marking can be carried end-to-end including the SIP terminals, even if a session originates and terminates in different networks.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on December 15, 2013.
1. Introduction

If users experience problems with setting up sessions using SIP, their service provider needs to find out why by examining the SIP signalling. Also, if network or client software or hardware is upgraded regression testing is needed. Such diagnostics apply to a small proportion of network traffic and can apply end-to-end, even if signalling crosses several networks possibly belonging to several different network operators. It may not be possible to predict the path through those networks in advance, therefore a mechanism is
needed to mark a session as being of interest to enable SIP entities along the signalling path to provide diagnostic logging. This draft describes the requirements for such a 'log me' marker for SIP signalling.

2. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

3. Motivating Scenario

Signalling for SIP session setup can cross several networks, and these networks may not have common ownership and also may be in different countries. If a single operator wishes to perform regression testing or fault diagnosis end-to-end, the separate ownership of networks that carry the signalling and the explosion in the number of possible signalling paths through SIP entities from the originating to the terminating user make it impractical to pre-configure logging of an end-to-end SIP signalling of a session of interest.

The figure below shows an example of a signalling path through multiple networks.
4. Skeleton Diagnostic Procedure

The skeleton diagnostic procedure is as follows:

- The user’s terminal is placed in debug mode. The terminal logs its own signalling and inserts a log me marker into SIP requests for session setup.

- All SIP entities that the signalling traverses, from the first proxy the terminal connects to at the edge of the network to the destination client terminal, can detect that the log me marker is present and can log SIP requests and responses that contain the marker if configured to do so.

- Subsequent responses and requests in the same dialog are logged.

- Logging stops, either because the dialog has ended or because a 'stop event', typically expiry of a certain amount of time, occurred.

- The user’s terminal and any other SIP entity that has logged signalling sends logs to a server that is co-ordinating diagnostics.

5. Requirements for a Log Me Marker

- REQ1: It shall be possible to mark a SIP request or response as of interest for logging by inserting a log me marker. This is known as log-me marking.

- REQ2: It shall be possible for a log-me marker to cross network boundaries.
o REQ3: A log-me marker is most effective if it passes end-to-end. However, source networks should behave responsibly and not leave it to a downstream network to detect and remove a marker that it will not use. A log-me marker should be removed at trust domain boundaries.

o REQ4: SIP entities should log SIP requests or responses with a log-me marker.

o REQ5: If a UA receives a request with a log-me marker, it shall echo that log-me marker in responses to that request.

o REQ6: A SIP proxy may perform log-me marking of requests and responses. Typical cases where a proxy needs to perform log-me marking are when a UA has not marked a request and when responses received on a dialog of interest for logging do not contain a log-me marker. In these cases, the entity that performs log-me marking is stateful inasmuch as it must remember when a dialog is of interest for logging.

o REQ7: For SIP proxies, logging of SIP requests that contain a log-me marker may be stateless. For example, it is not required for a SIP entity to maintain state of which SIP requests contained a log-me marker in order to log responses to those requests. Echoing a log-me marker in responses is the responsibility of the UA that receives a request.

o REQ8: A log-me marker may include an identifier that indicates the test case that caused it to be inserted, known as a test case identifier. The test case identifier does not have any impact on session setup, it is used by the diagnostic server to collate all logged SIP requests and responses to the initial SIP request in a dialog or standalone transaction. The Session-ID described in I-D.ietf-insipid-session-id-reqts [I-D.ietf-insipid-session-id-reqts] could be used as the test case identifier but it would be useful for the UA to log a human readable name together with this Session-ID when it performs log me marking of an initial SIP request.

o REQ9: A log-me marker may include a locator of the server that collects logs. This locator is known as the diagnostic server identifier and may be an address of a server. A SIP entity can use the diagnostic server identifier to send collected logs to the diagnostic server.

6. Security Considerations
All drafts are required to have a security considerations section. See RFC 3552 [RFC3552] for a guide.

6.1. Trust Domain

Since a log me marker may cause a SIP entity to log the SIP header and body of a request or response, the log me marker should be removed at a trust domain boundary. If a prior agreement to log sessions exists with the net hop network then the log me marker might not be removed.

6.2. Security Threats

The log me marker is not sensitive information, although it will sometimes be inserted because a particular device is experiencing problems.

The presence of a log me marker will cause some SIP entities to log signalling. Therefore, this marker must be removed at the earliest opportunity if it has been incorrectly inserted.

Activating a debug mode affects the operation of a terminal, therefore it must be supplied by an authorized server to an authorized terminal, it must not be altered in transit, and it must not be readable by an unauthorized third party.

Logged signalling is privacy-sensitive data, therefore it must be passed to an authorized server, it must not be altered in transit, and it must not be readable by an unauthorized third party.

7. Potential Solutions

This section describes potential solutions to the logme requirements.

7.1. Solution A: LogMe header field

A new SIP header field, e.g. 'LogMe:', is defined to indicate that a session is of interest to logging. The LogMe header field is inserted by a UA that sends an initial SIP request, and subsequent SIP requests belonging to the same dialog if any, and echoed in responses by the UA that receives the SIP request. The LogMe header field has two header field parameters defined, one free-text name of the test case being performed, and one address of a server where collected logging will be sent after logging has terminated.
(1) INVITE
  LogMe: testCaseName="test01"; debugServer="d1.foocorp.com"
----------------->
(2) INVITE
  Logme:
----------------->
(3) INVITE
  LogMe:
    -------> (to barcorp)

(9) 200 OK
(10) 200 OK
    LogMe:
----------------->
(8) 200 OK
    LogMe:
<----------------- (from barcorp)
(11) ACK

Proxy   Registrar   Bob         Debug Server
pl.barcorp.com  r1.barcorp.com  u1.barcorp.com  d1.barcorp.com
(3) INVITE
  Logme:
    -------> (from foocorp)
(4) INVITE
  Logme:
----------------->
(5) INVITE
  Logme:
    (u1 copies LogMe: to SIP response)
----------------->
(6) 200 OK
  LogMe:
<-----------------
(7) 200 OK
  LogMe:
<-----------------
(8) 200 OK
7.2. Solution B: New Value for purpose header field parameter in Call-Info:

A new value is defined for the purpose header field parameter used in Call-Info header field.

The Call-Info: header field is defined in clause 20.9 of RFC 3261 [RFC3261].

The Call-Info: header field can be included in methods INVITE, OPTIONS, REGISTER (Table 2: Summary of header fields, A--O in RFC 3261 [RFC3261] clause 20.1), INFO (RFC 6086 [RFC6086]), MESSAGE (RFC 3428 [RFC3428]), PUBLISH (RFC 3903 [RFC3903]), and UPDATE (RFC 3311 [RFC3311]), and in responses to those methods. Call-Info: header field cannot be included in methods NOTIFY, SUBSCRIBE, PRACK, or REFER.

7.3. Solution C: New 'debug' header field parameter to be used in Session-ID header field

A new header field parameter called debug is defined to be used with the Session-ID header field (described in I-D.ietf-insipid-session-id-reqts [I-D.ietf-insipid-session-id-reqts]).
Figure 4: Signalling example for the Session-ID: header field parameter solution

7.4. Comparison of Potential Solutions

The table below summarizes the features of each potential solution. Other solutions are not excluded.

<table>
<thead>
<tr>
<th>Solution</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Log-Me: header field</td>
</tr>
<tr>
<td>B</td>
<td>New value for the purpose parameter of the Call-Info header field e.g. &quot;debug&quot;</td>
</tr>
<tr>
<td>C</td>
<td>New header field parameter for Session-ID header field e.g. debug</td>
</tr>
</tbody>
</table>

Table 1: Summary comparison of potential solutions
8. References

8.1. Normative References


8.2. Informative References

[I-D.ietf-insipid-session-id-reqts]


Appendix A. Additional Stuff

This becomes an Appendix.
Author’s Address

Peter Dawes
Vodafone Group
The Connection
Newbury, Berkshire  RG14 2FN
UK

Phone: +44 7717 275009
Email: peter.dawes@vodafone.com
Abstract

This document describes an end-to-end Session Identifier for use in IP-based Multimedia Communication systems that enables endpoints, intermediate devices, and management systems to identify a session end-to-end, associate multiple endpoints with a given multipoint conference, track communication sessions when they are redirected, and associate one or more media flows with a given communication session.

Status of this Memo

This Internet-Draft is submitted to IETF in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html

This Internet-Draft will expire on January 15, 2014.

Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.
This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction...................................................2
2. Conventions used in this document..............................3
3. Session Identifier Requirements and Use Cases..................3
4. Constructing and Conveying the Session Identifier..............4
   4.1. Constructing the Session Identifier.......................4
   4.2. Conveying the Session Identifier..........................4
5. Transmitting the Session Identifier in SIP.....................5
6. Endpoint Behavior..............................................6
7. Processing by Intermediaries...................................8
8. Associating Endpoints in a Multipoint Conference...............9
9. Various Call Flow Operations Utilizing the Session ID...........9
   9.1. Basic Session ID Construction with 2 UUIDs.................9
   9.2. Basic Call Transfer using REFER..........................10
   9.3. Basic Call Transfer using reINVITE.......................12
   9.4. Single Focus Conferencing.................................13
   9.5. Single Focus Conferencing using WebEx....................15
   9.6. Cascading Conference Bridge Support for the Session ID...16
   9.7. Basic 3PCC for two UAs..................................17
   9.8. Session ID Handling in 100 Trying SIP Response and CANCEL
        Request.......................................................18
        9.8.1. Session ID Handling in a 100 Trying SIP Response....18
        9.8.2. Session ID in a CANCEL SIP Request.................19
   9.9. Session ID in an out-of-dialog REFER Transaction.........20
10. Compatibility with a Previous Implementation...............21
11. Security Considerations......................................22
12. IANA Considerations........................................23
   12.1. Registration of the "Session-ID" Header Field...........23
   12.2. Registration of the "remote" Parameter..................23
13. Acknowledgments..............................................23
14. References...................................................23
   14.1. Normative References....................................23
   14.2. Informative References..................................24
15. Author’s Addresses..........................................25

1. Introduction

IP-based multimedia communication systems like SIP [RFC3261] and H.323 [H.323] have the concept of a "call identifier" that is
globally unique. The identifier is intended to represent an end-to-end communication session from the originating device to the terminating device. Such an identifier is useful for troubleshooting, session tracking, and so forth.

Unfortunately, there are a number of factors that contribute to the fact that the current call identifiers defined in SIP and H.323 are not suitable for end-to-end session identification. A fundamental issue in protocol interworking is the fact that the syntax for the call identifier in SIP and H.323 is different between the two protocols. This important fact makes it impossible for call identifiers to be exchanged end-to-end when a network utilizes one or more session protocols.

Another reason why the current call identifiers are not suitable to identify the session end-to-end is that in real-world deployments devices like session border controllers often change the session signaling as it passes through the device, including the value of the call identifier. While this is deliberate and useful, it makes it very difficult to track sessions end-to-end.

This draft presents a new identifier, referred to as the Session Identifier, or "Session ID", and associated syntax intended to overcome the issues that exist with the currently defined call identifiers. The proposal in this document attempts to comply with the requirements specified in [I-D.ietf-insipid-session-id-reqts]. This proposal also has capabilities not mentioned in [RFC5234], shown in call flows in section 10. Additionally, this proposal attempts to account for a previous, proprietary version of a SIP Session ID header, proposing a backwards compatibility of sorts, described in section 11.

2. Conventions used in this document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119] when they appear in ALL CAPS. These words may also appear in this document in lower case as plain English words, absent their normative meanings.

The terms "Session Identifier" and "Session ID" refer to the value of the identifier, whereas "Session-ID" refers to the header used to convey the identifier.

3. Session Identifier Requirements and Use Cases

Requirements and Use Cases for the end-to-end Session Identifier can be found in a separate memo titled "Requirements for an End-to-End Session Identification in IP-Based Multimedia Communication Networks" [I-D.ietf-insipid-session-id-reqts].
4. Constructing and Conveying the Session Identifier

4.1. Constructing the Session Identifier

The Session Identifier is comprised of two RFC 4122 defined UUIDs [RFC4122], with each UUID representing one of the endpoints participating in the session.

The version number in the UUID indicates the manner in which the UUID is generated, such as using random values or using the MAC address of the endpoint. To satisfy the requirement that no user or device information be conveyed, endpoints SHOULD generate version 4 (random) or version 5 (SHA-1) UUIDs.

When generating a version 5 UUID, endpoints or intermediaries MUST utilize the following "name space ID" (see Section 4.3 of RFC4122):

```c

#include <uuid/uuid.h>

const uuid_t NameSpace_SessionID = {
  /* a58587da-c93d-11e2-ae90-f4ea67801e29 */
  0xa58587da,
  0xc93d,
  0x11e2,
  0x90, 0xf4, 0xea, 0x67, 0x80, 0x1e, 0x29
}
```

Further, the "name" to utilize for version 5 UUIDs is the concatenation of the Call-ID header value and the "tag" parameter that appears on the "From" or "To" line associated with the device for which the UUID is created. Once an endpoint generates a UUID for a session, the UUID never changes, even if values originally used as input into its construction change over time.

