Cisco BroadWorks

SIP Access Side Extensions Interface

Specification Guide

Document Version 2

Copyright Notice

Trademarks

Any product names mentioned in this document may be trademarks or registered trademarks of Cisco or their respective companies and are hereby acknowledged.

Document Revision History

Release	Version	Reason for Change	Date	Author
14.0	1	Updated document for rebranding.	February 15, 2006	Roberta Boyle
14.0	1	Updated document with Release 14 enhancements.	June 28, 2006	Sam Hoffpauir
14.0	1	Edited document.	July 5, 2006	Patricia Renaud
14.0	2	Updated document with Release 14.sp1, Release 14.sp2, and Release 14.sp3 content.	August 27, 2007	Martin Piotte
14.0	2	Edited changes and published document.	September 28, 2007	Andrea Fitzwilliam
14.0	3	Updated document with Release 14.sp4 content.	December 20, 2007	Martin Piotte
14.0	3	Edited changes and published document.	April 3, 2008	Andrea Fitzwilliam
15.0	1	Updated document for Release 15.0.	June 17, 2008	Martin Piotte
15.0	1	Edited changes and published document.	July 24, 2008	Andrea Fitzwilliam
15.0	2	Added clarification for the Application Server Feature Event package for EV 88108.	January 20, 2009	Martin Piotte
15.0	2	Edited changes and published document.	January 29, 2009	Andrea Fitzwilliam
16.0	1	Updated document for Release 16.0.	May 19, 2009	Martin Piotte
16.0	1	Edited changes and published document.	July 13, 2009	Andrea Fitzwilliam
16.0	2	Updated section *3.2 Appearance-Index, Appearance-State, and Appearance-URI* for EV 94612.	August 13, 2009	Roberta Boyle
16.0	2	Updated section *8.3 Example Message Flows* for EV 94427.	August 26, 2009	Roberta Boyle
16.0	2	Edited changes and published document.	February 9, 2010	Andrea Fitzwilliam
17.0	1	Updated document with Release 17.0 content.	April 8, 2010	Martin Piotte
17.0	1	Edited changes and published document.	April 21, 2010	Margot Hovey-Ritter
17.0	2	Added call center status event package for EV 113116.	June 10, 2010	Martin Piotte
17.0	2	Edited changes and published document.	June 25, 2010	Andrea Fitzwilliam
18.0	1	Updated document with Release 18.0 content.	November 4, 2011	Martin Piotte
18.0	1	Edited changes and published document.	November 8, 2011	Patricia Renaud

Release	Version	Reason for Change	Date	Author
18.0	2	Updated section *8.6.2 Set Forwarding* for EV 142029: Clarified that the ringCount element in the SetForwarding body is required for CFNA.	November 21, 2011	Doug Sauder
18.0	2	Edited changes and published document.	December 16, 2011	Jessica Boyle
19.0	1	Updated for Release 19.0	September 18, 2012	Doug Sauder
19.0	1	Edited changes and published document.	October 24, 2012	Patricia Renaud
19.0	2	Corrected call flow for Shared Call Appearance (SCA) Retrieve using *Replaces* header for EV 184045.	July 10, 2013	Doug Sauder
20.0	1	Updated document with Release 20.0 content.	October 4, 2013	Doug Sauder
20.0	1	Edited changes and published document.	November 1, 2013	Joan Renaud
21.0	1	Updated document with Release 21.0 content.	October 16, 2014	Doug Sauder
21.0	1	Edited changes and updated Cisco legal notice.	November 1, 2014	Joan Renaud
22.0	1	Updated document for Release 22.0.	September 28, 2016	Doug Sauder
22.0	1	Edited changes and published document.	December 7, 2016	Joan Renaud
23.0	1	Corrected the information about the SUBSCRIBE request for the “dialog” event package for PR-58398.	July 9, 2018	Doug Sauder
23.0	1	Updated document for Release 23.0.	September 14, 2018	Doug Sauder
23.0	1	Rebranded document for Cisco. Edited changes and published document.	November 15, 2018	Joan Renaud
23.0	2	Completed rebranding for Cisco and republished document.	March 12, 2019	Jessica Boyle
24.0	1	Updated document for Release 24.0.	April 30, 2020	Sebastien Cossette
24.0	1	Edited changes.	May 14, 2020	Jessica Boyle
24.0	1	Added keywords to the document properties, edited changes, and published document.	June 15, 2020	Jessica Boyle
24.0	2	Made minor changes and published document.	July 28, 2020	Jessica Boyle

Table of Contents

Table of Figures

1 Summary of Changes

This section describes the changes to the interface for each release.

1.1 Changes for Release 24.0, Document Version 2

There are no changes to the BroadWorks SIP extensions interface introduced in BroadWorks Release 24.0.

1.2 Changes for Release 23.0, Document Version 1

There are no changes to the BroadWorks SIP extensions interface introduced in BroadWorks Release 23.0.

1.3 Changes for Release 22.0, Document Version 2

The following change was made in this document version:

- Corrected the information about the SUBSCRIBE request for the “dialog” event package for PR-58398.

1.4 Changes for Release 22.0, Document Version 1

The following change was made in this document version:

- Corrected information about the XML content for the dialog event package for PR-52324.

1.5 Changes for Release 21.0, Document Version 1

There are no changes to the Cisco BroadWorks SIP extensions interface introduced in BroadWorks Release 21.0.

1.6 Changes for Release 20.0, Document Version 1

The following BroadWorks SIP extensions interface changes have been introduced in BroadWorks Release 20.0. The following changes are the interface differences between BroadWorks Release 19.0 and Release 20.0:

- Added information about these new parameters to the *Call-Info* header: *silent-monitor*, *client-session-info,* and *conference-subscription-uri*.
- Added a description of the *appid* parameter in the *From* header Uniform Resource Identifier (URI) and Cisco BroadWorks support for a single-login policy for clients that subscribe to the Call-Info package.
- Added information about these extensions to the as-feature-event event package: Executive, Executive-Assistant, Security Classification, and Call Recording.
- Added information about the new Security Classification INFO package.
- Added information about the new Client Session Information INFO package.

1.7 Changes for Release 19.0, Document Version 2

The following change was made in this document version:

- Corrected the call flow for Call Retrieve using *Replaces* header in section *9.3 Retrieve Call using Replaces Header*.

1.8 Changes for Release 19.0, Document Version 1

There are no changes to the Cisco BroadWorks SIP extensions interface introduced in BroadWorks Release 19.0.

1.9 Changes for Release 18.0

The following Cisco BroadWorks Session Initiation Protocol (SIP) extensions interface changes have been introduced in BroadWorks Release 18.0. The following changes are the interface differences between BroadWorks Release 17.0 and Release 18.0:

- Added the Call Park Event Package to section *12 Call Park Event Package*.
- Added the Call Park information to Call-Info (section *4 Call-Info Event Package*) and Dialog (section *9 Alternate Signaling for Shared Call Appearance*) Event Packages.
- Updated section *8.6.2 Set Forwarding* for EV 142029: clarified that the ringCount element in the SetForwarding body is required for Call Forwarding No Answer (CFNA).

1.10 Changes for Release 17.0

The following Cisco BroadWorks Session Initiation Protocol (SIP) extensions interface changes have been introduced in BroadWorks Release 17.0. The following changes are the interface differences between BroadWorks Release 16.0 and Release 17.0:

- Added the *AgentNotReadyReasonValue* to the Application Server feature event package (sections *8.6.3 Set Agent State* and *8.8.5 Agent Not Ready Event*).
- Added support for the Hoteling event package (section *10 Hoteling Event Package*).
- Added support for the Call Center Status event package (section *11 Call Center Status Event Package*) for EV 113116.

