Abstract

This document defines two new event packages for the SIP Events architecture, along with two new data formats used in notifications for those packages. The first is a call-leg package, and the second is a conference package. The call-leg package allows users to subscribe to another user, and receive notifications about the changes in state of call legs that the user is involved in. The conference package allows users to subscribe to a URL that is associated with a conference. Notifications are sent about changes in the membership of this conference, changes in active speaker, and floor control information. We also define two new SIP headers, To-Replace and To-Join, that can be used to convey globally routable join and replacement URLs. These general purpose packages and the new headers enable many new SIP services. We discuss how they can be used to support some of the more challenging services that have been discussed, including single line extension, automatic callback,
unattended consultation-hold transfer, call park and pickup, and IM-a-call.

1 Introduction

The SIP Events architecture [1] defines general mechanisms for subscription to, and notification of, events within SIP networks. It introduces the notion of a package, which is a specific "instantiation" of the events mechanism for a well-defined set of events. Packages have been defined for user presence [2], watcher information [3], and message waiting indicators [4], amongst others. Here, we define two new packages - one for call legs, and the other for conferences.

The need for these packages is driven based on the fact that many applications are driven off of knowledge about the progress of calls and conferences. In the case of call legs, we see many potential applications that require knowledge of call-leg state:

Automatic Callback: In this basic PSTN application, user A calls user B. User B is busy. User A would like to get a callback when user B hangs up. When B hangs up, user A’s phone rings. When A picks it up, they here ringing, and are being connected to B. In VoIP, this requires A to receive a notification when the call-legs at A are complete.

Presence-Enabled Conferencing: In this application, a user A wishes to set up a conference call with users B and C. Rather than scheduling it, it is to be created automatically when A, B and C are all available. To do this, the server providing the application would like to know whether A, B and C are "online", not idle, and not in a phone call. Determining whether or not A, B and C are in calls can be done in two ways. In the first, the server acts as a call stateful proxy for users A, B and C, and therefore knows their call state. This won’t always be possible, however, and it introduces scalability, reliability, and operational complexities. Rather, the server would subscribe to the call state of those users, and receive notifications as it changes. This enables the application to be provided in a distributed way; the server need not reside in the same domain as the users.

IM Conference Alerts: In this application, a user can get an IM sent to their phone whenever someone joins a conference that the phone is involved in. The IM alerts are generated by an application separate from the conference server.
In general, defining call-leg and conference state packages allows for construction of distributed applications, where the application requires information on call-leg and conference state, but is not co-resident with the end user or conference server. We think this is a very important piece of the SIP services model.

2 Call-Leg Event Package

This section fills in the template that is needed in order to fully specify a SIP event package for call-leg state.

2.1 Package Name

The name of this event package is "call-leg". This package name is carried in the Event and Allow-Events header, as defined in [1].

2.2 SUBSCRIBE Bodies

A SUBSCRIBE for a call-leg package MAY contain a body. This body defines a filter to apply to the subscription.

A SUBSCRIBE for a call-leg package MAY be sent without a body. This implies the default subscription filtering policy. The default policy is:

- Notifications are generated every time there is any change in the state of any call legs for the user identified in the request URI of the SUBSCRIBE.

- Notifications do not normally contain full state; rather, they only indicate the state of the call-leg whose state has changed. The exception is a NOTIFY sent in response to a SUBSCRIBE. These NOTIFYs contain the complete view of call leg state.

- The notifications contain the identities of the participants in the call leg, the call-leg identifiers, and a join URL. Additional information, such as the route set, CSeq numbers, SDP information, and so on, are not included normally unless explicitly requested.

2.3 Expiration

Call leg state changes fairly quickly; once established, a typical phone call lasts a few minutes (this is different for other session types, of course). However, the interval between new calls is typically infrequent.
We do note that there are two distinct use cases for call leg state. The first is when a subscriber is interested in the state of a specific call leg (and they are authorized to find out about just the state of that call leg). In that case, when the call leg terminates, so too does the subscription. In these cases, the refresh interval can be very long, since there exists an easy alternative way to destroy subscription state. As a result, a default of one day for these subscriptions is RECOMMENDED.

In another case, a subscriber is interested in the state of all call legs for a specific user. In these cases, a shorter interval makes more sense. One hour is RECOMMENDED as the default.

2.4 NOTIFY Bodies

The body of the notification contains a call-leg information document. The format of this document is described in Section 3. All subscribers MUST support this format, and MUST list its type in an Accept header in the SUBSCRIBE.

Other call leg information formats might be defined in the future. In that case, the subscriptions MAY indicate support for other formats. However, they MUST always support and list application/call-leg-info+xml as an allowed format.