Intermediaries that insert a Session-ID header into a SIP message on behalf of a sending User Agent MUST utilize version 5 UUIDs to ensure that UUIDs for the communication session are always generated with the same values. If an intermediary does not know the tag value for an endpoint, the intermediary MUST NOT attempt to generate a UUID for that endpoint. Note that if an intermediary is stateless and the endpoint on one end of the call is replaced with another endpoint due to some service interaction, the values used to create the UUID might change and, if so, the intermediary will compute a different UUID.

4.2. Conveying the Session Identifier

The SIP user agent (UA) initially transmitting the SIP request will create a UUID and transmit that to the ultimate destination UA. Likewise, the responding UA will create a UUID and transmit that to the first UA. These two distinct UUIDs form what is referred to as the Session Identifier and is represented in this document in set notation of the form (A,B), where A is UUID value from the UA transmitting a message and B is the UUID value from the intended
recipient of the message, i.e., not an intermediary server along the signaling path. The set \(\{A,B\}\) is equal to the set \(\{B,A\}\), and thus both represent the same Session Identifier.

In the case where only one UUID is known, such as when a UA first initiates a SIP request, the Session ID would be \(\{A\}\), where "A" represents the single UUID value transmitted.

Since SIP sessions are subject to any number of service interactions, SIP INVITE messages might be forked as sessions are established, and since conferences might be established or expanded with endpoints calling in or the conference focus calling out, the construction of the Session Identifier from a set of UUIDs is important.

To understand this better, consider that a UA participating in a communication session might be replaced with another, such as the case where two "legs" of a call are joined together by a PBX. Suppose that UA A and UA B both call UA C. Further suppose that UA C uses a local PBX function to join the call between itself and UA A with the call between itself and UA B. This merged call needs to be identified and identification of such sessions is natural and easily traceable when utilizing UUID values assigned by each entity in the communication session.

In the case of forking, UA A might send an INVITE that gets forked to five different UAs, as an example. A means of identifying each of these separate communication sessions is needed and allowing the set of \(\{A, B_1\}\), \(\{A, B_2\}\), \(\{A, B_3\}\), \(\{A, B_4\}\), and \(\{A, B_5\}\) makes this possible.

For conferencing scenarios, it is also useful to have a two-part Session Identifier where the conference focus specifies one UUID. This might allow for correlation among the participants in a single conference, for example.

How a device acting on Session Identifiers stores, processes, or utilizes the Session Identifier is outside the scope of this document.

5. Transmitting the Session Identifier in SIP

Each session initiated or accepted MUST have a local UA-generated UUID associated with the session. This value MUST remain unchanged throughout the duration of that session.

A SIP UA MUST convey its Session Identifier UUID in all transmitted messages within the same session. To do this, each transmitted message MUST include the Session-ID header. The Session-ID header has the following ABNF [RFC5234] syntax:

```
session-id = "Session-ID" HCOLON local-uuid
```
local-uuid = sess-uuid
remote-uuid = sess-uuid

the productions "SEMI", "EQUAL", and "generic-param" are defined in RFC 3261. The production DIGIT is defined in RFC 5234.

The Session-ID header MUST NOT have more than one "remote" parameter.

The "local-uuid" in the Session-ID header represents the UUID value of the UA transmitting the message. If the UA transmitting the message previously received a UUID value from its peer endpoint, it MUST include that UUID as the "remote" parameter in each message it transmits. For example, a Session-ID header might appear like this:

    Session-ID: ab30317f1a784dc48ff824d0d3715d86;
                remote=47755a9de7794ba387653f2099600ef2

The UUID values are presented as strings of lower-case hexadecimal characters, with the most significant byte of the UUID appearing first.

A UUID having the value of all zeros is a special UUID value. It is used in certain special cases, and hereafter is defined as the "null" UUID value. Either the "local-uuid" field or "remote-uuid" field can have a "null" value.

6. Endpoint Behavior

To comply with this specification, SIP UAs MUST include a Session-ID header-value in all SIP messages transmitted as a part of a communication session. The UUID of the transmitter of the message MUST appear in the "local-uuid" portion of the Session-ID header-value with one exception, mentioned below, and the UUID of the peer device, if known, must appear as the "remote" parameter following the transmitter’s UUID.

Once a UA allocates a UUID value for a communication session, the UA MUST NOT change that UUID value for the duration of the session, including when

- communication attempts are retried due to receipt of 4xx messages or request timeouts;
The exception to including the UUID of the transmitting entity mentioned above is in the case of provisional responses that occur before the destination UA has generated its UUID. The 100 (Trying) response and the 181 (Call Forwarding) response are examples of such provisional responses. In these cases, the sending intermediary places the one known UUID in the remote-uuid field, and leaves the "local-uuid" blank. This placement is always where a UA expects to receive its UUID value in SIP responses.

A non-intermediary UA that receives a Session-ID header MUST take note of the first UUID value (i.e., the "local-uuid") that it receives in the Session-ID header and assume that that is the UUID of the peer endpoint within that communications session. UAs MUST include this received UUID value as the "remote" parameter when transmitting subsequent messages, making sure not to change this UUID value in the process of moving the value internally from the "local-uuid" field to the "remote-uuid" field.

It should be noted that messages received by a UA might contain a "local-uuid" parameter that does not match what the UA expected the far end UA’s UUID to be. This might happen as a result of service interactions by intermediaries and MUST NOT negatively affect the communication session. However, the UA may log this event for the purposes of troubleshooting.

For any purpose the UA has for the Session Identifier, it MUST assume that the Session Identifier is \{A,B\} where "A" is the UUID value of this endpoint (i.e., "local-uuid") and "B" is the UUID value of the peer endpoint (i.e., "remote-uuid"), taken from the most recently received message within this session. Note that when comparing Session Identifiers for equivalence, the identifier \{A,B\} is equal to the set \{B,A\}.

An endpoint MUST assume that the UUID value of the peer UA MAY change at any time due to service interactions. If the UUID value of the peer UA changes, the UA MUST include this new UUID as the "remote" parameter in any subsequent messages.

It is also important to note that if a session is forked by an intermediary in the network, the initiating UA may receive multiple responses back from different endpoints, each of which will contain a different UUID ("local-uuid") value in each response received by this initiating UA. UAs MUST take care to ensure that the correct UUID value is returned in the "remote" parameter when responding to those endpoints.
Cascading MCUs all utilize the same UUID value ("local-uuid" portion of the Session-ID header-value) for all participants of the cascaded conference. An MCU conveys the UUID value to utilize via the "local-uuid" portion of the Session-ID header-value in an INVITE to a second MCU.

7. Processing by Intermediaries

Intermediaries MUST NOT alter the UUID values found in the Session-ID header, except as described in this section.

Intermediary devices that transfer a call, such as by joining together two different "call legs", MUST properly construct a Session-ID header that contains the correct UUID values and correct placement of those values. As described above, the recipient of any message initiated by the intermediary will assume that the first UUID value belongs to the peer endpoint.

If a SIP message having no Session-ID header is received by an intermediary, the intermediary MAY assign a "local-uuid" value to represent the sending endpoint and insert that value into all signaling messages on behalf of the sending endpoint. If the intermediary is aware of a "remote" value that identifies the receiving UA, it MUST insert that value if also inserting the "local-uuid" value.

Devices that initiate communication sessions following the procedures for third party call control MUST fabricate a UUID value that will be utilized only temporarily. Once the responding endpoint provides a UUID value in a response message, the temporary value MUST be discarded and replaced with the endpoint-provided UUID value. Refer to the third-party call control example for an illustration.

Whenever there is a UA that does not implement this specification communicating through a B2BUA, the B2BUA MAY become dialog stateful and insert a UUID value into the Session-ID header on behalf of the UA according to the rules stated in Section 6.

When intermediaries transmit provisional responses, such as 100 Trying, they MUST be consistent with the text in Section 6 that discussed how endpoints are expected to receive a certain UUID value (say, either a "local-uuid" or "remote-uuid", but not both). In provisional responses that have not reached their destination UAS to generate the other UUID value for that endpoint, intermediaries are to place the UAC’s UUID value in the "remote-uuid" portion of the Session-ID header-value, and a null "local-uuid" value.

A CANCEL request sent by an intermediary that has received no previous response from the target UA has a Session-ID constructed exactly like the INVITE to that UA, with only a "local-uuid" value in the Session-ID header-value.
8. Associating Endpoints in a Multipoint Conference

Multipoint Control Units (MCUs) group two or more sessions into a single multipoint conference. The MCU should utilize the same UUID value for each session that is grouped into the same conference. In so doing, each individual session in the conference will have a unique Session Identifier (since each endpoint will create a unique UUID of its own), but will also have one UUID in common with all other participants in the conference.

Intermediary devices, such as proxies or session border controllers, or network diagnostics equipment might assume that when they see two or more sessions with different Session Identifiers, but with one UUID in common, that the sessions are part of the same conference. Note, however, that this assumption of being part of the same conference is not always true. For example, in a SIP forking scenario, there might also be what appears to be multiple sessions with a shared UUID value. This is actually desirable. What is desired is to allow for the association of related sessions. Whether sessions are related because of forking or because endpoints are communicating as a part of a conference does not matter. They are nonetheless related.

9. Various Call Flow Operations Utilizing the Session ID

Seeing something frequently makes understanding easier. With that in mind, we include several call flow examples with the initial UUID and the complete Session ID indicated per message, as well as when the Session ID changes according to the rules within this document during certain operations/functions.

This section is for illustrative purposes only and is non-normative. In the following flows, RTP refers to the Real-time Transport Protocol [RFC3550].

9.1. Basic Session ID Construction with 2 UUIDs

<table>
<thead>
<tr>
<th>Session ID</th>
<th>Alice</th>
<th>B2BUA</th>
<th>Bob</th>
<th>Carol</th>
</tr>
</thead>
<tbody>
<tr>
<td>{A}</td>
<td>----INVITE-----&gt;</td>
<td></td>
<td>----INVITE-----&gt;</td>
<td></td>
</tr>
<tr>
<td>{A}</td>
<td></td>
<td>----INVITE-----&gt;</td>
<td>&lt;---200 OK------</td>
<td></td>
</tr>
<tr>
<td>{B,A}</td>
<td>&lt;---200 OK------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>{A,B}</td>
<td>------ACK------&gt;</td>
<td></td>
<td>------ACK------&gt;</td>
<td></td>
</tr>
<tr>
<td>{A,B}</td>
<td>&lt;----------------RTP----------------&gt;</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 1 - Session ID Creation when Alice calls Bob

General operation of this example:
o The originating transmitter of a SIP message populates the "local-uuid" portion of the Session-ID header-value.

o UA-Alice sends its UUID in the SIP INVITE.

o B2BUA receives an INVITE with a "local-uuid" portion of the Session-ID header-value from UA-Alice, and transmits INVITE towards UA-Bob with an unchanged Session-ID header-value.

o UA-Bob receives Session-ID and adds its "local-uuid" portion of the Session-ID header-value UUID to construct the whole/complete Session-ID header-value, at the same time transferring Alice’s UUID unchanged to the "remote-uuid" portion of the Session-ID header-value in the 200 OK SIP response.

o B2BUA receives the 200 OK response with a complete Session-ID header-value from UA-Bob, and transmits 200 OK towards UA-Alice with an unchanged Session-ID header-value.

o UA-Alice, upon reception of the 200 OK from the B2BUA, transmits the ACK towards the B2BUA. The construction of the Session-ID header-value in this ACK is that of Alice’s UUID is the "local-uuid", and Bob’s UUID populates the "remote-uuid" portion of the header-value.

o B2BUA receives the ACK with a complete Session-ID header-value from UA-Alice, and transmits ACK towards UA-Bob with an unchanged Session-ID header-value.

9.2. Basic Call Transfer using REFER

From the example built within Section 9.1 (the basic session ID establishment), we proceed to this 'Basic Call Transfer using REFER' example.

<table>
<thead>
<tr>
<th>Session ID</th>
<th>Alice</th>
<th>B2BUA</th>
<th>Bob</th>
<th>Carol</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>{{{{}}}}</td>
<td>&lt;-------------RTP-----------&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>{B,A}</td>
<td>{{{{}}}}</td>
<td>&lt;---reINVITE---&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>{B,A}</td>
<td>{{{{}}}}</td>
<td>&lt;---reINVITE----&gt; (puts Alice on Hold)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>{A,B}</td>
<td>{{{{}}}}</td>
<td>-----200 OK----&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>{A,B}</td>
<td>{{{{}}}}</td>
<td>-----200 OK----&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>{B,A}</td>
<td>{{{{}}}}</td>
<td>&lt;-----ACK--------&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>{B,A}</td>
<td>{{{{}}}}</td>
<td>&lt;-----ACK--------&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>{B,A}</td>
<td>{{{{}}}}</td>
<td>&lt;-----REFER------&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>{B,A}</td>
<td>{{{{}}}}</td>
<td>&lt;-----REFER------&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>{A,B}</td>
<td>{{{{}}}}</td>
<td>-----200 OK----&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>{A,B}</td>
<td>{{{{}}}}</td>
<td>-----200 OK----&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>{A,B}</td>
<td>{{{{}}}}</td>
<td>-----NOTIFY----&gt;</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
General operation of this example:

Starting from the existing Alice/Bob call described in Figure 1 of this document, which established an existing Session-ID header-value...