1.11 Changes for Release 16.0

The following Cisco BroadWorks Session Initiation Protocol (SIP) extensions interface changes have been introduced in BroadWorks Release 16.0. The following changes are the interface differences between BroadWorks Release 15.sp2 and Release 16.0:

- Added support for Shared Call Appearance (SCA) using the dialog event package in conjunction with the *Join* and *Replaces* SIP headers.
- Updated section *8.3 Example Message Flows* for EV 94427.
- Updated section *3.2 Appearance-Index, Appearance-State, and Appearance-URI* for EV 94612.

1.12 Changes for Release 15.sp2

There are no changes to the Cisco BroadWorks SIP extensions interface introduced in BroadWorks Release 15.sp2.

1.13 Changes for Release 15.0

There are no changes to the Cisco BroadWorks SIP extensions interface introduced in BroadWorks Release 15.0. There were no changes to this document for Release 15.0.

1.14 Changes for Release 14.sp6

There are no changes to the Cisco BroadWorks SIP extensions interface introduced in BroadWorks Release 14.sp6.

1.15 Changes for Release 14.sp5

There are no changes to the Cisco BroadWorks SIP extensions interface introduced in BroadWorks Release 14.sp5.

1.16 Changes for Release 14.sp4

There are no changes to the Cisco BroadWorks SIP extensions interface introduced in BroadWorks Release 14.sp4.

1.17 Changes for Release 14.sp3

The following Cisco BroadWorks SIP extensions interface changes are introduced in BroadWorks Release 14.sp3. The following changes are the interface differences between BroadWorks Release 14.sp2 and Release 14.sp3.

- Support for SIP phones to synchronize with the Cisco BroadWorks Application Server on the status of features is extended to include Call Center Agent Automatic Call Distribution (ACD) state.

1.18 Changes for Release 14.sp2

The following Cisco BroadWorks SIP extensions interface changes were introduced in BroadWorks Release 14.sp2. The following changes are the interface differences between BroadWorks Release 14.sp1 and Release 14.sp2:

- Added support for SIP phones to synchronize with the Cisco BroadWorks Application Server on the status of the following features: Do Not Disturb, Call Forwarding Always (CFA), Call Forwarding Busy (CFB), and Call Forwarding No Answer (CFNA).

1.19 Changes for Release 14.sp1

There is no change in the BroadWorks SIP extensions interface between BroadWorks Release 14.0 and Release 14.sp1.

1.20 Changes for Release 14.0

The following Cisco BroadWorks SIP extensions interface changes are introduced in BroadWorks Release 14.0. The following changes are the interface differences between BroadWorks Release 13.0 and Release 14.0:

- Added support for two new call-info event package appearance-states:
  - bridge-active
  - bridge-held

The two new appearance-states are used with the Shared Call Appearance service to allow different locations associated with a user to barge on to an active call involving another location. This bridging capability, in conjunction with the existing multiple call arrangement capability, allows Cisco BroadWorks to offer a complete-hosted key-system solution. Furthermore, this capability also enhances Cisco BroadWorks executive/administrator solution to fill a functional gap that exists with some Private Branch Exchange (PBX) executive/administrator solutions where the administrator is able to barge on to the executive calls.

2 Introduction

This document describes extensions to the Session Initiation Protocol (SIP) that can be used to enable traditional voice applications, including but not limited to, key system emulation, executive-administrator station emulation, Push-To-Talk, Click-To-Dial, and other remote control Computer Telephony Integration (CTI) applications. These extensions are typically employed between a line side Application Server and an end-user agent, such as an IP phone or soft client. Application Servers that employ these extensions can provide a tightly integrated end-user experience and a rich service offering, without being tightly coupled to the access device.

Note that these extensions complement existing standards and draft extensions aimed at integrating end-user access equipment with operator-hosted service platforms.

These extensions have been adopted and implemented by many equipment vendors in the industry, including IP PBX vendors, Internet Protocol (IP) phone vendors, access gateway vendors, and Application Server vendors.

3 Call-Info Header Extensions

3.1 Overview

In many traditional voice applications, it is important for the endpoint and the service delivery platform to maintain consistent “presentation” information. By this we mean, the relative order of call appearances on a line and the current state of the call appearances. This enables call control clients, attendant consoles, and other applications to maintain a synchronized view of call appearance information, so that end users can move from one endpoint or one interface to another, without being confused.

The *Call-Info* header is specified in *RFC 3261* [2]. It is used to provide additional information about the calling or called party, depending on whether it is found in a request or a response. Here we use this header to send “presentation” information about the call appearances associated with a given address of record (that is, a line). Following is an excerpt from the augmented Backus-Naur Format (ABNF) in *RFC 3261* [2] (for the *Call-Info* header):

```
Call-Info   =  "Call-Info" HCOLON info *(COMMA info)
info        =  LAQUOT absoluteURI RAQUOT *( SEMI info-param)
info-param  =  ( "purpose" EQUAL ( "icon" / "info"
               / "card" / token)) / generic-param
```

This shows that the *Call-Info* header allows for comma-delimited information elements. Each element must have an absolute URI, and each element may optionally have a sequence of parameters delimited by semicolons.

Cisco BroadWorks extends the syntax of the *Call-Info* header. The ABNF for the extended syntax is as follows.

```
info-param /= appearance-index-param / appearance-state-param
    / appearance-uri-param / answer-after-param / silent-monitor-param
    / client-session-info-param / conference-subscription-uri-param

appearance-index-param = "appearance-index" EQUAL ( (1*DIGIT) / STAR ) )

appearance-state-param = "appearance-state" EQUAL
    ( "idle" / "seized" / "progressing" / "alerting" / "active"
      / "held" / "held-private" / "bridge-active" / "bridge-held" )

appearance-uri-param  = "appearance-uri" EQUAL quoted-string

answer-after-param = "answer-after" EQUAL 1*DIGIT

silent-monitor-param = "silent-monitor"

client-session-info-param = "client-session-info" EQUAL quoted-string

conference-subscription-uri-param = "conference-subscription-uri"
    EQUAL quoted-string
```

This extended syntax matches the production for *generic-param* in the *RFC 3261* syntax; therefore, Cisco BroadWorks' extended syntax is fully compatible with the *RFC 3261* syntax.

NOTE: In the *Call-Info* header, the extended syntax parameters may appear in arbitrary order. Cisco does not guarantee the order of the parameters.

When used with Cisco BroadWorks access-side extensions, the *Call-Info* header's absolute URI should be a simple SIP URI consisting of a hostname with no user portion. The host name should be set to the provisioned proxy server or in this case, the Application Server.

For usability reasons, it is important to preserve the relative order of call appearances across endpoints. The access-side extensions use the *appearance-index* parameter to qualify the call information elements, by identifying a relative index of the call appearances on the line. Therefore, if the *appearance-index* is "1", it indicates the first appearance on the line. If the *appearance-index* is "2", it indicates the second appearance on the line, and so on, up to the maximum number of appearances allowed on the line. As a special case, if the *appearance-index* is "*", then it indicates that the information element is applicable to the rest of the call appearances on the line.

The access-side extensions use the *appearance-state* parameter to further qualify the call appearance with an actual visual state that can be used to light a lamp, or to display a light-emitting diode (LED) or bitmap on a liquid crystal display (LCD). The following table provides the defined states.