Of course, the notifications generated by the server MUST be in one of the formats specified in the Accept header in the SUBSCRIBE request.

2.5 Authorization Considerations

The call-leg information for a user contains very sensitive information. Therefore, all subscriptions SHOULD be authenticated and then authorized before approval. Authorization policy is at the discretion of the administrator, as always. However, a few recommendations can be made.

It is RECOMMENDED that if the policy of a user is that A is called to call them, subscriptions from user A be allowed. However, the information provided in the notifications does not contain any call leg identification information; merely an indication of whether the user is in one or more calls, or not. Specifically, they should not be able to find out any more information than if they sent an INVITE.

It is RECOMMENDED that if a user agent registers with the address-of-record X, that this user agent authorize subscriptions that come from any entity that can authenticate itself as X. Complete information on the call leg state SHOULD be sent in this case. This
authorization behavior allows a group of devices representing a single user to all become aware of each other’s state.

### 2.6 Generation of Notifications

Notifications are generated for the call-leg package when a new call-leg comes into existence at a UA, or when the state of an existing call leg changes.

For the purposes of this package, we define the states of a call leg through numeric codes. These codes are equivalent to the most recent SIP status codes sent in response to the INVITE which created the call leg. The status code "0" is reserved for the case where no response has yet been received or sent.

When a UAC initially creates an INVITE to establish a call, this causes a change to state "0". When it receives the first non-100 provisional response, the state changes to the value of that status code. Any further provisional responses cause the UA to change state to the value of that status code. When a final response is received, the state changes to the value of that response. If the response was a non-200, the call-leg is considered terminated, and no further state changes are possible. Multiple 2xx responses received create additional call legs, each with the state of that specific 2xx.

When a UAS initially receives an INVITE to establish a call, this causes a change to the state of the provisional response which was sent. Any subsequent provisional responses cause a change in state to the value of that response. A final response causes a transition in state to that response code. There is no change in state when the ACK arrives. However, if no ACK is received, and the UAS destroys the call, the state changes to a value of -1.

When the call is terminated as a result of a BYE, the state changes to -1.

**OPEN ISSUE: This is kind of ugly. We could alternately define a more formal state machine.**

### 2.7 Rate Limitations on NOTIFY

For reasons of congestion control, it is important that the rate of notifications not become excessive. As a result, it is RECOMMENDED that the server not generate notifications for a single subscriber at a rate faster than once every 5 seconds.

### 3 Call-Leg Data Format
We specify an XML-based data format to describe the state of call legs. The MIME type for this format is application/call-leg-info+xml, consistent with the recommendations provided in RFC 3023 [5].

3.1 Structure of Call Leg Information

A call-leg-info document starts with a user tag that identifies the user. Within that tag are a series of call-leg tags. Each of those use attributes to identify the call leg, using the local and remote URIs, local and remote tags, and the Call-ID. Within the call leg tags are a single mandatory tag which contains the status, followed by a series of optional tags that contain additional information about the call leg. There is also an optional tag called join, which contains a URL that can be used to join the conference associated with the call leg (and if there is none, one is created). There is another pair of optional tags called replace-local and replace-remote, which contain a URL to use that can replace the specific call leg at either side.

The top level tag is user:

```xml
<!ELEMENT user (call-leg*)>
<!ATTLIST user uri CDATA #REQUIRED>
```

The mandatory uri attribute is the identifier of the user whose call-leg state is being reported.

What follows is a series of call-leg tags:

```xml
<!ELEMENT call-leg (status,join?,replace-local?,replace-remote?,
local-sdp?,remote-sdp?,
route-set?,local-cseq?,remote-cseq?)>
<!ATTLIST call-leg
  call-id CDATA #IMPLIED
  local-uri CDATA #IMPLIED
  local-tag CDATA #IMPLIED
  remote-uri CDATA #IMPLIED
  remote-tag CDATA #IMPLIED>
```

The local-uri and local-tag specify the URL and tag placed in the From field of outgoing INVITEs, and present in the From field of incoming INVITEs. The remote-uri and remote-tag specify the URL and tag placed in the To field of outgoing INVITEs, and present in the
From field of incoming INVITEs. The call-id is the Call-ID for the leg. The tag attributes are not present if the tag is not specified (or not yet specified).