- UA-Bob reINVITEs Alice to call Carol, using a REFER transaction, as described in [RFC3515]. UA-Alice is initially put on hold, then told in the REFER who to contact with a new INVITE, in this case UA-Carol. This Alice-to-Carol dialog will have a new Call-ID, therefore it requires a new Session-ID header-value. The wrinkle here is we can, and will, use Alice’s UUID from her existing dialog with Bob in the new INVITE to Carol.

- UA-Alice retains her UUID from the Alice-to-Bob call {A} when requesting a call with UA-Carol. This is placed in the "local-uuid" portion of the Session-ID header-value, with no "remote-uuid" value (because Carol’s UA has not yet received the UUID value). This same UUID traverses the B2BUA unchanged.

- UA-Carol receives the INVITE with a Session ID UUID {A}, moves this UUID value into the "remote-uuid" portion of the Session-ID header-value and creates its own UUID {C} and places this value in the "local-uuid" portion of the Session-ID header-value. This combination forms a full Session ID {C,A} in the 200 OK to the

---

**Figure 2 - Call Transfer using REFER**

| (A,B) | -----NOTIFY----- |               |
| (B,A) | -----INVITE----- |         |
| (B,A) | <----200 OK----- |         |
| (A)   | <----200 OK----- |         |
| (A)   | <----200 OK----- |         |
| (C,A) | <----200 OK----- |         |
| (C,A) | <----200 OK----- |         |
| (A,C) | <-----ACK------- |         |
| (A,C) | <-----ACK------- |         |
|       | <======================RTP======================> |
| (A,B) | -----NOTIFY----- |               |
| (A,B) | -----NOTIFY----- |               |
| (B,A) | <----200 OK----- |               |
| (B,A) | <----BYE-------- |               |
| (B,A) | <----BYE-------- |               |
| (A,B) | <----200 OK----- |               |
| (A,B) | <----200 OK----- |               |
INVITE. This Session-ID header-value traverses the B2BUA unchanged towards UA-Alice.

- UA-Alice receives the 200 OK with the Session ID \( \{C,A\} \) and both responds to UA-Carol with an ACK (just as in Figure 1 - switches places of the two UUID fields), and generates a NOTIFY to Bob with a Session ID \( \{A,B\} \) indicating the call transfer was successful.

- It does not matter which UA terminates the Alice-to-Bob call; Figure 2 shows UA-Bob doing this transaction.

### 9.3. Basic Call Transfer using reINVITE

From the example built within Section 9.1 (the basic session ID establishment), we proceed to this ‘Basic Call Transfer using reINVITE’ example.

Alice is talking to Bob. Bob pushes a button on his phone to transfer Alice to Carol via the B2BUA (using reINVITE).

<table>
<thead>
<tr>
<th>Session ID</th>
<th>---</th>
<th>Alice</th>
<th>B2BUA</th>
<th>Bob</th>
<th>Carol</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>&lt;==========RTP==========&gt;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( {B,A} )</td>
<td>{B,A}</td>
<td>&lt;----reINVITE----&gt;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( {A,B} )</td>
<td>{A,B}</td>
<td>------200 OK------&gt;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( {B,A} )</td>
<td>{B,A}</td>
<td>&lt;-----ACK-------&gt;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( {A} )</td>
<td>{A}</td>
<td>&lt;-----INVITE--------&gt;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( {C,A} )</td>
<td>{C,A}</td>
<td>&lt;----200 OK---------------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( {A,C} )</td>
<td>{A,C}</td>
<td>------ACK----------------------&gt;</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| \( \{B,A\} \) | {B,A} |<-----BYE-------| | | |
| \( \{A,B\} \) | {A,B} |------200 OK---->| | | |

**Figure 3 - Call transfer using reINVITE**

General operation of this example:

- We assume the call between Alice and Bob from Section 9.1 is operational with Session ID \( \{A,B\} \).

- Bob sends a reINVITE to Alice (with the Session-ID "local-uuid" = Bob’s UUID and "remote-uuid" = Alice’s UUID), informing her to transfer her existing call to Carol.
The B2BUA intercepts this reINVITE and sends a new INVITE with Alice’s UUID ("local-uuid" = "A") to Carol.

Carol receives the INVITE and accepts the request and adds her UUID (C) to the Session ID for this session ("local-uuid" = "C", "remote-uuid" = "A").

Bob terminates the call with a BYE using the Session ID ("local-uuid" = "B", "remote-uuid" = "A"). The B2BUA responds to Bob since Alice and Carol are now in a new call.