State	Description
idle	This appearance on the line is available for use.
seized	This appearance has been seized by one of the endpoints in the SCA group.
progressing	This appearance on the line is currently making an outgoing call.
alerting	This appearance is receiving an incoming call.
active	This appearance is actively involved in a call.
held	This appearance has a call in the *held* state.
held-private	This appearance has a call in the *held* state and only the endpoint that held the call, may retrieve it.
bridge-active	This appearance has an SCA bridge active and at least one location is active and talking with the remote party.
bridge-held	This appearance has an SCA bridge active with all locations in the *held* state and the remote party is held.

The *appearance-uri* parameter further qualifies the call appearance with the remote party identification, which devices can use to augment the display. The value of the *appearance-uri* parameter is a SIP quoted-string that matches the name-addr syntax element in *RFC 3261*. Since the name-addr is within a quoted-string, it is escaped according to the SIP escaping rules. The *appearance-uri* parameter is omitted when the call appearance state is *idle* or *seized*, since there is no call involved with the call appearance in these states.

The *answer-after* parameter appears in an initial INVITE request and indicates to the User Agent Server (UAS) that it should automatically answer the new incoming call. The parameter's value is an integer, which the UAS should interpret as the number of ring cycles to allow before automatically answering the call. A value of "0" is useful for applications such as Push-To-Talk and Click-To-Dial.

The *silent-monitor* parameter appears in a barge-in initial INVITE request from the device and indicates that Cisco BroadWorks should interpret the INVITE request as a request for silent monitoring.

The *client-session-info* parameter provides application-level information for a client application. The Application Server does not interpret this client session information, but can receive it from a client or send it to a client.

The *conference-subscription-uri* parameter contains a conference subscription URI that the Application Server sends to the access device. If the access device supports the conference event package (*RFC 4975*), it can subscribe to the events at this URI.

3.2 Call-Info Header Usage

3.2.1 In INVITE Requests

A User Agent Client (UAC) may include a *Call-Info* header when sending an initial INVITE request. If it does, it must contain at most one call appearance information element. It may contain other information elements for different purposes, but only one call appearance information element should be present. The call appearance information element must contain exactly one *appearance-index* parameter with a numeric value. The numeric value indicates the call appearance index where the call is being presented.

So, for example, assume a phone has one line with four separate call appearances, and the user interface on the phone allows the end user to arbitrarily select any one of the call appearances when originating a call. If the user chose the third call appearance, then the initial INVITE request can look similar to the following.

```
INVITE sip:5551212@broadworks.net …
From:  <sip:5551000@broadworks.net>…
To: <sip:5551212@broadworks.net>…
Call-Info:  <sip:broadworks.net>;appearance-index=3
…
```

Effectively the phone is saying: “The user has selected the third call appearance of line 555-1000 to call 555-1212”.

The *Call-Info* header is not required to be present in a re-INVITE request, but if present, the re-INVITE request must use the same appearance-index (that is, a re-INVITE request is not allowed to change the appearance-index in use for the existing call).

When Cisco BroadWorks sends an initial INVITE request, the device must honor the appearance-index included in that INVITE request. If the device initiates a line-seize for an appearance-index, and at the same time Cisco BroadWorks sends an initial INVITE request for the same appearance-index, the device should allow the INVITE request for the incoming call to proceed. When such a race condition occurs, the device can expect Cisco BroadWorks to send a failure response to the line-seize SUBSCRIBE request shortly afterwards.

When the device sends an initial INVITE request that includes an appearance-index, Cisco BroadWorks may choose to override the appearance-index in the response, as described in section *3.2.2 In Responses.*

To designate a call as privately held, a device must send a re-INVITE request with hold Session Description Protocol (SDP) and an appearance-state of *held-private*. Cisco BroadWorks, upon receipt of the re-INVITE request for hold, inspects the *Call-Info* header and updates the state for the call appearance to *held-private*. All shared call appearances with subscriptions to the call-info package in the group are updated via NOTIFYs with the *held-private* call appearance state.

Following is an example of the re-INVITE request with hold SDP and an appearance-state of *held-private*.

```
INVITE sip:5551212@broadworks.net …
From:  <sip:5551000@broadworks.net>…
To: <sip:5551212@broadworks.net>…
Call-Info:  <sip:broadworks.net>;appearance-state=held-private
…
```

Following is the resulting NOTIFYs sent to the shared call appearances in the group.

```
NOTIFY sip:contact@endpoint …
From:  <sip:5551000@broadworks.net>…
To: <sip:5551000@broadworks.net>…
Call-Info:  <sip:broadworks.net>;appearance-index=1;appearance-
state=held-private;appearance-index=*;appearance-state=idle
```

When an endpoint in the shared call appearance group attempts to retrieve a privately-held call appearance which it did not put on hold, Cisco BroadWorks rejects the re-INVITE retrieval attempt with a *403 Forbidden*.

Cisco recommends that devices use the same *lamp* state for held-private calls that they use for active calls. The recommended *active lamp* state for all of the other endpoints which are not actively involved in the call is a solid red indicator. However, the *held-private* state is propagated to all of the endpoints in the shared call appearance group, such that an endpoint may choose a different visual indication for privately-held calls.

3.2.2 In Responses

Similarly, a UAS may include a *Call-Info* header when responding to an INVITE request. A *Call-Info* header with call appearance information elements may appear in any provisional or final response. They follow similar rules for "Call-Info" sent in INVITE requests by UACs. So in the above example, assume the endpoint that receives the call for the address of record 5551212@broadworks.net allows up to six call appearances. At the time that this INVITE arrives at the endpoint, assume there are three calls present on the first three call appearances. So the phone chooses the next available call appearance (4) and sends back a provisional response.

```
SIP/2.0 180 Ringing …
From:  <sip:5551000@broadworks.net>…
To: <sip:5551212@broadworks.net>…
Call-Info:  <sip:broadworks.net>;appearance-index=4
```

Effectively the phone is saying: "The incoming call from 555-1000 is ringing the user on the fourth call appearance of the line 555-1212".

Once the call is answered, then the *200 OK* response from the UAS may also contain a *Call-Info* header. It is possible that the call may have moved relative appearances by the time it is answered. This is totally dependent on the user interface of the endpoint.

```
SIP/2.0 200 OK …
From:  <sip:5551000@broadworks.net>…
To: <sip:5551212@broadworks.net>…
Call-Info:  <sip:broadworks.net>;appearance-index=3
```

Effectively the phone is saying: "The incoming call from 555-1000 has been answered by the user on the third call appearance of line 555-1212".

If the user interface on the endpoint does not choose a particular call appearance until it answered, then it is possible that the *180 Ringing* response does not have a *Call-Info* header and the *200 OK* response has a *Call-Info* header.

Note that the *Call-Info* header is not required to be present in responses to a re-INVITE request, but if present, the responses must use the same appearance-index (that is, a re-INVITE response is not allowed to change the appearance-index in use for the existing call).

When Cisco BroadWorks is acting as the UAS, the device must honor the appearance-index included in INVITE responses. In some scenarios (for example, when the device does not perform a line-seize prior to the INVITE and the requested appearance-index is already in use by Cisco BroadWorks), Cisco BroadWorks overrides the requested appearance-index when it responds to the INVITE request. In these scenarios, the device should move the call to the appearance-index specified in the response.

When the device is acting as the UAS and it receives an INVITE with a requested appearance-index, it must use the same appearance-index for all responses (that is, it is not allowed to change the appearance-index in a response).

Finally, a notifier can include a *Call-Info* header when sending a NOTIFY request to a subscriber of the *call-info* event package (description follows). The *Call-Info* header of a NOTIFY request must contain a complete set of call appearance information elements – one for each potential call appearance on the address of record. Each call appearance information element must contain one *appearance-index* and one *appearance-state* parameter. Optionally, each call appearance information element can contain one *appearance-uri* parameter. Therefore, the subscriber receives complete call appearance state information for the address of record in each NOTIFY.