For example, if a UAC sends an INVITE that looks like, in part:

```
INVITE sip:callee@foo.com SIP/2.0
From: sip:caller@bar.com;tag=123
To: sip:callee@foo.com
Call-ID: 987@1.2.3.4
```

the call-leg tag sent out in a notification might looks like:

```
<call-leg call-id="987@1.2.3.4" local-uri="sip:caller@bar.com"
    local-tag="123" remote-uri="sip:callee@foo.com">
```

If a 200 OK is received, which looks like, in part:

```
INVITE sip:callee@foo.com SIP/2.0
From: sip:caller@bar.com;tag=123
To: sip:callee@foo.com;tag=abc
Call-ID: 987@1.2.3.4
```

The call-leg ID is now complete, and the notification sent out will have a call-leg tag which looks like:

```
<call-leg call-id="987@1.2.3.4" local-uri="sip:caller@bar.com"
    local-tag="123" remote-uri="sip:callee@foo.com" remote-tag="abc">
```

### 3.2 Call Leg Subtags

There are many subtags defined for the call-leg.

#### 3.2.1 Status

The only mandatory subtag of call-leg is status.
The mandatory code attribute contains the status code. This is the SIP response code last sent or received for this leg in the initial INVITE that established the leg. If no response has been sent or received, the value of zero is used. If the call ends, a value of -1 is used.

The value within the status tag is a textual phrase that can be rendered to described call status. The reason phrase from the response is RECOMMENDED.

Example:

```
<status code="180">Ringing</status>
```

### 3.2.2 Join

The optional join tag provides a URL that can be used to join any conference associated with the call-leg. When the notifier receives an INVITE with this URL, it MUST treat that as a request to join the conference. The notifier can use any method to create the conference call, if it doesn’t already exist. Several approaches are described in [6]. The format of the URL is at the discretion of the notifier. It is RECOMMENDED that it be structured so that the notifier can associate it with the specific call leg. As with any other invitations, an INVITE received with this URL SHOULD be authenticated.

The URL MUST be globally routable.

OPEN ISSUE: Tall order. This means UAs will need a way to generate URLs which they know are globally routable to them.
Clearly, the notifier SHOULD only insert this tag if it can execute a multi-party conference call for the user.

3.2.3 Replace Tags

The optional replace-local and replace-remote tags provide URLs that can be used to replace the given call leg at either side. In other words, if A are B are in a call, and A generated a NOTIFY with a replace-local and replace-remote tag, the replace-local URL would get routed to A, and replace the call leg with B. The replace-remote URL would get routed to B, and replace the call leg with A.

Replacement means that the new call leg is silently accepted, and a BYE is sent on the old call leg.

```xml
<!ELEMENT replace-local >
<!ELEMENT replace-remote >
<!ATTLIST replace-local     uri        CDATA     #REQUIRED>
<!ATTLIST replace-remote    uri        CDATA     #REQUIRED>
```

This has the very interesting implication of removing the need for a separate Replaces header. Instead, the URI itself would indicate to the UA that a replaces for a specific call leg is desired. This is very much in the spirit of the web, and of RFC 3087 [7], where URLs are almost always server generated, and the semantics of the URL have meaning only within the context of the server that created them.

Clearly, the notifier SHOULD only insert this tag if it can execute the call replacement.

3.2.4 Discussion on Join and Replaces

The inclusion of the join and replace tags merits discussion.

Do we really need join and replaces as part of this specification? We could instead define a Join header, and revive the Replaces header draft. That has the benefit of being explicit. The benefit of the approach here is that we get to define a totally different URL. By making it globally routable, we fix the "unassisted transfer with consultation hold" problem [8]. Specifically, this transfer variant is hard since the transferor needs to pass, in the Refer-To header, a globally routable URL that reaches the specific UA that is the
transfer target. Neither the Contact nor the To/From have this property. But, by defining a specific URL just for the purposes of replacement, we can, by definition, make it globally routable to the UA which generated it. That seems very, very useful. It means that, unlike Contact and To/From, this URL can be emailed, IM’ed, REFERed, placed in html, or whatever, and it is guaranteed to get to the right place and do the right thing. Furthermore, by generating the URLs for each call, the notifier can embed information into the URLs for cookie functionality. Finally, using a separate URL means the join or replace requests can go to different hosts that the ones owning the call leg. Not sure if this is useful, though.

The replace-remote URL can only be known to a notifier for call-leg events if its received from the other party in the call-leg in the INVITE or 200 OK. As such, we would also propose a To-Join and To-Replace header, which is present in either INVITE or 200 OK. It contains a URL that can be used to replace the call leg for each respective side. Putting them both in a NOTIFY means that other entities besides the participants in a call can get at them, which is possibly useful (operator barge-in, for example).