9.4. Single Focus Conferencing

Multiple users call into a conference server (say, an MCU) to attend one of many conferences hosted on or managed by that server. Each user has to identify which conference they want to join, but this information is not necessarily in the SIP messaging. It might be done by having a dedicated address for the conference or via an IVR, as assumed in this example. Each user in this example goes through a two-step process of signaling to gain entry onto their conference call.

```
<table>
<thead>
<tr>
<th>Session ID</th>
<th>Alice</th>
<th>Conference</th>
<th>Bob</th>
<th>Carol</th>
</tr>
</thead>
<tbody>
<tr>
<td>{A}</td>
<td>---INVITE---</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(M1,A)</td>
<td>&lt;---200 OK---</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(A,M1)</td>
<td>-----ACK-----</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(M',A)</td>
<td>&lt;---reINVITE--</td>
<td>(to change the</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(A,M')</td>
<td>-----200 OK----</td>
<td>UUID to M'</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(M',A)</td>
<td>-----ACK-----</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>{B}</td>
<td>&lt;---INVITE---</td>
<td>&lt;-----200 OK----</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(M2,B)</td>
<td>-----ACK-----</td>
<td>&lt;-----RTP-----</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(B,M2)</td>
<td>&lt;-----RTP-----</td>
<td>&lt;-----reINVITE-----</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(M',B)</td>
<td>(to change the</td>
<td>UUID to M'</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(B,M')</td>
<td>&lt;-----200 OK----</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(M',B)</td>
<td>&lt;-----ACK-----</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>{C}</td>
<td>&lt;--------------------INVITE-----</td>
<td>&lt;--------------------200 OK----&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(M3,C)</td>
<td>&lt;--------------------200 OK----&gt;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(C,M3)</td>
<td>&lt;--------------------ACK-----</td>
<td>&lt;--------------------RTP-----</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(M',C)</td>
<td>(to change the</td>
<td>UUID to M'</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(C,M')</td>
<td>&lt;--------------------reINVITE----</td>
<td>&lt;--------------------200 OK----&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(M',C)</td>
<td>&lt;--------------------ACK-----</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
General operation of this example:

Alice calls into a conference server to attend a certain conference. This is a two-step operation since Alice cannot include the conference ID at this time and/or any passcode in the INVITE request. The first step is Alice’s UA calling another UA to participate in a session. This will appear to be similar as the call-flow in Figure 1 (in section 9.1). What is unique about this call is the second step: the conference server calls back with a reINVITE request with its second UUID, but maintaining the UUID Alice sent in the first INVITE. This subsequent UUID from the conference server will be the same for each UA that calls into this conference server participating in this same conference bridge/call, which is generated once Alice typically authenticates and identifies which bridge she wants to participate on.

- Alice sends an INVITE to the conference server with her UUID {A}.
- The conference server responds with a 200 OK response which includes a temporary UUID ("M1") as the "local-uuid" and a "remote-uuid" = "A".

NOTE: this ‘temporary’ UUID is a real UUID; it is only temporary to the conference server because it knows that it is going to generate another UUID to replace the one just send in the 200 OK.

- Once Alice, the user, gains access to the IVR for this conference server, she enters a specific conference ID and whatever passcode (if needed) to enter a specific conference call.
- Once the conference server is satisfied Alice has identified which conference she wants to attend (including any passcode verification), the conference server reINVITEs Alice to the specific conference and includes the Session-ID header-value of "local-uuid" = "M’" (and "remote-uuid" = "A") for that conference. All valid participants in the same conference will receive this same UUID for identification purposes and to better enable monitoring, and tracking functions.
- Bob goes through this two-step process of an INVITE transaction, followed by a reINVITE transaction to get this same UUID ("M’") for that conference.
- In this example, Carol (and each additional user) goes through the same procedures and steps as Alice and Bob to get on this same conference.
9.5. Single Focus Conferencing using WebEx

Alice, Bob and Carol call into same Webex conference.

<table>
<thead>
<tr>
<th>Session ID</th>
<th>Conference Focus</th>
<th>Bob</th>
<th>Carol</th>
</tr>
</thead>
<tbody>
<tr>
<td>{M}</td>
<td><strong>HTTPS</strong>***</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Transaction</td>
<td></td>
<td></td>
</tr>
<tr>
<td>{A,M}</td>
<td>------INVITE-----</td>
<td>-----</td>
<td>-------</td>
</tr>
<tr>
<td>{M,A}</td>
<td>----200 OK------</td>
<td>-----</td>
<td>-------</td>
</tr>
<tr>
<td></td>
<td>------ACK-------</td>
<td>-----</td>
<td>-------</td>
</tr>
<tr>
<td></td>
<td>======RTP======</td>
<td>-----</td>
<td>-------</td>
</tr>
<tr>
<td>{M}</td>
<td><strong>HTTPS</strong>***</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Transaction</td>
<td></td>
<td></td>
</tr>
<tr>
<td>{B,M}</td>
<td>------INVITE-----</td>
<td>-----</td>
<td>-------</td>
</tr>
<tr>
<td>{M,B}</td>
<td>----200 OK------</td>
<td>-----</td>
<td>-------</td>
</tr>
<tr>
<td></td>
<td>------ACK-------</td>
<td>-----</td>
<td>-------</td>
</tr>
<tr>
<td></td>
<td>======RTP======</td>
<td>-----</td>
<td>-------</td>
</tr>
<tr>
<td>{M}</td>
<td>----------------INVITE-----</td>
<td>-----</td>
<td>-------</td>
</tr>
<tr>
<td>{C,M}</td>
<td>-----------------200 OK------</td>
<td>-----</td>
<td>-------</td>
</tr>
<tr>
<td>{M,C}</td>
<td>------------------ACK------&gt;</td>
<td>-----</td>
<td>-------</td>
</tr>
<tr>
<td></td>
<td>===============RTP====&gt;</td>
<td>-----</td>
<td>-------</td>
</tr>
</tbody>
</table>

Figure 5 - Single Focus Webex Conference

General operation of this example:

- Alice communicates with Webex server with desire to join a certain meeting, by meeting number; also includes UA-Alice's contact information (phone number, URI and/or IP address, etc.) for each device she wants for this conference call. For example, the audio and video play-out devices could be separate units.

- Conference Focus server sends INVITE (Session-ID header-value "local-uuid" = M, where M equals the "local-uuid" for each participant on this conference bridge) to UA-Alice to start session with the of that server for this A/V conference call.

- Upon receiving the INVITE request from the conference focus server, Alice responds with a 200 OK. Her UA moves the "local-uuid" unchanged into the "remote-uuid" field, and generates her
own UUID and places that into the "local-uuid" field to complete the Session-ID construction.

- Bob and Carol perform same function to join this same A/V conference call as Alice.

9.6. Cascading Conference Bridge Support for the Session ID

To expand conferencing capabilities requires cascading conference bridges. A conference bridge, or MCU, needs a way to identify itself when contacting another MCU. RFC 4579 [RFC4579] defines the 'isfocus' Contact: header parameter just for this purpose.

Session ID

--- MCU-1 MCU-2 MCU-3 MCU-4

{M'} ----INVITE------>
{J,M'} <---200 OK------
{M',J} -----ACK------>

Figure 6 - MCUs Communicating Session ID UUID for Bridge

Regardless of which MCU (1 or 2) a UA contacts for this conference, once the above exchange has been received and acknowledged, the UA will get the same M' UUID from the MCU for the complete Session ID.

A more complex form would be a series of MCUs all being informed of the same UUID to use for a specific conference. This series of MCUs can either be informed

- All by one MCU (that initially generates the UUID for the conference),

- The one MCU that generates the UUID informs one or several MCUs of this common UUID, and they inform downstream MCUs of this common UUID each will be using for this one conference, or

Session ID

--- MCU-1 MCU-2 MCU-3 MCU-4

{M'} ----INVITE------>
{J,M'} <---200 OK------
{M',J} -----ACK------>

{M'} ---------------INVITE------>
{K,M'} <------------------200 OK------>
{M',K} ---------------ACK------>

{M'} ----------------------------------INVITE------>
{L,M'} <----------------------------------200 OK------>
{M',L} ----------------------------------ACK------>
Figure 7 - MCU Communicating Session ID UUID to More than One

General operation of this example:

- The MCU generating the Session ID UUID communicates this in a separate INVITE, having a Contact header with the ‘isfocus’ header parameter. This will identify the MCU as what RFC 4579 conference-aware SIP entity.

- The MCU that is contacted, i.e., the UAS MCU, does not populate or complete the Session-ID header value. The UAS MCU transmits a 200 OK response acknowledging it is to respond with this M’ UUID to all requests for the designated conference.

- An MCU that receives this M’ UUID in an inter-MCU transaction, can communicate the M’ UUID in a manner in which it was received (though this time this second MCU would be the UAC MCU), unless local policy dictates otherwise.

9.7. Basic 3PCC for two UAs

External entity sets up call to both Alice and Bob for them to talk to each other.

<table>
<thead>
<tr>
<th>Session ID</th>
<th>Alice</th>
<th>B2BUA</th>
<th>Bob</th>
<th>Carol</th>
</tr>
</thead>
<tbody>
<tr>
<td>{X}</td>
<td>------INVITE------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>{A,X}</td>
<td>-----200 OK-----&gt;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>{A}</td>
<td></td>
<td>-----INVITE-----&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>{B,A}</td>
<td></td>
<td>&lt;---200 OK------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>{A,B}</td>
<td></td>
<td>&lt;-----ACK-------</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;--------------RTP--------------&gt;</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 8 - 3PCC initiated call between Alice and Bob

General operation of this example:

- Some out of band procedure directs a B2BUA (or other SIP server) to have Alice and Bob talk to each other.

- The SIP server INVITEs Alice to a session and uses a temporary UUID {X}.

- Alice receives and accepts this call set-up and includes her UUID {A} in the Session ID, now {A,X}.

- The SIP server uses Alice’s UUID {A}, and discards its own {X} to INVITE Bob to the session as if this came from Alice originally.
o Bob receives and accepts this INVITE and adds his own UUID \{B\} to the Session ID, now \{B,A\} for the response.

o And the session is established.

9.8. Session ID Handling in 100 Trying SIP Response and CANCEL Request

The following two subsections show examples of the Session ID for a 100 Trying response and a CANCEL request in a single call-flow.

9.8.1. Session ID Handling in a 100 Trying SIP Response

The following 100 Trying response is taken from an existing RFC, from [RFC5359] Section 2.9 ("Call Forwarding - No Answer").

<table>
<thead>
<tr>
<th>Session ID</th>
<th>Alice</th>
<th>SIP Server</th>
<th>Bob-1</th>
<th>Bob-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>{A}</td>
<td>------INVITE------&gt;</td>
<td>---INVITE----&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>{A}</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>{null,A}</td>
<td>&lt;--100 Trying---</td>
<td>&lt;180 Ringing-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>{B1,A}</td>
<td>&lt;180 Ringing---</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>{B1,A}</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>{A,B1}</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>{B1,A}</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>{A,B1}</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>{null,A}</td>
<td>&lt;181 Call Fwd--</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>{A}</td>
<td></td>
<td>&lt;-------------INVITE-------&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>{B2,A}</td>
<td></td>
<td>&lt;-------------180 Ringing---</td>
<td></td>
<td></td>
</tr>
<tr>
<td>{B2,A}</td>
<td>&lt;180 Ringing---</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>{B2,A}</td>
<td>&lt;200 OK------</td>
<td>&lt;-------------200 OK ------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>{A,B2}</td>
<td>----ACK--------&gt;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>{A,B2}</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>{null,A}</td>
<td>&lt;----------- Both way RTP Established ==========&gt;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>{A,B2}</td>
<td>-----BYE-------&gt;</td>
<td>&lt;-------------BYE-------&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>{A,B2}</td>
<td>&lt;-------------200 OK------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>{B2,A}</td>
<td>&lt;200 OK------</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 9 - Session ID in the 100 Trying and CANCEL Messaging
Below is the explanatory text from RFC 5359 Section 2.9 detailing what the desired behavior is in the above call flow (i.e., what the call-flow is attempting to achieve).

"Bob wants calls to B1 forwarded to B2 if B1 is not answered (information is known to the SIP server). Alice calls B1 and no one answers. The SIP server then places the call to B2."

General operation of this example:

- Alice generates an INVITE request because she wants to invite Bob to join her session. She creates a UUID as described in section 9.1, and places that value in the "local-uuid" field of the Session-ID header-value.

- The SIP server (imagine this is a B2BUA), upon receiving Alice’s INVITE, and generates the optional provisional response 100 Trying. Since the SIP server has no knowledge Bob’s UUID for his part of the Session ID value, it cannot include his UUID. Rather, the 100 Trying response only includes Alice’s UUID in the "remote-uuid" portion of the Session-ID header-value with a null "local-uuid" value in the response. This is consistent with what Alice’s UA expects to receive in any SIP response containing this UUID.

9.8.2. Session ID in a CANCEL SIP Request

In the same call-flow example as the 100 Trying response is a CANCEL request. Please refer to Figure 9 for the CANCEL request example.

General operation of this example:

- In Figure 9 above, Alice generates an INVITE with her UUID value in the Session-ID header-value.

- Bob-1 responds to this INVITE with a 180 Ringing. In that response, he includes his UUID in the Session-ID header-value; thus completing the Session-ID header-value for this session, even though no final response has been generated by any of Bob’s UAs.

- This means that if the SIP server were to generate a SIP request within this session, in this case a CANCEL request, it would have a complete Session ID to include in that request. In this case, the "local-uuid" = "A", and the "remote-uuid" = "B1".

- As it happens with this CANCEL, the SIP server intends to invite another UA of Bob for Alice to communicate with.

- In this example call-flow, taken from RFC 5359, Section 2.9, a 181 (Call is being Forwarded) response is sent to Alice. Since
the SIP server generated this SIP request, and has no knowledge
of Bob-2’s UUID value, it cannot include that value in this 181.
Thus, and for the exact reasons the 100 Trying including the
Session ID value, only Alice’s UUID is included in the remote-
uuid field of the Session-ID header-value, with a null UUID
present in the "local-uuid" field.

9.9. Session ID in an out-of-dialog REFER Transaction

The following call-flow was extracted from Section 6.1 of [RFC5589]
("Successful Transfer"), with the only changes being the names of the
UAs to maintain consistency within this document.

Alice is the transferee
Bob is the transferer
and Carol is the transfer-target

Session ID  Bob                Alice                Carol

{A}        <------INVITE-------
{B,A}      -----200 OK------>
{A,B}      <------ACK-------->

{B,A}      --INVITE {hold}---->
{A,B}      <-200 OK-------------->
{B,A}      ----- ACK ------------

{X}        --REFER------------- (Target-Dialog:Carol)
{Y,X}      <-202 Accepted------
{Y,X}      <NOTIFY {100 Trying}
{X,Y}      -200 OK-------------

{A}        <--INVITE--------->
{C,A}      <-200 OK------------
{A,C}      ---ACK---------->

{A,B}      <--NOTIFY {200 OK}--
{B,A}      -----200 OK---------
{B,A}      --BYE-------------
{A,B}      <-200 OK------------
{C,A}      <---------BYE------
{A,C}      ------------200 OK->

Figure 10: Basic Transfer Call Flow

General operation of this example:
Just as in Section 9.2, Figure 2, Alice invites Bob to a session, and Bob eventually transfers Alice to communicate with Carol.

What is different about the call-flow in Figure 10 is that Bob’s REFER is not in-dialog, meaning it would have the same UUID pair. Rather, in this case, Bob’s using an out-of-dialog REFER, meaning Bob generates a new UUID for this SIP request, and Alice, subsequently would also generate a new UUID for the 202 (Accepted) response.

Alice will use her existing UUID "A" in the INVITE towards Carol (who generates UUID "C" for this session), thus maintaining the common UUID within the Session ID for this new Alice-to-Carol session.

10. Compatibility with a Previous Implementation

There is a much earlier and proprietary document that specifies the use of a Session-ID header that we will herewith attempt to achieve backwards compatibility. Neither Session-ID has any versioning information, so merely adding that this document describes "version 2" is insufficient. Here are the set of rules for compatibility between the two specifications. For the purposes of this discussion, we will label the proprietary specification of the Session-ID as the "old" version and this specification as the "new" version of the Session-ID.

The previous (i.e., "old") version only has a single value as a Session-ID, but has a generic-parameter value that can be of use.

In order to have an "old" version talk to an "old" version implementation, nothing needs to be done as far as the IETF is concerned.

In order to have a "new" version talk to a "new" version implementation, both implementations need to following this document (to the letter) and everything should be just fine.

In order to have an "old" version talk to a "new" version implementation, several aspects need to be looked at. They are:

- The "old" version UA will include a single UUID as its Session-ID.

- The "new" version UA will respond by including a complete Session-ID with two UUIDs, with the "new" version’s UUID listed first (because it cannot know it is talking with an "old" version implementation at this point).
o The "old" version UA will have to ignore the first UUID, and consider its singular "old" UUID as valid, as long as the value does not change.

o During subsequent transactions within this session, the "new" version may receive SIP requests without its UUID, but with the "old" version’s UUID. The "new" version UA MUST add its UUID to the received Session-ID. The "old" version implementation will merely disregard it each time it receives this "new" version UUID (if it was not the first UUID).

In order to have a "new" version talk to an "old" Version implementation, several aspects need to be looked at. They are:

o The "new" version UA will include a single UUID as its initial Session-ID header always, not knowing which version of UA it is communicating with.

o The "old" version UA will respond by seeing the UUID as a valid and complete Session-ID and not include another UUID or generic-param. Thus, the 200 OK will not include any Session-ID part of its own from the "old" version implementation.

Rule: implementation supporting a "new" version of the Session-ID MUST NOT error or otherwise reject receiving only its own UUID back in any transaction. It MUST interpret this response to mean that it is communicating with an "old" Session-ID implementation.

o Open question - how do we want all intermediaries and/or monitoring systems to interpret this single UUID complete Session-ID?

11. Security Considerations

When creating a UUID value, endpoints SHOULD ensure that there is no user or device-identifying information contained within the UUID. In some environments, though, use of a MAC address, which is one option when constructing a UUID, may be desirable, especially in some enterprise environments. When communicating over the Internet, though, the UUID value MUST utilize random values.

The Session Identifier might be utilized for logging or troubleshooting, but MUST NOT be used for billing purposes.

Other considerations???
12. IANA Considerations

12.1. Registration of the "Session-ID" Header Field

The following is the registration for the 'Session-ID' header field to the "Header Name" registry at http://www.iana.org/assignments/sip-parameters:

RFC number: RFC XXXX

Header name: 'Session-ID'

Compact form: none

[RFC Editor: Please replace XXXX in this section and the next with the this RFC number of this document.]

12.2. Registration of the "remote" Parameter

The following parameter is to be added to the "Header Field Parameters and Parameter Values" section of the SIP parameter registry:

+------------------+----------------+-------------------+-----------+
| Header Field     | Parameter Name | Predefined Values | Reference |
+------------------+----------------+-------------------+-----------+
| Session-ID       | remote         | No                | [RFCXXXX] |
+------------------+----------------+-------------------+-----------+

13. Acknowledgments

The authors would like to thank Robert Sparks, Hadriel Kaplan, Christer Holmberg, Paul Kyzivat, and Charles Eckel for their invaluable comments during the development of this document.

14. References

14.1. Normative References


14.2. Informative References


Author’s Addresses

Paul E. Jones  
Cisco Systems, Inc.  
7025 Kit Creek Rd.  
Research Triangle Park, NC 27709  
USA  
Phone: +1 919 476 2048  
Email: paulej@packetizer.com  
IM: xmpp:paulej@packetizer.com

Chris Pearce  
Cisco Systems, Inc.  
2300 East President George Bush Highway  
Richardson, TX 75082  
USA  
Phone: +1 972 813 5123  
Email: chrep@cisco.com  
IM: xmpp:chrep@cisco.com

James Polk  
Cisco Systems, Inc.  
3913 Treemont Circle  
Colleyville, Texas  
USA  
Phone: +1 817 271 3552  
Email: jmpolk@cisco.com  
IM: xmpp:jmpolk@cisco.com

Gonzalo Salgueiro  
Cisco Systems, Inc.  
7025 Kit Creek Rd.  
Research Triangle Park, NC 27709  
USA  
Phone: +1 919 392 3266  
Email: gsalguei@cisco.com  
IM: xmpp:gsalguei@cisco.com
Abstract

This document describes an end-to-end Session Identifier for use in IP-based multimedia communication systems that enables endpoints, intermediate devices, and management systems to identify a session end-to-end, associate multiple endpoints with a given multipoint conference, track communication sessions when they are redirected, and associate one or more media flows with a given communication session.

This document also describes a backwards compatibility mechanism for an existing (RFC 7329) session identifier implementation that is sufficiently different from the procedures defined in this document.

Status of this Memo

This Internet-Draft is submitted to IETF in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html

This Internet-Draft will expire on September 7, 2015.
Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction...................................................3
2. Conventions used in this document..............................3
3. Session Identifier Requirements and Use Cases..................4
4. Constructing and Conveying the Session Identifier..............4
   4.1. Constructing the Session Identifier.......................4
   4.2. Conveying the Session Identifier..........................5
5. The Session-ID Header Field....................................6
6. Endpoint Behavior..............................................7
7. Processing by Intermediaries...................................8
8. Associating Endpoints in a Multipoint Conference..............11
9. Various Call Flow Operations..................................11
   9.1. Basic Call with 2 UUIDs..................................12
   9.2. Basic Call Transfer using REFER................................15
   9.3. Basic Call Transfer using re-INVITE..........................17
   9.4. Single Focus Conferencing...................................18
   9.5. Single Focus Conferencing using WebEx......................20
   9.6. Cascading Conference Bridges................................21
      9.6.1. Establishing a Cascaded Conference..................21
      9.6.2. Calling into Cascaded Conference Bridges.............23
   9.7. Basic 3PCC for two UAs.....................................24
   9.8. Handling in 100 Trying SIP Response and CANCEL Request...24
      9.8.1. Handling in a 100 Trying SIP Response...............24
      9.8.2. Handling a CANCEL SIP Request........................26
10. Compatibility with a Previous Implementation.................28
11. Security Considerations......................................29
12. IANA Considerations..........................................30
   12.1. Registration of the "Session-ID" Header Field............30
   12.2. Registration of the "remote" Parameter...................30
13. Acknowledgments..............................................30
14. References...................................................30
   14.1. Normative References....................................30
   14.2. Informative References..................................31
Authors’ Addresses...............................................33
1. Introduction

IP-based multimedia communication systems like SIP [RFC3261] and H.323 [H.323] have the concept of a "call identifier" that is globally unique. The identifier is intended to represent an end-to-end communication session from the originating device to the terminating device. Such an identifier is useful for troubleshooting, session tracking, and so forth.

For several reasons, however, the current call identifiers defined in SIP and H.323 are not suitable for end-to-end session identification. A fundamental issue in protocol interworking is the fact that the syntax for the call identifier in SIP and H.323 is different. Thus, if both protocols are used in a call, it is impossible to exchange the call identifier end-to-end.

Another reason why the current call identifiers are not suitable to identify a session end-to-end is that, in real-world deployments, devices like session border controllers often change the session signaling as it passes through the device, including the value of the call identifier. While this is deliberate and useful, it makes it very difficult to track a session end-to-end.

This draft presents a new identifier, referred to as the Session Identifier, and associated syntax intended to overcome the issues that exist with the currently defined call identifiers used in SIP. The procedures specified in this document attempt to comply with the requirements specified in [RFC7206]. The procedures also specify capabilities not mentioned in [RFC7206], shown in call flows in section 9. Additionally, the specification attempts to account for a previous, proprietary version of a SIP Session Identifier header [RFC7329], specifying a backwards compatibility approach in section 10.

2. Conventions used in this document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119] when they appear in ALL CAPS. These words may also appear in this document in lower case as plain English words, absent their normative meanings.

The term "Session Identifier" refers to the value of the identifier, whereas "Session-ID" refers to the header field used to convey the identifier.
3. Session Identifier Requirements and Use Cases

Requirements and use cases for the end-to-end Session Identifier, along with a definition of "session identifier" and "communication session", can be found in [RFC7206].

As mentioned in section 6.1 of RFC 7206, the ITU-T undertook a parallel effort to define compatible procedures for an H.323 Session Identifier. They are documented in [H.460.27].

4. Constructing and Conveying the Session Identifier

4.1. Constructing the Session Identifier

The Session Identifier comprises two UUIDs [RFC4122], with each UUID representing one of the endpoints participating in the session.

The version number in the UUID indicates the manner in which the UUID is generated, such as using random values or using the MAC address of the endpoint. To satisfy the requirement that no user or device information be conveyed, endpoints SHOULD generate version 4 (random) or version 5 (SHA-1) UUIDs to address relevant privacy concerns.

When generating a version 5 UUID, endpoints or intermediaries MUST utilize the procedures defined in Section 4.3 of [RFC4122] and employ the following "name space ID"