For example, if the address of record sip:5551000@broadworks.net has four call appearances allowed, and there is one call active on the first call appearance and one call held on the second call appearance, and the rest of the call appearances are idle, then the following NOTIFY is sent to each subscriber of sip:5551000@broadworks.net.

```
NOTIFY sip:contact@endpoint …
From:  <sip:5551000@broadworks.net>…
To: <sip:5551000@broadworks.net>…
Call-Info:  <sip:broadworks.net>;appearance-index=1;appearance-
state=active;appearance-uri="\"George Smith\"
<sip:3015551234@broadworks.net>",
            <sip:broadworks.net>;appearance-index=2;appearance-
state=held,
            <sip:broadworks.net>;appearance-index=3;appearance-
state=idle,
            <sip:broadworks.net>;appearance-index=4;appearance-state=idle
```

The "*" can be used as a short-cut to indicate "the rest of the call appearances", which is useful when initializing the call appearances for an address of record to "idle".

```
NOTIFY sip:contact@endpoint …
From:  <sip:5551000@broadworks.net>…
To: <sip:5551000@broadworks.net>…
Call-Info:  <sip:broadworks.net>;appearance-index=*;appearance-state=idle
```

… is semantically equivalent to:

```
NOTIFY sip:contact@endpoint …
From:  <sip:5551000@broadworks.net>…
To: <sip:5551000@broadworks.net>…
```

```
Call-Info:  <sip:broadworks.net>;appearance-index=1;appearance-
state=idle,
            <sip:broadworks.net>;appearance-index=2;appearance-
state=idle,
            <sip:broadworks.net>;appearance-index=3;appearance-
state=idle,
            <sip:broadworks.net>;appearance-index=4;appearance-state=idle
```

3.2.3 In SUBSCRIBE Requests

A subscriber must include a *Call-Info* header with an appearance-index information element in a SUBSCRIBE request to the line-seize event package. The Application Server handling the subscription uses the appearance-index in the *Call-Info* header to identify which call appearance the line-seize subscription is being applied to. The line-seize event package is described in section *5 Line-Seize Event Package*.

3.2.4 Extensions for Shared Call Appearance Bridging

To allow different locations associated with a user to barge-in to an active call involving another location, two appearance-states are defined with the Shared Call Appearance service:

- bridge-active
- bridge-held

This bridging capability, in conjunction with the existing multiple call arrangement capability, allows Cisco BroadWorks to offer a complete-hosted key-system solution. Furthermore, this capability also enhances Cisco BroadWorks executive/administrator solution to fill a functional gap that exists with some PBX executive/administrator solutions where the administrator is able to barge on to the executive calls.

- **bridge-active**: Indicates that an SCA-Bridge is active on a particular call appearance. At least one location is active and talking with the remote party.
- **bridge-held**: Indicates that an SCA-Bridge is held and that the remote party is held. An SCA-Bridge is held if all locations are holding their respective call leg to the bridge. An SCA-Bridge that is in the *bridge-held* state cannot be retrieved by a non-participating location. An INVITE request from a non-participating location for a call appearance in the *bridge-held* state is always interpreted as a request to join the bridge.

With the call-info event package, the call appearance state is sent in a NOTIFY request to all locations, whether or not they are involved in the call. The call appearance state is sent in the following scenarios for Shared Call Appearance Bridging:

- SCA-Bridge creation (bridge-active).
- SCA-Bridge is held, that is, the remote party is held (bridge-held). The remote party is held when all locations are held or when the call client is used to hold the call to the remote party.
- SCA-Bridge is retrieved, that is, the remote party is retrieved (bridge-active). The remote party is retrieved when one held location retrieves its respective call leg or when the call client is used to retrieve the call to the remote party.
- SCA-Bridge is released with a remaining held location (held).
- SCA-Bridge is released with a remaining active location (active).

3.2.4.1 Functional Requirements for Device Support of Shared Call Appearance Bridging

To properly support the bridging capability facilitated by these extensions, the device must support the following functional requirements:

- The user must be able to select an active call appearance among all current call appearances on a particular line.
- Upon selecting the active call appearance, the user must be presented with the option to barge-in to an active call appearance.

The following section describes how the barge-in option may be presented to the user. The diagrams are included for illustrative purposes only. It is the responsibility of the device vendor to determine the most appropriate user interface.

3.2.4.1.1 Barge-In Operation

A user can retrieve a held-call appearance by selecting the call appearance and pressing a feature key (for example, “retrieve”) on the SIP phone. If the user presses the “retrieve” feature key, then the device sends an INVITE request with a *Call-Info* header that refers to the held-call appearance index. The Application Server interprets the INVITE request as a request to retrieve the held-call appearance and proceeds as such.

A user can barge-in to an active call appearance by selecting the call appearance and pressing a feature key or other invocation method on the device (for example, “barge-in”). If the user presses the “barge-in” feature key, then the device sends an INVITE request with a *Call-Info* header that refers to the active call appearance index. The Application Server interprets the INVITE request as a request to barge-in to the active call appearance and proceeds as such.

If the device supports silent monitoring, then it may add a *silent-monitor* parameter to the *Call-Info* header entry to request silent monitoring from Cisco BroadWorks. Silent monitoring is a special form of SCA bridging in which the barge-in party can listen to the other parties but cannot be heard by them.

The following diagrams demonstrate how a device might present call appearances to a user and allow that user to either retrieve a held-call appearance or barge-in to an active call appearance. The device demonstrated in the diagrams is a SIP phone with four lines, each of which is able to handle multiple call appearances. The line keys are used to select a line and display current call appearances for that line. Each line key has an associated lamp that indicates activity for that line.

The diagrams are shown for illustrative purposes only and different vendors may implement different user interfaces to activate the “barge-in” and “retrieve” functions.

Also, notice that the INVITE request for the retrieve and barge-in scenarios are identical. The Application Server distinguishes between a retrieve and barge-in request based on the call appearance state, that is *held* or *active*. If two users attempt to retrieve a call appearance at the same time, then the first request is processed as a retrieve request, but the second is processed as a barge-in request. Similarly, if a user attempts to barge-in to a call appearance at the same time the other user holds the call appearance, then the barge-in request is processed as a retrieve request if it gets processed immediately after the hold request.