It’s worth observing that currently, there is an asymmetry between the way one joins a conference, and the way one replaces a call leg. To join a conference call, we have agreed that one sends an INVITE to a URL on a conference server which interprets that URL to mean “mix me into the call with everyone else that has used the same URL”. This is exactly consistent with the approach here for joining. However, we have defined an explicit Replaces header. Why the difference? We should either have both a Join and Replaces header, or neither.

Indeed, there is a problem right now with ad-hoc conferences that is related to the proposal here. Lets say A clicks a URL in a web page that says “click here to call”. This results in some call being placed to B. At some point during the call, A decides to add another party, C. According to the multiparty conferencing models draft [6], A would obtain a URL for a conference server, and REFER B to that server. However, this is totally unneeded (and bad), if B is already a conference server! Indeed, A has no way, right now, of ascertaining whether that URL in the web page is for a single user, or to a conference server that automatically dials out to B. In one case (where its a user), A would need to obtain a conference URL, and REFER B to it. In the other, A could directly REFER C to the URL it used to call B.

The problem is solved when we stop making assumptions about the semantics of the URL. URLs should ideally be handed to a user, either through interpersonal contact (i.e., on a business card), or through a transmission mechanism that defines a specific usage for those
URLs. In the above scenario, if A wants to add C, A should not make any assumptions about whether the URL to call B can be used for adding users to a conference. Only B can know this. So, A needs to ask B what URL to use to join. That can be done using the SUBSCRIBE/NOTIFY mechanisms here (although this is a lot of overhead; these URLs could be passed directly in the INVITE and 200 OK). B would then provide the URL, which would be the same URL used to call it in the case of a dial-out conference server, or a new URL obtained by B, pointing to a conference server, if B was for an end user that didn’t want to do endpoint mixing. Indeed, many different URLs could be used, and B is the ideal party to decide.

3.2.5 Local SDP

The local SDP tag contains the SDP used by the notifier for its end of the call leg. This tag should generally NOT be included in the notifications, unless explicitly requested by the subscriber.

<!ELEMENT local-sdp CDATA>

The SDP is included, verbatim, between the tags.

3.2.6 Remote SDP

The remote SDP tag contains the SDP used by the notifier for the other end of the call leg. This tag should generally NOT be included in the notifications, unless explicitly requested by the subscriber.

<!ELEMENT remote-sdp CDATA>

The SDP is included, verbatim, between the tags.

3.2.7 Route Set

The route-set tag contains the route set, as defined in RFC 2543bis [9]. It is the combination of the Record-Route and Contact headers used for this call leg. This tag should generally NOT be included in the notifications, unless explicitly requested by the subscriber.

<!ELEMENT route-set CDATA>
The route set is included verbatim. It is structured as a comma-separated list of URLs.

Example:

```
<route-set>sip:user@host,sip:user@proxy</route-set>
```

3.2.8 Local CSeq

The local-cseq tag contains the most recent value of the CSeq header used by the UA in an outgoing request on the call leg. This tag should generally NOT be included in the notifications, unless explicitly requested by the subscriber.

```
<!ELEMENT local-cseq CDATA>
```

The numeric value of the CSeq is included as the CDATA.

3.2.9 Remote CSeq

The remote-cseq tag contains the most recent value of the CSeq header seen by the UA in an incoming request on the call leg. This tag should generally NOT be included in the notifications, unless explicitly requested by the subscriber.

```
<!ELEMENT remote-cseq CDATA>
```

The numeric value of the CSeq is included as the CDATA.

4 Conference Event Package

The conference event package allows a user to subscribe to a conference. A conference is a collection of users that are all able to communicate with each other. Generally, when multicast is not used, a conference is associated by a set of call legs that have their media mixed together. This is true for all of the non-multicast models in [6]. However, some of the models use topologies where there is no root to which all call-legs are connected. These topologies do not work well with the mechanism here.
This package allows a user to subscribe to a conference, identified by a SIP URL. Ideally, this SIP URL routes the SUBSCRIBE to the entity acting as the root of the topology (which is why it doesn’t work well for the non-centralized topologies). The notifications contain information on the participants in the conference. The specific information conveyed is:

- The SIP URL identifying the user.
- Their status in the conference (active, declined, departed).
- The replace URLs for the call leg connecting to that user.
- If floor control policies are in place, what the user’s floor control status is.

This section fills in the template that is needed in order to fully specify the SIP event package for conferences.

4.1 Package Name

The name of this event package is "conference". This package name is carried in the Event and Allow-Events header, as defined in [1].

4.2 SUBSCRIBE Bodies

A SUBSCRIBE for a call-leg package MAY contain a body. This body defines a filter to apply to the subscription.