```c
uuid_t NameSpace_SessionID = { /* a58587da-c93d-11e2-ae90-f4ea67801e29 */
  0xa58587da, 0xc93d, 0x11e2, 0xae, 0x90, 0xf4, 0xea, 0x67, 0x80, 0x1e, 0x29 }
```

Further, the "name" to utilize for version 5 UUIDs is the concatenation of the Call-ID header value and the "tag" parameter that appears on the "From" or "To" line associated with the device for which the UUID is created. Once an endpoint generates a UUID for a session, the UUID never changes, even if values originally used as input into its construction change over time.

Stateless intermediaries that insert a Session-ID header into a SIP message on behalf of a sending User Agent MUST utilize version 5 UUIDs to ensure that UUIDs for the communication session are consistently generated. If a stateless intermediary does not know the tag value for the endpoint (e.g., a new INVITE without a To: tag value or an older SIP [RFC2543] implementation that did not include a tag parameter), the intermediary MUST NOT attempt to generate a UUID for that endpoint. Note that if an intermediary is stateless and the endpoint on one end of the call is replaced with another endpoint due
to some service interaction, the values used to create the UUID should change and, if so, the intermediary will compute a different UUID.

4.2. Conveying the Session Identifier

The SIP user agent (UA) initially transmitting the SIP request, i.e., a User Agent Client (UAC), will create a UUID and transmit that to the ultimate destination UA. Likewise, the responding UA, i.e., a User Agent Server (UAS), will create a UUID and transmit that to the first UA. These two distinct UUIDs form what is referred to as the Session Identifier and is represented in this document in set notation of the form \(\{A, B\}\), where \(A\) is UUID value from the UA transmitting a message and \(B\) is the UUID value of the intended recipient of the message, i.e., not an intermediary server along the signaling path. The Session Identifier \(\{A, B\}\) is equal to the Session Identifier \(\{B, A\}\).

In the case where only one UUID is known, such as when a UA first initiates a SIP request, the Session Identifier would be \(\{A, N\}\), where "A" represents the UUID value transmitted by the UA and "N" is what is referred to as the null UUID (see section 5).

Since SIP sessions are subject to any number of service interactions, SIP INVITE messages might be forked as sessions are established, and since conferences might be established or expanded with endpoints calling in or the conference focus calling out, the construction of the Session Identifier as a set of UUIDs is important.

To understand this better, consider that a UA participating in a communication session might be replaced with another, such as the case where two "legs" of a call are joined together by a PBX. Suppose that UA A and UA B both call UA C. Further suppose that UA C uses a local PBX function to join the call between itself and UA A with the call between itself and UA B, resulting in a single remaining call between UA A and UA B. This merged call can be identified using two UUID values assigned by each entity in the communication session, namely \(\{A, B\}\) in this example.

In the case of forking, UA A might send an INVITE that gets forked to five different UAs, as an example. A means of identifying each of these separate communication sessions is needed and allowing the set of \(\{A, B_1\}, \{A, B_2\}, \{A, B_3\}, \{A, B_4\}, \text{ and } \{A, B_5\}\) makes this possible.

For conferencing scenarios, it is also useful to have a two-part Session Identifier where the conference focus specifies the same UUID for each conference participant. This will allow for correlation among the participants in a single conference. For example, in a conference with three participants, the Session Identifiers might be...
(A,M), (B,M), and (C,M), where "M" is assigned by the conference focus.

How a device acting on Session Identifiers stores, processes, or utilizes the Session Identifier is outside the scope of this document.

5. The Session-ID Header Field

The syntax specified here replaces the Session-ID header field syntax defined in RFC 7329 [RFC7329].

Each session initiated or accepted MUST have a unique local UA-generated UUID. This value MUST remain unchanged throughout the duration of the session.

A SIP UA or proxy that generates a Session-ID header on behalf of a UA MUST convey its Session Identifier UUID in all transmitted messages by including the Session-ID header. The Session-ID header has the following ABNF [RFC5234] syntax:

```
session-id          = "Session-ID" HCOLON session-id-value
session-id-value    = local-uuid *(SEMI sess-id-param)
local-uuid          = sess-uuid / null
remote-uuid         = sess-uuid / null
sess-uuid           = 32(DIGIT / %x61-66)  ;32 chars of [0-9a-f]
sess-id-param       = remote-param / generic-param
remote-param        = "remote" EQUAL remote-uuid
null                = 32("0")
```

The productions "SEMI", "EQUAL", and "generic-param" are defined in [RFC3261]. The production DIGIT is defined in [RFC5234].

The Session-ID header MUST NOT have more than one "remote" parameter. In the case where an entity compliant with this specification is interworking with an entity that implemented [RFC7329], the "remote" parameter may be absent, but otherwise the remote parameter MUST be present. The details under which those conditions apply are described in Section 10. Except for backwards compatibility with [RFC7329], the "remote" parameter MUST be present.

A special null UUID value composed of 32 zeros is required in certain situations. A null UUID is expected as the "remote-uuid" of every initial standard SIP request since the initiating endpoint would not
initially know the UUID value of the remote endpoint. This null value will get replaced by the ultimate destination UAS when that UA generates a UUID in response. One caveat is explained in Section 10 for a possible backwards compatibility case. A null UUID value is also returned by some intermediary devices that send provisional replies as a "local-uuid", as described in Section 7.

The "local-uuid" in the Session-ID header represents the UUID value of the UA transmitting the message. If the UA transmitting the message previously received one or more UUID values from its peer endpoint in a session, it MUST include the most recently received UUID as the "remote" parameter in each message it transmits within that session. (Exceptions are explicitly called out elsewhere in this document.) For example, a Session-ID header might appear like this:

```
Session-ID: ab30317f1a784dc48ff824d0d3715d86;
            remote=47755a9de7794ba387653f20999600ef2
```

The UUID values are presented as strings of lower-case hexadecimal characters, with the most significant octet of the UUID appearing first.

The Session-ID header-value is technically case-INSENSITIVE, but only lowercase characters are allowed in the sess-uuid components. Receiving entities MUST treat sess-uuid components as case-insensitive and not produce an error if an uppercase hexadecimal character is received in a sess-uuid.

6. Endpoint Behavior

To comply with this specification, endpoint (non-intermediary) SIP UAs MUST include a Session-ID header-value in all SIP messages transmitted as a part of a communication session. The UUID of the transmitter of the message MUST appear in the "local-uuid" portion of the Session-ID header-value. The UUID of the peer device, if known, MUST appear as the "remote" parameter following the transmitter’s UUID. The null UUID value MUST be used if the peer device’s UUID is not known.

Once a UA allocates a UUID value for a communication session, the UA MUST NOT change that UUID value for the duration of the session, including when

- communication attempts are retried due to receipt of 4xx messages or request timeouts;
- the session is redirected in response to a 3xx message; or
- a session is transferred via a REFER message [RFC3515]; or
- a SIP dialog is replaced via an INVITE with Replaces [RFC3891].

A UA that receives a Session-ID header MUST take note of the "local-uuid" value that it receives and assume that is the UUID of the peer endpoint within that communications session. UAs MUST include this received UUID value as the "remote" parameter when transmitting subsequent messages, making sure not to change this UUID value in the process of moving the value internally from the "local-uuid" field to the "remote-uuid" field.

It should be noted that messages received by a UA might contain a "local-uuid" value that does not match what the UA expected the far end UA’s UUID to be. It is also possible for the UA to receive a "remote-uuid" value that does not match the UA’s assigned UUID for the session. Either might happen as a result of service interactions by intermediaries and MUST NOT negatively affect the communication session. However, the UA may log this event for the purposes of troubleshooting.

A UA MUST assume that the UUID value of the peer UA MAY change at any time due to service interactions. If the UUID value of the peer UA changes, the UA MUST accept the new UUID as the peer’s UUID and include this new UUID as the "remote" parameter in any subsequent messages.

It is also important to note that if an intermediary in the network forks a session, the initiating UA may receive multiple responses back from different endpoints, each of which contains a different UUID ("local-uuid") value. UAs MUST take care to ensure that the correct UUID value is returned in the "remote" parameter when interacting with each endpoint. The one exception is when the UA sends a CANCEL message, in which case the Session-ID header value MUST be identical to the Session-ID header value sent in the original INVITE.

If a UA receives a message that does not contain a Session-ID header, that message shall have no effect on what the UA believes is the UUID value of the remote UA.

A Multipoint Control Unit (MCU) is a special type of conferencing endpoint and is discussed in Section 8.

7. Processing by Intermediaries

The following applies only to an intermediary that wishes to comply with this specification and does not impose a conformance requirement on intermediaries that elect not to provide any special treatment for the Session-ID header.

The Call-ID often reveals personal, device, domain or other sensitive information associated with a user, which is why intermediaries, such
as session border controllers, sometimes alter the Call-ID. In order to ensure the integrity of the end-to-end Session Identifier, it is constructed in a way which does not reveal such information, removing the need for intermediaries to alter it.

When an intermediary receives messages from one UA in a communication session that causes the transmission of one or more messages toward the second UA in a communication session, the intermediary MUST include the Session-ID header in the transmitted messages with the same UUID values found in the received message, except as outlined in this section.

Intermediary devices that transfer a call, such as by joining together two different "call legs", MUST properly construct a Session-ID header that contains the correct UUID values and correct placement of those values. As described above, the recipient of any message initiated by the intermediary will assume that the first UUID value belongs to the peer endpoint.

If an intermediary receives a SIP message from a UA without a Session-ID header field or valid header value, the intermediary MAY assign a "local-uuid" value to represent that UA and insert the value into all signaling messages on behalf of the UA for that dialog. If the intermediary is aware of a "remote-uuid" value that identifies the receiving UA, it MUST insert that value if also inserting the "local-uuid" value.

Whenever there is a UA that does not implement this specification communicating through a B2BUA, the B2BUA MAY become dialog stateful and insert a UUID value into the Session-ID header on behalf of the UA according to the rules stated in Section 6.

When an intermediary originates a response, such as a provisional response or a response to a CANCEL request, the "remote-uuid" field will contain the UUID value of the receiving UA. When the UUID of the peer UA is known, the intermediary MUST insert the UUID of the peer UA in the "local-uuid" field of the message. Otherwise, the intermediary MAY set the "local-uuid" field of the message to the "null" UUID value.

When an intermediary originates a request message without first having received a SIP message that triggered the transmission of the message (e.g., sending a BYE message to terminate a call for policy reasons), the intermediary MUST, if it has knowledge of the UUID values for the two communicating UAs, insert a Session-ID header value with the "remote-uuid" field set to the UUID value of the receiving UA and the "local-uuid" field set to the UUID value of the other UA. When the intermediary does not have knowledge of the UUID value of a UA in the communication session, the intermediary SHOULD set the UUID value to the "null" UUID value.
With respect to the previous two paragraphs, note that if an intermediary transmits a "null" UUID value, the receiving UA might use that value in subsequent messages it sends. This effectively violates the requirement of maintaining an end-to-end Session Identifier value for the communication session if a UUID for the peer UA had been previously conveyed. Therefore, intermediaries MUST only send the "null" UUID when the intermediary has not communicated with the peer UA to learn its UUID. This means that intermediaries SHOULD maintain state related to the UUID values for both ends of a communication session. An intermediary that does not maintain this state and that originates a message as described in the previous two paragraph MUST NOT insert a Session-ID header in order to avoid unintended, incorrect reassignment of a UUID value.

The Session-ID included in a CANCEL request MUST be identical to the Session-ID included in the corresponding INVITE.

Devices that initiate communication sessions following the procedures for third party call control MUST fabricate a UUID value that will be utilized only temporarily. Once the responding endpoint provides a UUID value in a response message, the temporary value MUST be discarded and replaced with the endpoint-provided UUID value. Refer to the third-party call control example for an illustration. If the third-party call controller sends a re-INVITE to obtain an offer for connecting the endpoint to a different session, the Session-ID MAY reflect the current session; the ACK’s Session-ID would reflect the newly connected session.

If a SIP intermediary initiates a dialog between two UAs in a 3PCC [RFC3725] scenario, the SIP request in the initial INVITE will have a non-null "local-uuid" value; call this temporary UUID X. The request will still have a null "remote-uuid" value; call this value N. The SIP server MUST be transaction stateful. The UUID pair in the INVITE will be \( \{X, N\} \). A non-redirected or rejected response will have a UUID pair \( \{A, X\} \). This transaction stateful, dialog initiating SIP server MUST replace its own UUID, i.e., X, with a null UUID (i.e., \( \{A, N\} \)) as expected by other UAS (see Section 9.7 for an example).

Having said the foregoing, intermediaries that manipulate messages containing Session-ID header values SHOULD be aware of what values were last received by an endpoint and, following any kind of service interaction initiated or affected by the intermediary, of what values the receiving endpoint should have knowledge to ensure that both endpoints in the session have the correct and same values. If an intermediary can determine that an endpoint might not have received a current, correct Session-ID header value, the Intermediary SHOULD attempt to provide the correct Session-ID header value to the endpoint such as by sending a re-INVITE message.
8. Associating Endpoints in a Multipoint Conference

Multipoint Control Units (MCUs) group two or more sessions into a single multipoint conference. MCUs, including cascaded MCUs, MUST utilize the same UUID value ("local-uuid" portion of the Session-ID header-value) with all participants in the conference. In so doing, each individual session in the conference will have a unique Session Identifier (since each endpoint will create a unique UUID of its own), but will also have one UUID in common with all other participants in the conference.

When creating a cascaded conference, an MCU MUST convey the UUID value to utilize for a conference via the "local-uuid" portion of the Session-ID header-value in an INVITE to a second MCU when using SIP to establish the cascaded conference. A conference bridge, or MCU, needs a way to identify itself when contacting another MCU. RFC 4579 [RFC4579] defines the "isfocus" Contact: header parameter just for this purpose. The initial MCU MUST include the UUID of that particular conference in the "local-uuid" of an INVITE to the other MCU(s) participating in that conference. Also included in this INVITE is an "isfocus" Contact header parameter identifying that this INVITE is coming from an MCU and that this UUID is to be given out in all responses from UAs into those MCUs participating in this same conference. This ensures a single UUID is common across all participating MCUs of the same conference, but is unique between different conferences.

Intermediary devices or network diagnostics equipment might assume that when they see two or more sessions with different Session Identifiers, but with one UUID in common, that the sessions are part of the same conference. However, the assumption that two sessions having one common UUID being part of the same conference is not always correct. In a SIP forking scenario, for example, there might also be what appears to be multiple sessions with a shared UUID value; this is intended. The desire is to allow for the association of related sessions, regardless of whether a session is forked or part of a conference.

9. Various Call Flow Operations

Seeing something frequently makes understanding easier. With that in mind, we include several call flow examples with the initial UUID and the complete Session Identifier indicated per message, as well as when the Session Identifier changes according to the rules within this document during certain operations/functions.

This section is for illustrative purposes only and is non-normative. In the following flows, RTP refers to the Real-time Transport Protocol [RFC3550].
In the examples in this section, "N" represents a null UUID and other letters represents the unique UUID values corresponding to endpoints or MCUs.

9.1. Basic Call with 2 UUIDs

<table>
<thead>
<tr>
<th>Session-ID</th>
<th>--- INVITE F1----&gt;</th>
<th>--- INVITE F2----&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>(A,N)</td>
<td>--- INVITE F1----&gt;</td>
<td>--- INVITE F2----&gt;</td>
</tr>
<tr>
<td>(B,A)</td>
<td>&lt;---200 OK F4-----&gt;</td>
<td>-----ACK F5-----&gt;</td>
</tr>
<tr>
<td>(A,B)</td>
<td>-----ACK F6-----&gt;</td>
<td>&lt;==============RTP==============&gt;</td>
</tr>
<tr>
<td>(A,B)</td>
<td>-----ACK F6-----&gt;</td>
<td>&lt;==============RTP==============&gt;</td>
</tr>
</tbody>
</table>

Figure 1 - Session-ID Creation when Alice calls Bob

General operation of this example:

- UA-Alice populates the "local-uuid" portion of the Session-ID header-value.

- UA-Alice sends its UUID in the SIP INVITE, and populates the "remote" parameter with a null value (32 zeros).

- B2BUA receives an INVITE with both a "local-uuid" portion of the Session-ID header-value from UA-Alice as well as the null "remote-uuid" value, and transmits the INVITE towards UA-Bob with an unchanged Session-ID header-value.

- UA-Bob receives Session-ID and generates its "local-uuid" portion of the Session-ID header-value UUID to construct the whole/complete Session-ID header-value, at the same time transferring Alice's UUID unchanged to the "remote-uuid" portion of the Session-ID header-value in the 200 OK SIP response.

- B2BUA receives the 200 OK response with a complete Session-ID header-value from UA-Bob, and transmits 200 OK towards UA-Alice with an unchanged Session-ID header-value.

- UA-Alice, upon reception of the 200 OK from the B2BUA, transmits the ACK towards the B2BUA. The construction of the Session-ID header-value in this ACK is that of Alice’s UUID is the "local-uuid", and Bob’s UUID populates the "remote-uuid" portion of the header-value.

- B2BUA receives the ACK with a complete Session-ID header-value from UA-Alice, and transmits ACK towards UA-Bob with an unchanged Session-ID header-value.
Below is a complete SIP message exchange illustrating proper use of the Session-ID header. For the sake of brevity, non-essential headers and message bodies are omitted.

F1 INVITE Alice -> B2BUA

INVITE sip:alice@biloxi.com SIP/2.0
Via: SIP/2.0/UDP pc33.atlanta.example.com
;branch=z9hG4bK776asdhds
Max-Forwards: 70
To: Alice <sip:alice@atlanta.example.com>;tag=1928301774
From: Alice <sip:alice@atlanta.example.com>;tag=1928301774
Call-ID: a84b4c76e66710@pc33.atlanta.example.com
Session-ID: ab30317f1a784dc48ff824d0d3715d86
;remote=00000000000000000000000000000000
CSeq: 314159 INVITE
Contact: <sip:alice@pc33.atlanta.example.com>
Content-Type: application/sdp
Content-Length: 142

(Alice’s SDP not shown)

F2 INVITE B2BUA -> Bob

INVITE sip:alice@biloxi.com SIP/2.0
Via: SIP/2.0/UDP server10.biloxi.example.com
;branch=z9hG4bK4b43c2ff8.1
Via: SIP/2.0/UDP pc33.atlanta.example.com
;branch=z9hG4bK776asdhds;received=10.1.3.33
Max-Forwards: 69
To: Bob <sip:alice@biloxi.example.com>
From: Alice <sip:alice@atlanta.example.com>;tag=1928301774
Call-ID: a84b4c76e66710@pc33.atlanta.example.com
Session-ID: ab30317f1a784dc48ff824d0d3715d86
;remote=00000000000000000000000000000000
CSeq: 314159 INVITE
Contact: <sip:alice@pc33.atlanta.example.com>
Record-Route: <sip:server10.biloxi.example.com;lr>
Content-Type: application/sdp
Content-Length: 142

(Alice’s SDP not shown)
F3 200 OK Bob -> B2BUA

SIP/2.0 200 OK
Via: SIP/2.0/UDP server10.biloxi.example.com