Barge-In Option (New)

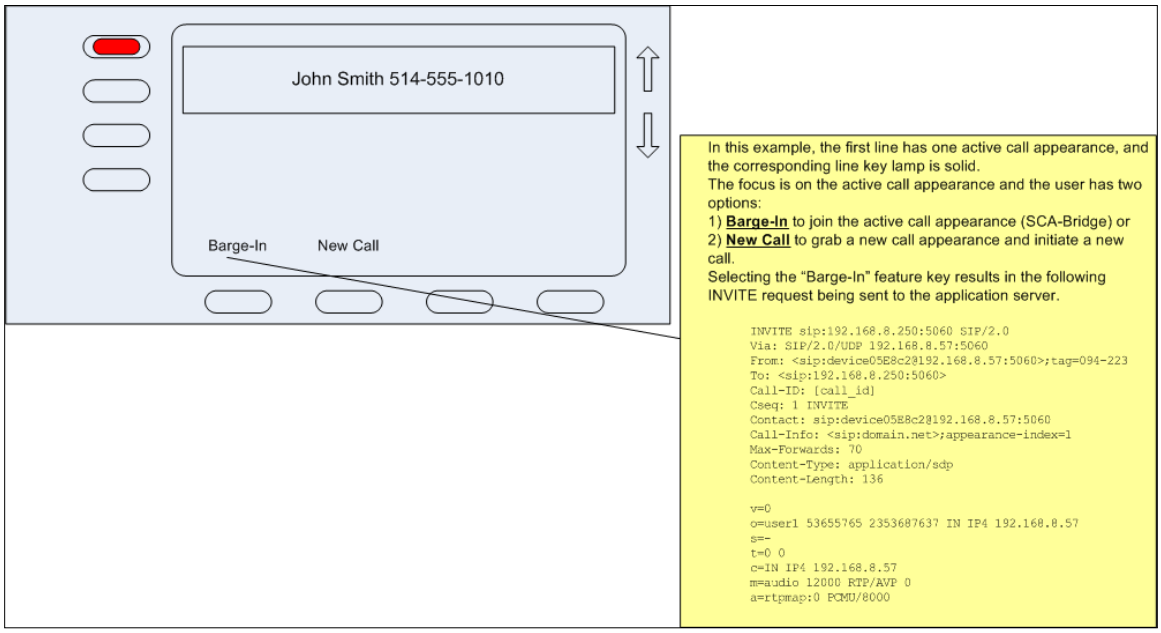


Figure 1 Barge-In Option (New)

Retrieve Function (Existing)

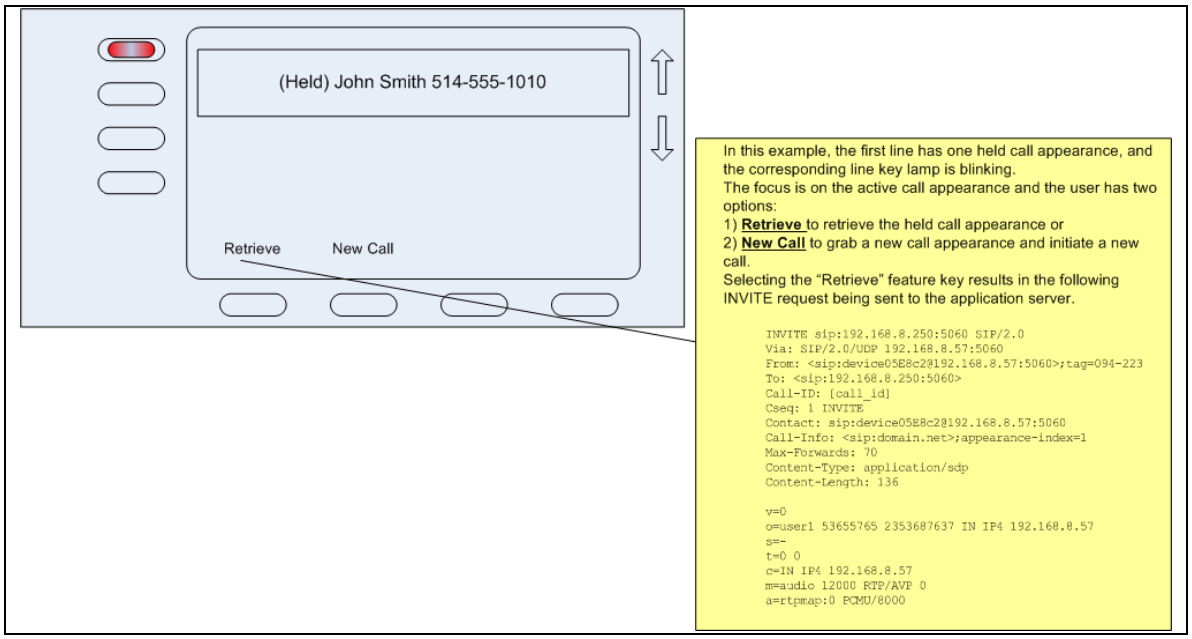


Figure 2 Retrieve Function (Existing)

Multiple Call Appearances

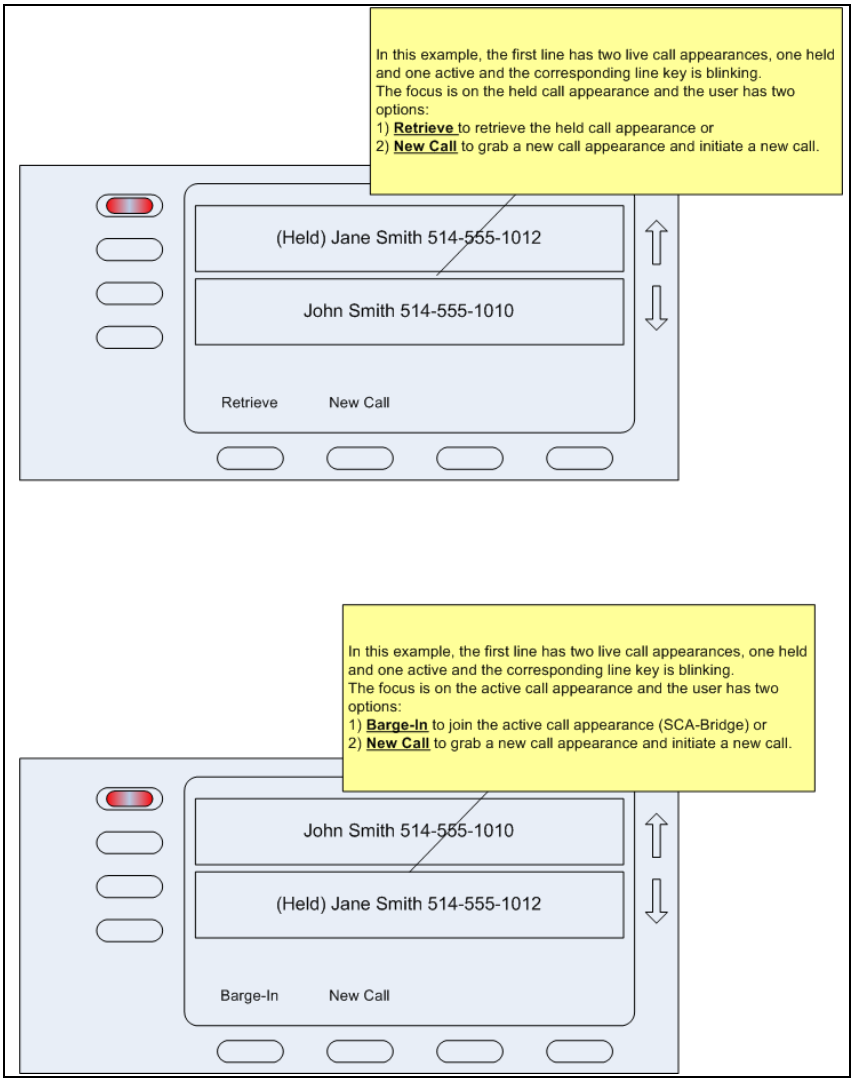


Figure 3 Multiple Call Appearances

3.2.4.2 Signaling Requirements for Device Support of Shared Call Appearance Bridging

If the user activates the barge-in option on an active call appearance, then the device must send an INVITE request to the Application Server. The INVITE request is encoded as follows:

1) The Request-URI corresponds to the contact received from the Application Server for the active call appearance.
2) The *Call-Info* header is included and the *appearance-index* parameter is set to the active call appearance being barged-in.
3) The message body may or may not include an offer SDP.

The Application Server sends a *200* response if the SCA-Bridging is enabled on the user profile and a *403* response if it is not enabled. The Application Server sends a *480* response if a conference resource cannot be allocated for the SCA-Bridge.