A SUBSCRIBE for a conference package MAY be sent without a body. This implies the default subscription filtering policy. The default policy is:

- Notifications are generated every time there is any change in the set of users participating in the conference, or a change in floor control status (assuming floor control is in use).
- Notifications do not normally contain full state; rather, they only indicate the state of the participant whose state has changed. The exception is a NOTIFY sent in response to a SUBSCRIBE. These NOTIFYs contain the complete view of conference state.
- For a given user, the notifications contain the identity information, status, and replace URLs. The floor control information is not provided unless explicitly requested.

4.3 Expiration
The default expiration time for a subscription to a conference is one hour. Of course, once the conference ends, all subscriptions to that particular conference are terminated.

4.4 NOTIFY Bodies

The body of the notification contains a conference information document. The format of this document is described in Section 5. All subscribers MUST support this format, and MUST list its type in an Accept header in the SUBSCRIBE.

Other conference information formats might be defined in the future. In that case, the subscriptions MAY indicate support for other formats. However, they MUST always support and list application/conference-info+xml as an allowed format.

Of course, the notifications generated by the server MUST be in one of the formats specified in the Accept header in the SUBSCRIBE request.

4.5 Authorization Considerations

The conference information contains very sensitive information. Therefore, all subscriptions SHOULD be authenticated and then authorized before approval. Authorization policy is at the discretion of the administrator, as always. However, a few recommendations can be made.

It is RECOMMENDED that all users in the conference be allowed to subscribe to the conference.

4.6 Generation of Notifications

Notifications are generated for the conference whenever a new participant joins, a participant leaves, a dial-out attempt succeeds or fails, floor control status changes, or the call leg replace URLs change.

4.7 Rate Limitations on NOTIFY

For reasons of congestion control, it is important that the rate of notifications not become excessive. As a result, it is RECOMMENDED that the server not generate notifications for a single subscriber at a rate faster than once every 5 seconds.

5 Conference Data Format

The conference data format is an XML document of MIME type
application/conference-info+xml, consistent with the recommendations provided in RFC 3023 [5].

5.1 Structure of the Format

The conference data format has the top level tag of conference. It consists of a set of sub-tags of type user, which contain information on the users in the conference. Each user tag contains the identity of the user, their status, their replace URL, and their floor control status.

The top level tag is conference:

```xml
<!ELEMENT conference (user*)>
<!ATTLIST conference uri CDATA #REQUIRED>
```

The mandatory uri attribute contains the URL used to join the conference call (and to subscribe to its state).

What follows are a series of user tags:

```xml
<!ELEMENT user (status,replace?,floor-status?)>
<!ATTLIST user   uri    CDATA   #REQUIRED
   name   CDATA   #IMPLIED>
```

The uri attribute contains the URL for the user. This is a logical identifier, not a machine specific one (i.e., its taken from the To/From, not the Contact). The name is a textual name for rendering to a human. It is usually taken from the display name.

5.2 User Sub-Elements

The sub-elements of the user tag are status, replace, and floor-status.

Status contains the status of the user in the conference.

```xml
<!ELEMENT status>
<!ATTLIST status
   value   (active|departed|booted|failed) "active" >
```
The statuses have the following meaning:

- active: The user is in an active call leg with the conference host.
- departed: The user sent a BYE, thus leaving the conference.
- booted: The user was sent a BYE by the conference host, booting them out of the conference.
- failed: The conference host is a dialout conference server, and its attempt to contact the specific user resulted in a non-200 class final response.

The replace URL is the same replace-remote URL defined for the call leg package above. It is exposed through the conference server, so that subscribers have, if authorized, the ability to pull a user out of the conference.

OPEN ISSUE: Do we really want or need this? The only real use I found was to back out of a centralized conference server, to a point-to-point call, when only two users remain. Is this sufficient need? Other uses?

```xml
<!ELEMENT replace>
<!ATTLIST url CDATA #IMPLIED>
```

The floor-status contains the status of the user as far as floor control is concerned.

```xml
<!ELEMENT floor-status>
<!ATTLIST status
  value (owner|non-owner|chair) "non-owner" >
```

The values have the following meaning:

- owner: The user has floor control.
- non-owner: The user does not have floor control.
- chair: The user is the chair, and is the one who controls who
gets floor control.

OPEN ISSUE: Does this belong here? If we have a separate floor control protocol, perhaps the notifications of state changes are in the specific protocol for floor control. Or, perhaps this is a separate package.

5.3 Example

The following is an example conference information document:

```xml
<conference>
  <user uri="sip:jdrosen@dynamicsoft.com" name="Jonathan Rosenberg">
    <status value="active"/>
    <replace uri="sip:p09asd8asd-0-f88a70@dynamicsoft.com"/>
  </user>
  <user uri="sip:hgs@cs.columbia.edu" name="Henning Schulzrinne">
    <status value="active"/>
    <replace uri="sip:nj99999skasdka@cs.columbia.edu"/>
  </user>
</conference>
```

This document describes a conference with two users, both of which are active.