   ;branch=z9hG4bK4b43c2ff8.1;received=192.168.10.1
Via: SIP/2.0/UDP pc33.atlanta.example.com
   ;branch=z9hG4bK776asdhds;received=10.1.3.33
To: Bob <sip:bob@biloxi.example.com>;tag=a6c85cf
From: Alice <sip:alice@atlanta.example.com>;tag=1928301774
Call-ID: a84b4c76e66710@pc33.atlanta.example.com
Session-ID: 47755a9de7794ba387653f2099600ef2
   ;remote=ab30317f1a784dc48ff824d0d3715d86
CSeq: 314159 INVITE
Contact: <sip:bob@192.168.10.20>
Record-Route: <sip:server10.biloxi.example.com;lr>
Content-Type: application/sdp
Content-Length: 131

(Bob’s SDP not shown)

F4 200 OK B2BUA -> Alice

SIP/2.0 200 OK
Via: SIP/2.0/UDP pc33.atlanta.example.com
   ;branch=z9hG4bK776asdhds;received=10.1.3.33
To: Bob <sip:bob@biloxi.example.com>;tag=a6c85cf
From: Alice <sip:alice@atlanta.example.com>;tag=1928301774
Call-ID: a84b4c76e66710@pc33.atlanta.example.com
Session-ID: 47755a9de7794ba387653f2099600ef2
   ;remote=ab30317f1a784dc48ff824d0d3715d86
CSeq: 314159 INVITE
Contact: <sip:bob@192.168.10.20>
Record-Route: <sip:server10.biloxi.example.com;lr>
Content-Type: application/sdp
Content-Length: 131

(Bob’s SDP not shown)

F5 ACK Alice -> B2BUA
The remaining examples in this Section do not display the complete SIP message exchange. Instead, they simply use the set notation described in Section 4.2 to show the Session Identifier exchange throughout the particular call flow being illustrated.

9.2. Basic Call Transfer using REFER

From the example built within Section 9.1, we proceed to this ‘Basic Call Transfer using REFER’ example. Note that this is a mid-dialog REFER in contrast with the out-of-dialog REFER in Section 9.9.

\[
\begin{array}{cccc}
\text{Session-ID} & \text{Alice} & \text{B2BUA} & \text{Bob} & \text{Carol} \\
--- & | & | & | \\
\{B,A\} & \text{<----------------RTP---------------->} & \text{<---re-INVITE--->} & \\
\end{array}
\]
Starting from the existing Alice/Bob call described in Figure 1 of this document, which established an existing Session-ID header-value...

- UA-Bob requests Alice to call Carol, using a REFER transaction, as described in [RFC3515]. UA-Alice is initially put on hold, then told in the REFER who to contact with a new INVITE, in this case UA-Carol. This Alice-to-Carol dialog will have a new Call-ID, therefore it requires a new Session-ID header-value. The wrinkle here is we can, and will, use Alice’s UUID from her existing dialog with Bob in the new INVITE to Carol.
o UA-Alice retains her UUID from the Alice-to-Bob call \{A\} when requesting a call with UA-Carol. This is placed in the "local-uuid" portion of the Session-ID header-value, at the same time inserting a null "remote-uuid" value (because Carol's UA has not yet received the UUID value). This same UUID traverses the B2BUA unchanged.

o UA-Carol receives the INVITE with a Session Identifier UUID \{A,N\}, replaces the A UUID value into the "remote-uuid" portion of the Session-ID header-value and creates its own UUID \{C\} and places this value in the "local-uuid" portion of the Session-ID header-value - thereby removing the N (null) value altogether. This combination forms a full Session Identifier \{C,A\} in the 200 OK to the INVITE. This Session-ID header-value traverses the B2BUA unchanged towards UA-Alice.

o UA-Alice receives the 200 OK with the Session Identifier \{C,A\} and responds to UA-Carol with an ACK (just as in Figure 1 - switches places of the two UUID fields), and generates a NOTIFY to Bob with a Session Identifier \{A,B\} indicating the call transfer was successful.

o It does not matter which UA terminates the Alice-to-Bob call; Figure 2 shows UA-Bob doing this transaction.

9.3. Basic Call Transfer using re-INVITE

From the example built within Section 9.1, we proceed to this 'Basic Call Transfer using re-INVITE' example.

Alice is talking to Bob. Bob pushes a button on his phone to transfer Alice to Carol via the B2BUA (using re-INVITE).

\[
\begin{array}{|c|c|c|c|}
\hline
\text{Session-ID} & \text{Alice} & \text{B2BUA} & \text{Bob} & \text{Carol} \\
\hline
& \text{<-------------------RTP------------------->} & & & \\
\hline
\{A,B\} & \text{<--- (non-standard signaling)} & \text{---re-INVITE---} & & \\
\hline
\{B,A\} & \text{<-----200 OK-----} & \text{-----ACK------} & & \\
\hline
\{A,N\} & \text{-----INVITE---------------------->} & & & \\
\hline
\{C,A\} & \text{<-----200 OK---------------------->} & \text{-----ACK---------------------->} & & \\
\hline
\{A,C\} & \text{<-------------------RTP------------------->} & & & \\
\hline
\{A,B\} & \text{-----BYE------} & \text{<-----200 OK------} & & \\
\hline
\{B,A\} & & \text{<-----200 OK------} & & \\
\hline
\end{array}
\]
General operation of this example:

- We assume the call between Alice and Bob from Section 9.1 is operational with Session Identifier \{A,B\}.

- Bob uses non-standard signaling to the B2BUA to initiate a call transfer from Alice to Carol. This could also be initiated via a REFER message from Bob, but the signaling that follows might still be similar to the above flow. In either case, Alice is completely unaware of the call transfer until a future point in time when Alice receives a message from Carol.

- The B2BUA sends a new INVITE with Alice’s UUID \"local-uuid\" = "A\" to Carol.

- Carol receives the INVITE and accepts the request and adds her UUID \{C\} to the Session Identifier for this session \"local-uuid\" = "C", \"remote-uuid\" = "A\".

- The B2BUA then terminates the call to Bob with a BYE using the Session Identifier \"local-uuid\" = "A", \"remote-uuid\" = "B\".

- Since Alice never received Carol’s UUID from the B2BUA, when Alice later attempts to modify the session with a re-INVITE, Alice would send the \"remote-uuid\" = "B\" toward Carol. Carol replies with the \"local-uuid\" = "A", \"remote-uuid\" = "A\" to reflect what was received in the INVITE (which Carol already knew from previous exchanges with the B2BUA). Alice then include \"remote-uuid\" = "C\" in the following ACK message.

9.4. Single Focus Conferencing

Multiple users call into a conference server (say, an MCU) to attend one of many conferences hosted on or managed by that server. Each user has to identify which conference they want to join, but this information is not necessarily in the SIP messaging. It might be done by having a dedicated address for the conference or via an IVR, as assumed in this example and depicted with the use of M1, M2, and M3. Each user in this example goes through a two-step process of
signaling to gain entry onto their conference call, which the conference focus identifies as M’.

### Figure 4 - Single Focus Conference Bridge

<table>
<thead>
<tr>
<th>Session-ID</th>
<th>Conference</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Alice</td>
</tr>
<tr>
<td>{A,N}</td>
<td>----INVITE----&gt;</td>
</tr>
<tr>
<td>{M1,A}</td>
<td>&lt;---200 OK------</td>
</tr>
<tr>
<td>{A,M1}</td>
<td>-----ACK-------&gt;</td>
</tr>
<tr>
<td>{M',A}</td>
<td>&lt;---re-INVITE---</td>
</tr>
<tr>
<td>{A,M'}</td>
<td>-----200 OK------&gt;</td>
</tr>
<tr>
<td>{M',A}</td>
<td>&lt;-----ACK-------</td>
</tr>
<tr>
<td>{B,N}</td>
<td>&lt;-----INVITE-----</td>
</tr>
<tr>
<td>{M2,B}</td>
<td>-----200 OK------&gt;</td>
</tr>
<tr>
<td>{B,M2}</td>
<td>-----ACK-------&gt;</td>
</tr>
<tr>
<td>{M',B}</td>
<td>&lt;---re-INVITE---</td>
</tr>
<tr>
<td>{B,M'}</td>
<td>-----200 OK------&gt;</td>
</tr>
<tr>
<td>{M',B}</td>
<td>&lt;-----ACK-------</td>
</tr>
<tr>
<td>{C,N}</td>
<td>&lt;---------------------INVITE-----</td>
</tr>
<tr>
<td>{M3,C}</td>
<td>---------------------200 OK------&gt;</td>
</tr>
<tr>
<td>{C,M3}</td>
<td>&lt;---------------------ACK-------</td>
</tr>
<tr>
<td>{M',C}</td>
<td>&lt;---------------------re-INVITE--&gt;</td>
</tr>
<tr>
<td>{C,M'}</td>
<td>&lt;---------------------200 OK------&gt;</td>
</tr>
<tr>
<td>{M',C}</td>
<td>&lt;---------------------ACK-------</td>
</tr>
</tbody>
</table>

General operation of this example:

Alice calls into a conference server to attend a certain conference. This is a two-step operation since Alice cannot include the conference ID at this time and/or any passcode in the INVITE request. The first step is Alice’s UA calling another UA to participate in a session. This will appear to be similar as the call-flow in Figure 1 (in section 9.1). What is unique about this call is the second step: the conference server sends a re-INVITE request with its second UUID, but maintaining the UUID Alice sent in the first INVITE. This subsequent UUID from the conference server will be the same for each UA that calls into this conference server participating in this same conference bridge/call, which is generated once Alice typically authenticates and identifies which bridge she wants to participate on.
o Alice sends an INVITE to the conference server with her UUID {A} and a "remote-uuid" = N.

o The conference server responds with a 200 OK response which replaces the N UUID with a temporary UUID ("M1") as the "local-uuid" and a "remote-uuid" = "A".

NOTE: this 'temporary' UUID is a real UUID; it is only temporary to the conference server because it knows that it is going to generate another UUID to replace the one just send in the 200 OK.

o Once Alice, the user, gains access to the IVR for this conference server, she enters a specific conference ID and whatever passcode (if needed) to enter a specific conference call.

o Once the conference server is satisfied Alice has identified which conference she wants to attend (including any passcode verification), the conference server re-INVITEs Alice to the specific conference and includes the Session-ID header-value of "local-uuid" = "M'" (and "remote-uuid" = "A") for that conference. All valid participants in the same conference will receive this same UUID for identification purposes and to better enable monitoring, and tracking functions.

o Bob goes through this two-step process of an INVITE transaction, followed by a re-INVITE transaction to get this same UUID ("M'") for that conference.

o In this example, Carol (and each additional user) goes through the same procedures and steps as Alice and Bob to get on this same conference.

9.5. Single Focus Conferencing using WebEx

Alice, Bob and Carol call into same WebEx conference.

<table>
<thead>
<tr>
<th>Session-ID</th>
<th>Alice</th>
<th>Conference</th>
<th>Bob</th>
<th>Carol</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Focus</td>
<td></td>
<td></td>
</tr>
<tr>
<td>{M,N}</td>
<td></td>
<td>&lt;----INVITE-----</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(A,M)</td>
<td></td>
<td>-----200 OK-----</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(M,A)</td>
<td></td>
<td>&lt;-----ACK-------</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;-----RTP-------</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;---HTTPS-----**&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Transaction</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 5 - Single Focus WebEx Conference

General operation of this example:

- Alice communicates with WebEx server with desire to join a certain meeting, by meeting number; also includes UA-Alice’s contact information (phone number, URI and/or IP address, etc.) for each device she wants for this conference call. For example, the audio and video play-out devices could be separate units.

- Conference Focus server sends INVITE (Session-ID header-value "local-uuid" = M and a remote UUID of N, where M equals the "local-uuid" for each participant on this conference bridge) to UA-Alice to start session with that server for this A/V conference call.

- Upon receiving the INVITE request from the conference focus server, Alice responds with a 200 OK. Her UA moves the "local-uuid" unchanged into the "remote-uuid" field, and generates her own UUID and places that into the "local-uuid" field to complete the Session-ID construction.

- Bob and Carol perform same function to join this same A/V conference call as Alice.

9.6. Cascading Conference Bridges

9.6.1. Establishing a Cascaded Conference

To expand conferencing capabilities requires cascading conference bridges. A conference bridge, or MCU, needs a way to identify itself when contacting another MCU. RFC 4579 [RFC4579] defines the 'isfocus' Contact: header parameter just for this purpose.
Regardless of which MCU (1 or 2) a UA contacts for this conference, once the above exchange has been received and acknowledged, the UA will get the same \( \{M',N\} \) UUID pair from the MCU for the complete Session Identifier.

A more complex form would be a series of MCUs all being informed of the same UUID to use for a specific conference. This series of MCUs can either be informed:

- All by one MCU (that initially generates the UUID for the conference).
- The MCU that generates the UUID informs one or several MCUs of this common UUID, and they inform downstream MCUs of this common UUID that each will be using for this one conference.

**Figure 6 - MCUs Communicating Session Identifier UUID for Bridge**

<table>
<thead>
<tr>
<th>Session-ID</th>
<th>---</th>
<th>MCU-1</th>
<th>MCU-2</th>
<th>MCU-3</th>
<th>MCU-4</th>
</tr>
</thead>
<tbody>
<tr>
<td>( {M',N} )</td>
<td>----INVITE-------&gt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( {J,M'} )</td>
<td>----ACK-------&gt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( {M',J} )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 7 - MCU Communicating Session Identifier UUID to More than One MCU**

General operation of this example:

- The MCU generating the Session Identifier UUID communicates this in a separate INVITE, having a Contact header with the ‘isfocus’ header parameter. This will identify the MCU as what RFC 4579 calls a conference-aware SIP entity.
- An MCU that receives this \( \{M',N\} \) UUID pair in an inter-MCU transaction can communicate the M’ UUID in a manner in which it was received to construct a hierarchical cascade (though this time this second MCU would be the UAC MCU).
Once the conference is terminated, the cascaded MCUs will receive a BYE message to terminate the cascade.

9.6.2. Calling into Cascaded Conference Bridges

Here is an example of how a UA, say Robert, calls into a cascaded conference focus. Because MCU-1 has already contacted MCU-3, the MCU where Robert is going to join the conference, MCU-3 already has the Session-ID (M’) for this particular conference call.

<table>
<thead>
<tr>
<th>Session-ID</th>
<th>MCU-1</th>
<th>MCU-2</th>
<th>MCU-3</th>
<th>Robert</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>{M’,N}</td>
<td>-------INVITE------&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>{J,M’}</td>
<td>&lt;---200 OK------&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>{M’,J}</td>
<td>-----ACK-------&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>{M’,N}</td>
<td>------------------------INVITE--------&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>{K,M’}</td>
<td>---------------200 OK------&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>{M’,K}</td>
<td>--------------------ACK-------&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>{R,N}</td>
<td></td>
<td>&lt;---INVITE-----</td>
<td></td>
</tr>
<tr>
<td></td>
<td>{M’,R}</td>
<td></td>
<td>----200 OK----&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>{R,M’}</td>
<td></td>
<td>&lt;----ACK-------</td>
<td></td>
</tr>
</tbody>
</table>

Figure 8 - A UA Calling into a Cascaded MCU UUID

General operation of this example:

- The UA, Robert in this case, INVITEs the MCU to join a particular conference call. Robert’s UA does not know anything about whether this is the main MCU of the conference call, or a cascaded MCU. Robert likely does not know MCUs can be cascaded, he just wants to join a particular call. Like as with any standard implementation, he includes a null "remote-uuid".