Once the SCA-Bridge is activated, the Application Server sends a NOTIFY request to all locations, including the non-participating locations. The NOTIFY request for the Call-Info event package contains the *Call-Info* header that indicates the index and state of the call appearance in the *appearance-index* and *appearance-state* parameters, respectively.

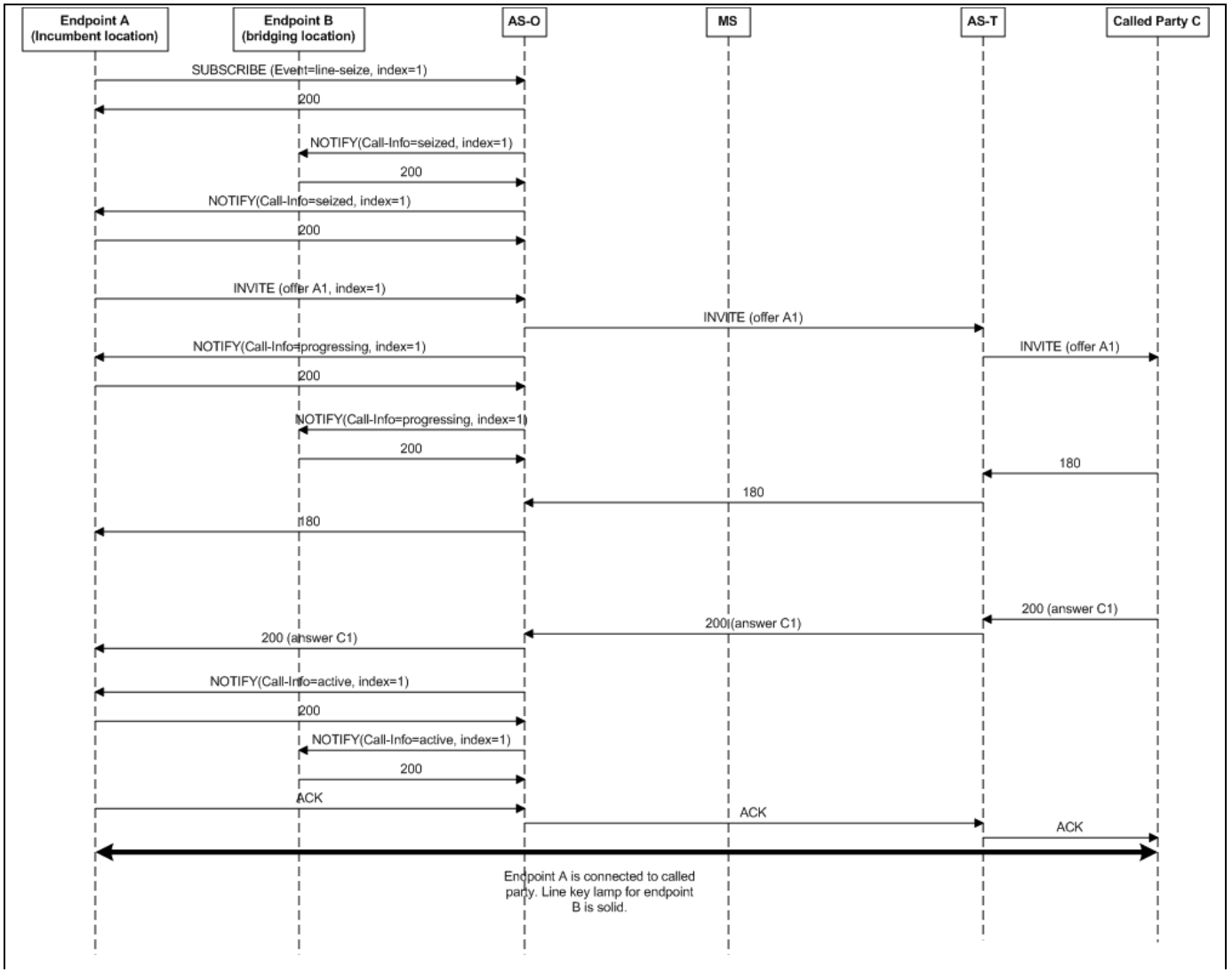
3.2.4.3 Shared Call Appearance Bridging Call Flows

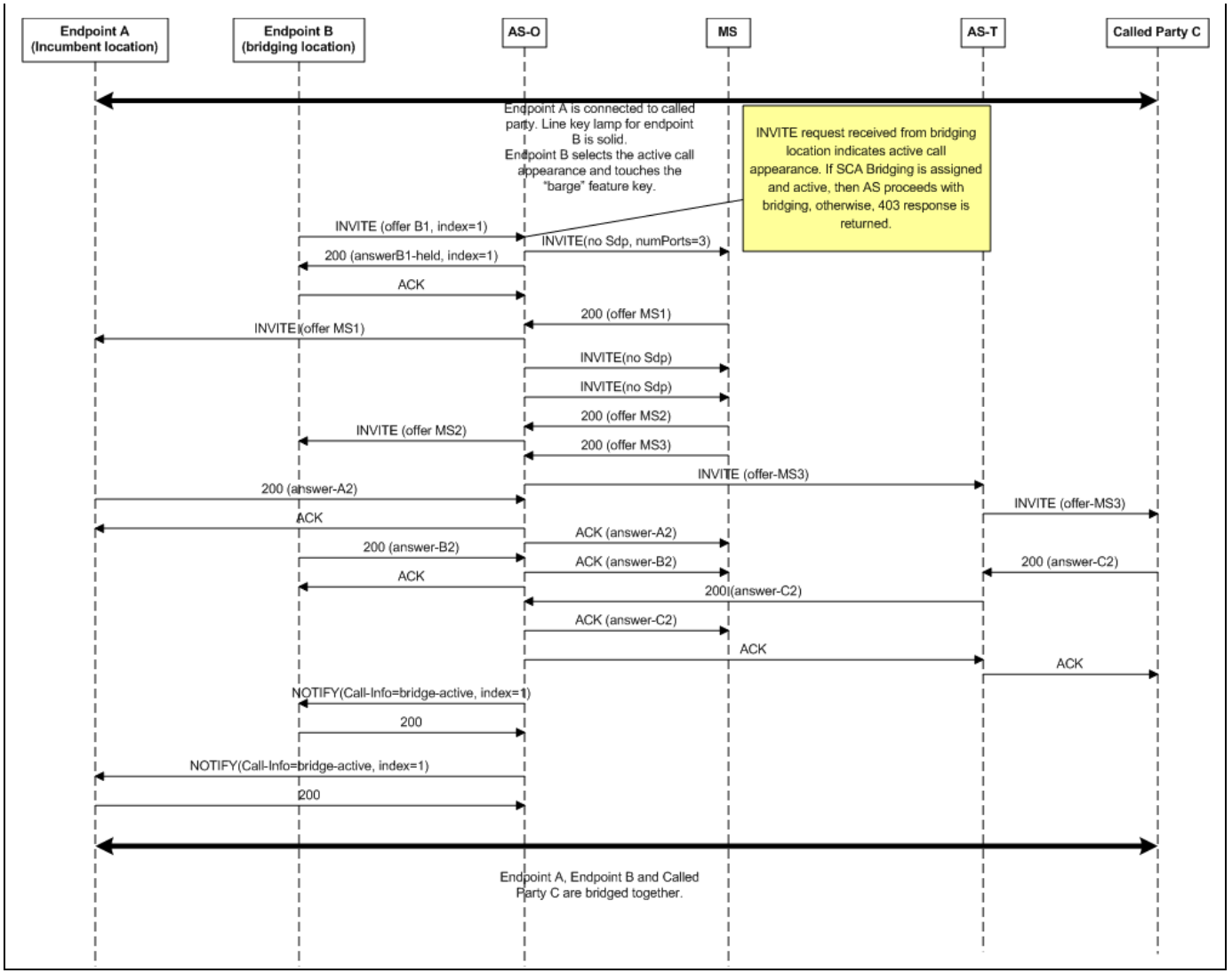
The following call flows demonstrate how an SCA-Bridge is created if a location sends an INVITE request with an active appearance index.