6 Relationship to User Presence

The SIP events package for user presence [2] has a close relationship with these two event packages. It is fundamental to the presence model that the information used to obtain user presence is constructed from any number of different input sources. Examples of such sources include SIP REGISTER requests and uploads of presence documents. These two packages can be considered another mechanism that allows a presence agent to determine the presence state of the user. Specifically, a user presence server can act as a subscriber for the call-leg and conference packages to obtain additional information that can be used to construct a presence document.

7 Example Services

This section overviews some example services that can be enabled by the extensions described here.

7.1 Automatic Callback
Automatic callback is a simple service. User A calls user B. User B is already on the phone, and so returns a 486 Busy to the INVITE from A. Rather than continually trying to call B, user A asks for automatic callback. With this service, A’s phone will ring when B is available. When A picks up, A hears ringing, which is B’s phone.

The call flow for this service is shown in Figure 1. In (1-3), A calls B, but B is busy. So, in (4), A sends a SUBSCRIBE to B. This results in a 200 OK (5) followed by a NOTIFY with the current call-leg state for B. This is a call state document which looks like:

```xml
<user uri="sip:B@B.com">
  <call-leg>
    <status code="200"/>
  </call-leg>
</user>
```

The call leg identifiers are not included in this notification, because this is not information that A would normally be able to obtain by sending an INVITE.

When B hangs up their call, this causes a second notify containing document 2:

```xml
<user uri="sip:B@B.com">
  <call-leg>
    <status code="-1"/>
  </call-leg>
</user>
```

A can then try the INVITE again.

7.2 Single Line Extension

In the single line extension application, we wish to have a group of phones which are all treated as "extensions" of a single line. This means that a call for one rings them all. As soon as one picks up, the others stop ringing (all of that is standard forking behavior). The additional complexity is that once the call is answered, one of the extensions should be able to "pick up" and join the call. This emulates the home phone behavior.
This feature is enabled by the mechanisms described in this draft. The basic idea is that the phones all share a common SIP URL, and there is a forking proxy that forwards calls to all of them. The

---

A's phone rings
A picks up

B

---

Figure 1: Automatic callback flow
belonging to the same extension group. The phones are also configured with the address of a standard conferencing server, as described in [10]. To support the feature, only the phones have to know about how to do it.

The call flow for this feature is shown in Figures 2 and 3. First, the caller sends an INVITE to call a phone in the extension group (1). This is a normal INVITE, with a request URI of sip:joe@joeshouse.net, for example. This INVITE arrives at the proxy serving Joe’s house. Since both extension 1 and extension 2 had registered contacts for this URL previously (not shown), the INVITE is forked to extension 1 (2) and extension 2 (3). Joe picks up extension 1, generating a 200 OK to the INVITE (4). The proxy forwards the 200 OK from extension 1 upstream (5), and cancels the other branch (6), which causes extension B to stop ringing, and then response 200 OK to the CANCEL (7), followed by a 487 to the INVITE (8). The proxy ACKs the 487 (9). The caller sends an ACK for the 200 OK (10), and the caller is now talking to Joe on extension 1. In order to keep track of that call, so that its status can be displayed in its UI, extension 2 now generates a SUBSCRIBE for sip:joe@joeshouse.net. This goes to the proxy (11), which forks it as it would any normal request. It gets forked to extension 1 (12) and extension 2 (13). Extension 2 receives its own SUBSCRIBE, and generates a 482 Loop Detected error response (14). Extension 1 accepts the SUBSCRIBE, and sends a 200 OK (15) (its assumed that the SUBSCRIBE was authenticated with the shared secret; the 401 and resubmission are not shown). The 200 OK is forwarded back to extension 2 (16). In the event that there were more than two extensions, only a single 200 OK would still be returned to extension 2. However, as described in [1], the NOTIFY’s that are generated will allow extension 2 to find out about all of the other extensions which accepted the subscription. In this case, its just one, extension 1. It generates a NOTIFY (17) that contains document 1, shown below:

```
<user uri="sip:joe@joeshouse.net">
  <call-leg local-uri="sip:joe@joeshouse.net" remote-uri="sip:caller@foo.edu"
    local-tag="00a9" remote-tag="ffd2" call-id="aa9@1.2.3.4">
  
  <status code="200"/>
  <replace-local uri="sip:00a9.ffd2.aa9_1-2-3-4@ex1.joeshouse.net">
  </replace-local>
  <replace-remote uri="sip:ffd2.00a9.aa9_1-2-3-4@hispc.foo.edu">
  </replace-remote>
 <join uri="sip:join-00a9.ffd2.aa9_1-2-3-4@ex1.joeshouse.net">
 </call-leg>
</user>
```
Figure 2: Single line extension flow: Part I
This allows extension 2 to show that a call is active on the extension group, and a UI can be provided for the call to be picked up by someone on extension 2.