- The cascaded MCU, upon receiving this INVITE from Robert, replaces the null UUID with the UUID value communicated from MCU-1 for this conference call as the "local-uuid" in the SIP response. Thus, moving Robert’s UUID "R" to the "remote-uuid" value.

- The ACK has the Session-ID {R,M’}, completing the 3-way handshake for this call establishment. Robert has now joined the conference call originated from MCU-1.

- Once the conference is terminated, the cascaded MCUs will receive a BYE message to terminate the cascade.
9.7. Basic 3PCC for two UAs

External entity sets up call to both Alice and Bob for them to talk to each other.

<table>
<thead>
<tr>
<th>Session-ID</th>
<th>Alice</th>
<th>B2BUA</th>
<th>Bob</th>
<th>Carol</th>
</tr>
</thead>
<tbody>
<tr>
<td>{X,N}</td>
<td>--------</td>
<td>&lt;-----INVITE------</td>
<td>---------</td>
<td>-------</td>
</tr>
<tr>
<td>{A,X}</td>
<td>--------</td>
<td>-----200 OK--------</td>
<td>--------</td>
<td>-------</td>
</tr>
<tr>
<td>{A,N}</td>
<td>--------</td>
<td>------INVITE------&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>{B,A}</td>
<td>--------</td>
<td>&lt;-----ACK--------&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>{B,A}</td>
<td>--------</td>
<td>------200 OK--------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>{A,B}</td>
<td>&lt;---------RTP----------</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 9 - 3PCC initiated call between Alice and Bob

General operation of this example:

- Some out of band procedure directs a B2BUA (or other SIP server) to have Alice and Bob talk to each other. In this case, the SIP server MUST be transaction stateful, if not dialog stateful.
- The SIP server INVITEs Alice to a session and uses a temporary UUID (X) and a null UUID pairing.
- Alice receives and accepts this call set-up and replaces the null UUID with her UUID (A) in the Session Identifier, now {A,X}.
- The transaction stateful SIP server receives Alice’s UUID (A) in the local UUID portion and keeps it there, and discards its own UUID (X), replacing this with a null UUID value in the INVITE to Bob as if this came from Alice originally.
- Bob receives and accepts this INVITE and adds his own UUID (B) to the Session Identifier, now (B,A) for the response.
- And the session is established.

9.8. Handling in 100 Trying SIP Response and CANCEL Request

The following two subsections show examples of the Session Identifier for a 100 Trying response and a CANCEL request in a single call-flow.

9.8.1. Handling in a 100 Trying SIP Response

The following 100 Trying response is taken from an existing RFC, from [RFC5359] Section 2.9 ("Call Forwarding - No Answer").
Figure 10 - Session Identifier in the 100 Trying and CANCEL Messaging

Below is the explanatory text from RFC 5359 Section 2.9 detailing what the desired behavior is in the above call flow (i.e., what the call-flow is attempting to achieve).

"Bob wants calls to B1 forwarded to B2 if B1 is not answered (information is known to the SIP server). Alice calls B1 and no one answers. The SIP server then places the call to B2."

General operation of this example:

- Alice generates an INVITE request because she wants to invite Bob to join her session. She creates a UUID as described in section 9.1, and places that value in the "local-uuid" field of the Session-ID header-value. Alice also generates a "remote-uuid" of null and sends this along with the "local-uuid".

The SIP server (imagine this is a B2BUA), upon receiving Alice’s INVITE, generates the optional provisional response 100 Trying. Since the SIP server has no knowledge Bob’s UUID for his part of the Session Identifier value, it cannot include his "local-uuid". Rather, any 100 Trying response includes Alice’s UUID in the "remote-uuid" portion of the Session-ID header-value with a null "local-uuid" value in the response. This is consistent with what Alice’s UA expects to receive in any SIP response containing this UUID.

9.8.2. Handling a CANCEL SIP Request

In the same call-flow example as the 100 Trying response is a CANCEL request. Please refer to Figure 10 for the CANCEL request example.

General operation of this example:

- In Figure 10 above, Alice generates an INVITE with her UUID value in the Session-ID header-value.

- Bob-1 responds to this INVITE with a 180 Ringing. In that response, he includes his UUID in the Session-ID header-value (i.e., \{B1,A\}); thus completing the Session-ID header-value for this session, even though no final response has been generated by any of Bob’s UAs.

- While this means that if the SIP server were to generate a SIP request within this session it could include the complete SessionID, the server sends a CANCEL and a CANCEL MUST always use the same Session-ID header value as the original INVITE. Thus, the CANCEL would have a Session Identifier with the "local-uuid" = "A", and the "remote-uuid" = "N".

- As it happens with this CANCEL, the SIP server intends to invite another UA of Bob (i.e., B2) for Alice to communicate with.

- In this example call-flow, taken from RFC 5359, Section 2.9, a 181 (Call is being Forwarded) response is sent to Alice. Since the SIP server generated this SIP request, and has no knowledge of Bob-2’s UUID value, it cannot include that value in this 181. Thus, and for the exact reasons the 100 Trying including the Session Identifier value, only Alice’s UUID is included in the remote-uuid field of the Session-ID header-value, with a null UUID present in the "local-uuid" field.

9.9. Out-of-dialog REFER Transaction

The following call-flow was extracted from Section 6.1 of [RFC5589] ("Successful Transfer"), with the only changes being the names of the UAs to maintain consistency within this document.
Alice is the transeree
Bob is the transferer
and Carol is the transfer-target

<table>
<thead>
<tr>
<th>Session-ID</th>
<th>Bob</th>
<th>Alice</th>
<th>Carol</th>
</tr>
</thead>
<tbody>
<tr>
<td>{A,N}</td>
<td>&lt;-----INVITE--</td>
<td>--</td>
<td></td>
</tr>
<tr>
<td>{B,A}</td>
<td>-----200 OK--</td>
<td></td>
<td></td>
</tr>
<tr>
<td>{A,B}</td>
<td>&lt;------ACK----</td>
<td>--</td>
<td></td>
</tr>
<tr>
<td>{B,A}</td>
<td>--INVITE {hold}--</td>
<td>--</td>
<td>(Refer-To:Carol)</td>
</tr>
<tr>
<td>{A,B}</td>
<td>&lt;---200 OK------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>{A,B}</td>
<td>--- ACK--------</td>
<td>--</td>
<td></td>
</tr>
<tr>
<td>{B,A}</td>
<td>--REFER---------</td>
<td>--</td>
<td></td>
</tr>
<tr>
<td>{A,B}</td>
<td>&lt;---202 Accepted--</td>
<td>--</td>
<td></td>
</tr>
<tr>
<td>{A,B}</td>
<td>&lt;NOTIFY {100 Trying}</td>
<td>--</td>
<td></td>
</tr>
<tr>
<td>{B,A}</td>
<td>-200 OK---------</td>
<td>--</td>
<td></td>
</tr>
<tr>
<td>{A,N}</td>
<td>--INVITE--------</td>
<td>--</td>
<td></td>
</tr>
<tr>
<td>{C,A}</td>
<td>&lt;---200 OK------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>{A,C}</td>
<td>---ACK----------</td>
<td>--</td>
<td></td>
</tr>
<tr>
<td>{A,B}</td>
<td>&lt;---NOTIFY {200 OK}--</td>
<td>--</td>
<td></td>
</tr>
<tr>
<td>{B,A}</td>
<td>---200 OK--------</td>
<td>--</td>
<td></td>
</tr>
<tr>
<td>{B,A}</td>
<td>--BYE----------</td>
<td>--</td>
<td></td>
</tr>
<tr>
<td>{A,B}</td>
<td>&lt;---200 OK------</td>
<td>--</td>
<td></td>
</tr>
<tr>
<td>{C,A}</td>
<td>&lt;------------BYE-----</td>
<td>--</td>
<td></td>
</tr>
<tr>
<td>{A,C}</td>
<td>---------------200 OK-&gt;</td>
<td>--</td>
<td></td>
</tr>
</tbody>
</table>

Figure 11: Out-Of-Diallog Call Transfer

General operation of this example:

- Just as in Section 9.2, Figure 2, Alice invites Bob to a session, and Bob eventually transfers Alice to communicate with Carol.

- What is different about the call-flow in Figure 11 is that Bob’s REFER is not in-dialog. Even so, this is treated as part of the same communication session and, thus, the Session Identifier in those messages is {A,B}.

- Alice will use her existing UUID and the null UUID ((A,N)) in the INVITE towards Carol (who generates UUID "C" for this session), thus maintaining the common UUID within the Session Identifier for this new Alice-to-Carol session.
10. Compatibility with a Previous Implementation

There is a much earlier and proprietary document that specifies the use of a Session-ID header (namely, [RFC7329]) that we will herewith attempt to achieve backwards compatibility. Neither Session-ID header has any versioning information, so merely adding that this document describes "version 2" is insufficient. Here are the set of rules for compatibility between the two specifications. For the purposes of this discussion, we will label the proprietary specification of the Session-ID as the "old" version and this specification as the "new" version of the Session-ID.

The previous (i.e., "old") version only has a single value as a Session-ID, but has a generic-parameter value that can be of use.

In order to have an "old" version talk to an "old" version implementation, nothing needs to be done as far as the IETF is concerned.

In order to have a "new" version talk to a "new" version implementation, both implementations need to follow this document (to the letter) and everything should be just fine.

But that is where compatibility is not ensured, given the unknowns related to the behavior of entities implementing the pre-standard implementation. For this "new" implementation to work with the "old" implementation and an "old" implementation to work with "new" implementations, there needs to be a set of rules that all "new" implementations MUST follow.

- Since no option tags or feature tags are to be used for distinguishing versions, the presence and order of any "remote-uuid" value within the Session-ID header value is to be used to distinguish implementation versions.

- If a SIP request has a "remote-uuid" value, this comes from a standard implementation, and not a pre-standard one.

- If a SIP request has no "remote-uuid" value, this comes from a pre-standard implementation, and not a standard one. In this case, one UUID is used to identify this dialog, even if the responder is a standard implementation of this specification.

- If a SIP response has a non-null "local-uuid" that is 32 octets long and differs from the endpoint’s own UUID value, this response comes from a standard implementation.

- If a SIP response has a non-null "local-uuid" that is not 32 octets long, this response comes from a misbehaving implementation, and its Session-ID header value MUST be discarded. That said, the
response might still be valid according to the rules within SIP [RFC3261], and SHOULD be checked further.

- If a SIP response arrives that has the same value of Session-ID UUIDs in the same order as was sent, this comes from a pre-standard implementation, and MUST NOT be discarded for not altering the null "remote-uuid". In this case, any new transaction within this dialog MUST preserve the order of the two UUIDs within all Session-ID header-values, including the ACK, until this dialog is terminated.

- If a SIP response only contains the "local-uuid" that was sent originally, this comes from a pre-standard implementation and MUST NOT be discarded for removing the null "remote-uuid". In this case, all future transactions within this dialog MUST contain only the UUID received in the first SIP response. Any new transaction starting a new dialog from the standard Session-ID implementation MUST include a "local-uuid" and a null "remote-uuid", even if that new dialog is between the same two UAs.

- Standard implementations SHOULD NOT expect pre-standard implementations to be consistent in their implementation, even within the same dialog. For example, perhaps the first, third and tenth responses contain a "remote-uuid", but all the others do not. This behavior MUST be allowed by implementations of this specification.

- The foregoing does not apply to other, presently unknown parameters that might be defined in the future. They are ignored for the purposes of interoperability with previous implementations.

11. Security Considerations

When creating a UUID value, endpoints MUST ensure that there is no user or device-identifying information contained within the UUID. In particular, this means that a UUID MUST NOT be constructed using a MAC address on the host.

The Session Identifier might be utilized for logging or troubleshooting, but MUST NOT be used for billing purposes.

The Session Identifier could be misused to discover relationships between two or more parties. For example, suppose that Alice calls Bob and Bob, via his PBX, forwards or transfers the call to Carol. Without use of the Session Identifier, an unauthorized third party that is observing the communications between Alice and Bob might not know that Alice is actually communicating with Carol. If Alice, Bob, and Carol include the Session Identifier as a part of the signaling messages, it is possible for the third party to observe that the endpoint associated with Bob changed to some other endpoint. If the third party also has access to signaling messages between Bob and
Carol, the third party can then discover that Alice is communicating with Carol. This would be true even if all other information relating to the session is changed by the PBX, including both signaling information and media address information.

12. IANA Considerations

12.1. Registration of the "Session-ID" Header Field

The following is the registration for the 'Session-ID' header field to the "Header Name" registry at http://www.iana.org/assignments/sip-parameters:

RFC number: RFC XXXX
Header name: 'Session-ID'
Compact form: none

Note: This document replaces the "Session-ID" header originally registered via [RFC7329].

[RFC Editor: Please replace XXXX in this section and the next with the this RFC number of this document.]

12.2. Registration of the "remote" Parameter

The following parameter is to be added to the "Header Field Parameters and Parameter Values" section of the SIP parameter registry:

+------------------+----------------+-------------------+-----------+
| Header Field     | Parameter Name | Predefined Values | Reference |
+------------------+----------------+-------------------+-----------+
| Session-ID       | remote         | No                | [RFCXXXX] |
+------------------+----------------+-------------------+-----------+

13. Acknowledgments

The authors would like to thank Robert Sparks, Hadriel Kaplan, Christer Holmberg, Paul Kyzivat, Brett Tate, Keith Drage, Mary Barnes, Charles Eckel, Peter Dawes, Andrew Hutton, Arun Arunachalam, Adam Gensler, Roland Jesske, and Faisal Siyavudeen for their invaluable comments during the development of this document.

14. References

14.1. Normative References


14.2. Informative References


[H.460.27] Recommendation ITU-T H.460.27, "End-to-End Session Identifier in for H.323 Systems", Work In Progress (RFC Editor: A month and year should be available when the RFC is published).
Authors’ Addresses

Paul E. Jones (Ed.)
Cisco Systems, Inc.
7025 Kit Creek Rd.
Research Triangle Park, NC 27709
USA
Phone: +1 919 476 2048
Email: paulej@packetizer.com
IM: xmpp:paulej@packetizer.com

Chris Pearce
Cisco Systems, Inc.
2300 East President George Bush Highway
Richardson, TX 75082
USA
Phone: +1 972 813 5123
Email: chrep@cisco.com
IM: xmpp:chrep@cisco.com

James Polk (Ed.)
Cisco Systems, Inc.
3913 Treemont Circle
Colleyville, Texas
USA
Phone: +1 817 271 3552
Email: jmpolk@cisco.com
IM: xmpp:jmpolk@cisco.com

Gonzalo Salgueiro
Cisco Systems, Inc.
7025 Kit Creek Rd.
Research Triangle Park, NC 27709
USA
Phone: +1 919 392 3266
Email: gsalguei@cisco.com
IM: xmpp:gsalguei@cisco.com