In this section, the following terms are used to designate the locations involved in an SCA-Bridge.

- Incumbent location: This is typically the location that is linked to the call at the time the SCA-Bridge is created. This is typically the location that answered or originated the call and does not necessarily correspond to the primary shared call appearance endpoint.
- Bridging location: This is the location that barges in to the established call.

There is always one incumbent location, but there can be many bridging locations.





3.3 Answer-After Parameter Usage

3.3.1 Overview

To support Click-To-Dial and Push-To-Talk applications, an application platform must be able to originate calls that force the endpoint off-hook, without end-user intervention.

This section describes the use of the *answer-after* parameter. When present in the *Call-Info* header of an incoming INVITE request, it indicates how many alert cycles should be applied by the UAS before the call is automatically answered. For example, the following *Call-Info* header indicates that the call should be automatically answered after one alert cycle.

```
INVITE sip:123456789@broadworks.net  SIP/2.0
From: <sip:jamie@broadworks.net>; tag=1
To: <sip:foo@broadworks.net>
Call-Info: <sip:broadworks.net>; answer-after=1
```

If the *answer-after* value is “0”, then the call should be automatically answered without applying any alert tones, as shown in the following example.

```
INVITE sip:123456789@broadworks.net  SIP/2.0
From: <sip:jamie@broadworks.net>; tag=1
To: <sip:foo@broadworks.net>
Call-Info: <sip:broadworks.net>; answer-after=0
```

In all cases, the *Call-Info* header is just a suggestion to the UAS. Depending on the UAS capabilities or local policy, the UAS may elect to ignore the *answer-after* parameter. For instance, if the UAS is a phone that does not support a speakerphone, then it cannot automatically answer the call, and must wait for the user to actively answer the call. Similarly, the phone may have a speakerphone, but the user may have enabled a local privacy policy that rejects or ignores all “answer-after” requests. A UAC should be prepared to receive an “18x provisional” response, even when it has requested that the UAS automatically answer the call without alerting (meaning answer-after = “0”).

In all cases, the UAS should attempt to authorize the incoming request. At minimum, the UAS should perform access control. Ideally, the UAS should challenge the incoming INVITE for credentials that authorize the request to perform an automatic answer.

3.3.2 Example Message Flow

The following diagram shows the message flow as a result of the users performing a Click-To-Dial operation from their thin client interface.

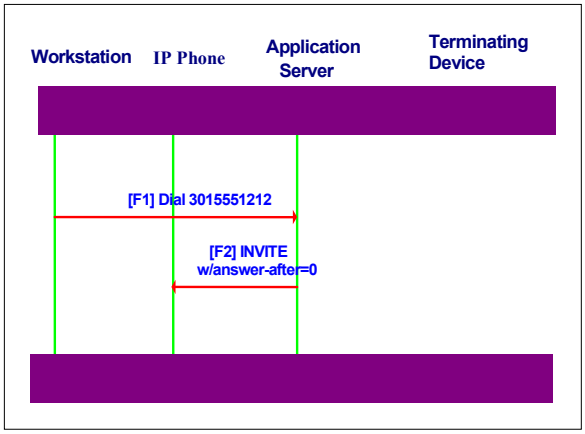


Figure 4 Message Flow as a Result of Click-To-Dial Operation

[F1] starts the flow with the user initiating a Click-To-Dial operation from the thin call client on the workstation. The Application Server processes the Click-To-Dial event. The Application Server sends an INVITE to the associated endpoint in [F2] as follows.

```
[F2] INVITE Application Server -> IP phone
INVITE sip:3015551000@ipphone.com SIP/2.0
Via: SIP/2.0/UDP a1.foo.com:5060;branch=m9hG4bK74bf9
Max-Forwards: 70
From: "Joe" <sip:3015551212@a1.foo.com>;tag=5fxced76sl
To: "Jane" <sip:3015551000@a1.foo.com>
Call-ID: 98482762982201885l1@a1.foo.com
Call-Info:  <sip:as.foo.com>;appearance-index=1; answer-after=0
CSeq: 43234 INVITE
Contact: <sip:a1.foo.com:5060>
Allow-Events: conference
Allow: ACK, BYE, CANCEL, INFO, INVITE, OPTIONS, PRACK, REFER
Supported:  100rel, timer
Content-Length: 0
```

4 Call-Info Event Package

4.1 Overview

The ability to support SCA is a fundamental building block for a variety of enhanced telephony services. Features like Attendant Console, Line Extensions, and Key System Emulation cannot be delivered without some mechanism for sharing call appearances across access devices. Although SIP (*RFC 3261* [2]) by itself offers no inherent semantics for supporting SCA features, when coupled with an appropriate instantiation of the "SIP Specific Event Notification" framework (*RFC 6665* [3]), these services can be deployed quite easily in a distributed network.

A shared line is an address of record managed by a central controlling element, such as an Application Server. The Application Server allows multiple endpoints to register locations against the address of record. The Application Server is responsible for policing who can register and who cannot register against the shared line.

For incoming call appearances, the Application Server sends separate INVITE requests to each of the registered endpoints. As the incoming call appearance progresses to the *answered* state on one of the endpoints, the INVITE requests to the other endpoints in the shared call appearance group are cancelled. For outgoing call appearances, the Application Server presents any call from the registered endpoints as originating from the same address of record. Effectively, the line is shared because any endpoint in the shared call appearance group can originate and terminate calls on behalf of the same address of record.

The following sections describe how the *Call-Info* header can be used to suggest the presentation state of the call appearances on a shared line. A Call-Info event package can then be used to broadcast this presentation state to endpoints that are not actively involved in the call appearances.

The Cisco BroadWorks Shared Call Appearance service can also be optionally configured to provide Call Park information similarly to the Call Park Event Package (section *12 Call Park Event Package*) but without the requirement for a separate subscription.

The Call-Info event package is an instantiation of the SIP Specific Event Notification Framework (as defined in *RFC 6665* [3]). For the applications described earlier, the Application Server acts as the "notifier", while the IP phones configured with shared lines act as the "subscribers".

Following is a description of the event package details, as per *RFC 6665* [3].

4.2 Example Message Flow

4.2.1 Shared Call Appearance Client Subscription

This process allows the endpoint to SUBSCRIBE to the call appearance state of a given line. The *Request-URI* for these transactions is the same as the *Request-URI* used in the registration process for the address of record (in this case, a "line").

Note that the call flow does not show the SUBSCRIBE being challenged. The Application Server may choose to use access control (using the IP address authenticated in the REGISTER transaction), or optionally it could challenge the transaction. If the transaction is challenged, then the IP phone should use the same credentials it used in the corresponding registration process for that call appearance. Throughout the rest of this example, it is assumed that standard access control is being applied to all transactions.

After accepting the SUBSCRIBE, the Application Server initializes the phone to the appropriate call status by sending a NOTIFY with a call state document. This NOTIFY event is sent even if the call state is *idle*, and serves as a synchronization event between the Application Server and the IP phone.