At some point, Joe’s wife picks up extension 2 in order to join the active call. That causes the phone to send an INVITE to the join URI in the notification it received. This INVITE (19) goes directly to extension 1. When extension 1 receives this, it knows this is a request to join the call. It challenges and authenticates the INVITE to make sure that its another extension in the group (not shown). It then redirects the call, providing a Contact header which is a new conference URI at the conference server (20). Presumably, each extension is configured with the domain name of the conference server, and can create conferences by choosing usernames that are globally unique in space and time. The resulting user@domain SIP URL can be used for ad-hoc conference calls, like this one. Extension 2 ACKs the 300 (21). Extension 1 knows it needs to join that conference call. So, it sends an INVITE to the conference URI it just returned to extension 2 (23), which is accepted (25) and acknowledged (26).

Extension 1 is now the only member of the call. In the meantime, extension 1 knows that it needs to get the caller into the conference as well. So, it sends out a REFER (22), containing a Refer-To URL that points to the conference URL being used. The REFER is accepted by the caller (24). Extension 2 recurses on the redirect it receives, and sends an INVITE to the conference URL (27), which is accepted (28) and acknowledged (29). Finally, the caller acts on the REFER, and generates an INVITE (30) that will join it into the conference as well. This is accepted (31) and acknowledged (32). Now, there is a three-party conference between the caller, extension 1 and extension 2. The caller generates a NOTIFY as discussed in [8] (33) which is accepted (34). The NOTIFY tells extension 1 that the caller has been connected to the conference server. So, extension 1 terminates its direct call leg with the caller (35).

If other extensions pick up, a similar thing happens - they are redirected to the conference URL. By using a conference server, we have the advantage that the call remains active as long as any one extension is in the call. This also emulates typical home phone line behavior.

OPEN ISSUE: Its not clear that extension 1 should REFER the caller to the conference server. We want the change to the conference server to be transparent to the caller. A REFER will trigger a UI query at the caller, most likely. An alternative is to have extension 1 REFER the conference server to the caller, using the replaces URL learned from
Figure 3: Single line extension flow: Part II
the caller as the Refer-To header. This works better. However, the INVITE triggered from this will be challenged by the caller, and its not clear how the conference server will obtain credentials.

7.3 Unattended Consultation-Hold Transfer

In unattended consultation-hold transfer, A is talking to B. A wishes to transfer B to C. So, A first calls C, and talks to them to OK the transfer. If C agrees, B is connected to C.

As discussed in [8], this flow is difficult. The main problem is that once the transfer has been verbally approved, A needs to send a REFER to C, containing B in the Refer-To header. However, what is desired is to refer C to that very specific instance of B. Placing the To/From with B into the Refer-To therefore won’t work, since it won’t necessarily route to the UA that user B is using. Putting the Contact at B may not work either, since it may not be globally routable.

Our To-Replace header fixes this. By definition, it contains a globally routable URL which can be used to replace the specific call leg. B would return this in its 200 OK to A. When A REFERs C to B, this URL would be placed in the Refer-To header. The result is that the service executes perfectly.

7.4 Call Park and Pickup

In the PSTN, call park and pickup is defined as follows. Joe (using UA A) is talking to Bob (using UA B). Joe would like to walk over to another phone (UA C) that Joe can see, but doesn’t have the number for, and pick continue on the call at that new phone. To do that, Joe places Bob on hold, walks over to the phone, picks it up, dials some numbers, and then talks to Bob.

A SIP flow for call park is given in the service examples document [11]. However, that service only works when the parking phone is the same as the picking up phone (and thus is more like music-on-hold). There is also a flow for pickup. This relies on registrations to cause an in-progress INVITE to fork to the new phone. However, this only works for calls that haven’t yet completed at the first phone.

The flow for the call park and pickup service is shown in Figure 4. First, A calls B as a normal SIP call (1-3). The INVITE (1) contains a To-Replace header with the value sip:00a9.ffd2.aa9_1-2-3-4@mypc.company.com, and the 200 OK has a To-Replace header with value sip:ffd2.00a9.aa9_1-2-3-4@hispc.foo.edu. At some point, Joe decides to park the call for later pickup. A simply places B on hold (4-6).
Joe walks over to another phone, C, and enters in his extension (i.e., the identity of the phone which the call is being retrieved from). To pickup, C needs to learn the call legs at A. So, it sends a SUBSCRIBE to A’s extension (7). The SUBSCRIBE is authenticated (Joe will need to enter his own username and password), which is not shown, and the resubmitted SUBSCRIBE generates a 200 OK (8). That triggers a NOTIFY (9) which contains the call leg state at A:

```
<user uri="sip:A@company.com">
  <call-leg local-uri="sip:A@company.com"
    remote-uri="sip:B@foo.edu"
    local-tag="00a9"
    remote-tag="ffd2"
    call-id="aa9@1.2.3.4">
    <status code="200"/>
    <replace-local uri="sip:00a9.ffd2.aa9_1-2-3-4@mypc.company.com">
      <replace-remote uri="sip:ffd2.00a9.aa9_1-2-3-4@hispc.foo.edu">
    </call-leg>
  </user>
```

From this, C learns there is a single accepted call at A. The notification also contains a replace-remote URL, which can be used to replace the existing call leg at B with a new one. So, C takes that URL, and generates an INVITE for it (15). This is authenticated and authorized by B (specifically, B allows a call-leg to be replaced if the authenticated identity of the new leg matches the identity of the replaced leg), and then silently accepted (16). Now, Joe is talking to Bob using UA C. Since B has replaced its old call leg, it sends a BYE on it (18). UA A is now disconnected from the call.

### 7.5 IM a call

This service is a much "cooler" variant on transfer. A calls B, and they talk. During the call, A wants C to take over the call. Rather than sending a REFER to execute a transfer, A sends C an instant message. This IM has HTML inside of it, which asks C to click on a URL to take the call. When C clicks on it, they take over the call, and A is disconnected.

The call flow for this is shown in Figure 5. A calls B using a standard INVITE sequence (1-3). The 200 OK from B contains a To-Replace URL. A decides to send the call to B using an IM. So, it sends a MESSAGE (4) that has an HTML body. This request looks like:
Figure 4: Call Park and Pickup
Figure 5: Call flow for IM-a-call

```
MESSAGE sip:C@foo.com SIP/2.0
Via: SIP/2.0/UDP pc22.foo.com
From: sip:A@foo.com
To: sip:C@foo.com
Call-ID: 9as8da8s@1.2.3.4
CSeq: 99 MESSAGE
Content-Type: text/html
Content-Length: ...
```
Hi, Jack. Would you take this call? Thanks, Bob.

Where sip:hhggff@bar.com is the URL from the To-Replace header in the 200 OK from B (message 2).

C returns a 200 OK to this MESSAGE (5). At some time later, C clicks on the URL in the MESSAGE, which causes a call to be made to the replace URL. This goes to B. This is a standard INVITE sequence (7-9). Once done, since B knows this is a replace URL, it hangs up with A (10-11).

8 Open Issues and To-Dos

There is a strong relationship between the call-leg event package, and the notifications used by the REFER specification [8]. We believe that these should be unified, so that a REFER basically implies a subscription to the call leg state created by that REFER. That still needs to be done.

There are many security issues to be worked out. Authentication of join and replaces INVITEs are complex, and need further investigation.

The requirement for globally routable join and replaces URLs is a real issue. Its not clear how that can be done. Using the hostname of the UA wont work in general, since calls may need to flow through a proxy. This introduces the need for a UA to generate a new URL which contains the domain name of the top-level proxy in its network, yet is routed to that UA. The UA could then REGISTER this URL at its proxy. This might work in some networks, but not in more complex ones with multiple tiers of proxies, some of which use database queries to route calls to specific users. More thinking on this is needed.

More details and examples are needed.

9 Security Considerations

Subscriptions to call-leg state and conference state can reveal very sensitive information. For this reason, the document recommends authentication and authorization, and provides guidelines on sensible authorization policies.
Since the data in notifications is sensitive as well, end-to-end SIP encryption mechanisms SHOULD be used to protect it.

Furthermore, the To-Replace and To-Join URLs provide significant power to any user that can obtain them. INVITEs to these URLs SHOULD be authenticated and authorized.

10 Authors Addresses

Jonathan Rosenberg
dynamicsoft
72 Eagle Rock Avenue
First Floor
East Hanover, NJ 07936
e-mail: jdrosen@dynamicsoft.com

Henning Schulzrinne
Columbia University
M/S 0401
1214 Amsterdam Ave.
New York, NY 10027-7003
e-mail: schulzrinne@cs.columbia.edu

11 Bibliography