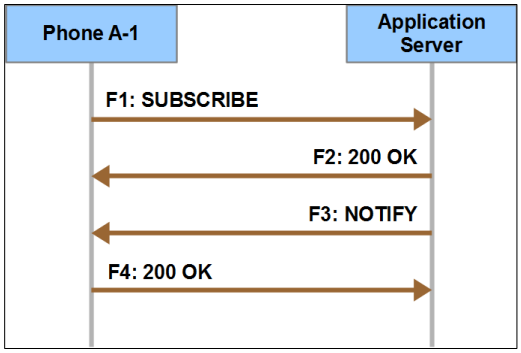


Figure 5 Call Flow for Initial Call-Info Subscription

```
[F1] SUBSCRIBE A-1 -> Application Server
SUBSCRIBE sip:shared-a@as.foo.com SIP/2.0
Via: SIP/2.0/UDP a1.foo.com:5060;branch=dsdf32nashds7
Max-Forwards: 70
From: "Shared-A" <sip:shared-a@as.foo.com>;tag=dsa3dszlfl
To: "Shared-A" <sip:shared-a@as.foo.com>
Call-ID: 5j9FpLxk3uxtm8to@a1.foo.com
CSeq: 6 SUBSCRIBE
Event: call-info
Expires: 3600
Contact: <sip:shared-a@a1.foo.com>
Content-Length: 0


[F2] 200 OK Application Server → A-1
SIP/2.0 200 OK
Via: SIP/2.0/UDP a1.foo.com:5060;branch=dsdf32nashds7
From: "Shared-A" <sip:shared-a@as.foo.com>;tag=dsa3dszlfl
To: "Shared-A" <sip:shared-a@as.foo.com>;tag=323423233
Call-ID: 5j9FpLxk3uxtm8tn@a1.foo.com
CSeq: 6 SUBSCRIBE
Event:  call-info
Contact: sip:shared-a@as.foo.com
Expires: 3600
Content-Length: 0


[F3] NOTIFY Application Server -> A-1
NOTIFY sip:shared-a@a1.foo.com SIP/2.0
Via: SIP/2.0/UDP as.foo.com:5060;branch=gsdf32nashds7
Max-Forwards: 70
From: "Shared-A" <sip:shared-a@as.foo.com>;tag=dsa3dszlfl
To: "Shared-A" <sip:shared-a@as.foo.com>;tag=323423233
Call-ID: 5j9FpLxk3uxtm8tn@a1.foo.com
CSeq: 1344 NOTIFY
Event: call-info
Subscription-State: active; expires=3599
Contact:  sip:shared-a@as.foo.com
Call-Info:  <sip:as.foo.com>;appearance-index=*;appearance-state=idle
Content-Length: 0
```

```
[F4] 200 OK A-1 → Application Server
SIP/2.0 200 OK
Via: SIP/2.0/UDP as.foo.com:5060;branch=gsdf32nashds7
From: "Shared-A" <sip:shared-a@as.foo.com>;tag=dsa3dszlfl
To: "Shared-A" <sip:shared-a@as.foo.com>;tag=323423233
Call-ID: 5j9FpLxk3uxtm8tn@a1.foo.com
CSeq: 1344 NOTIFY
Content-Length: 0
```

4.2.2 Outgoing Call

The following figure shows the call flow as a result of endpoint A-1 originating a call to B using A's shared line. It is assumed that A-1 has successfully seized the line.

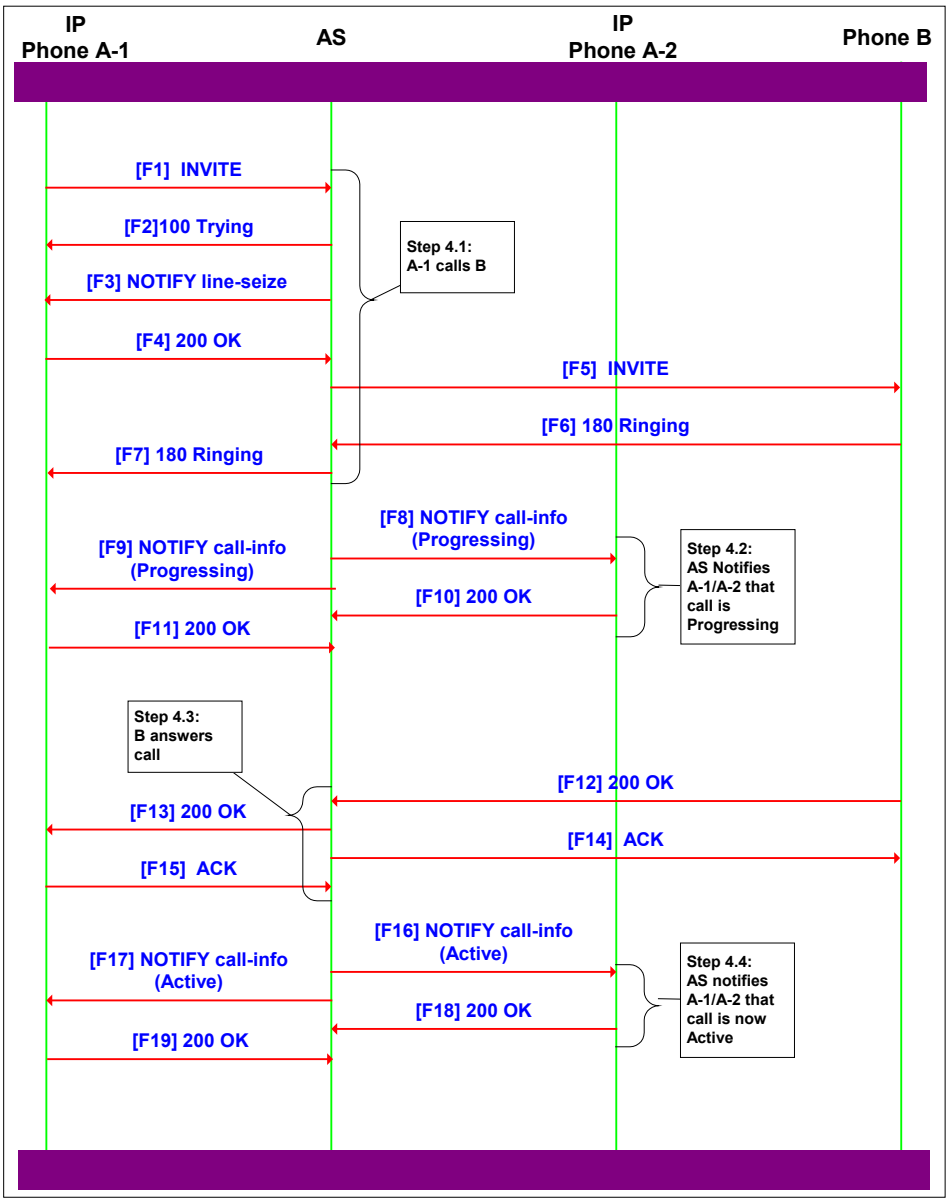


Figure 6 Call Flow as Result of Endpoint A-1 Originating Call to B using A's Shared Line

4.2.2.1 A-1 Calls B

In [F1 to F7], the user on IP phone A-1 has entered digits that originate a call to B. Because the endpoint has already seized the line, the Application Server accepts the INVITE and simply treats the call as a standard outbound call to an off-network device. Note that in [F3], the Application Server sends a NOTIFY-request to A-1, indicating that the line-seize subscription has been terminated. This is normal once an INVITE has been received from the endpoint that has seized the call appearance on the shared line.

```
[F1] INVITE A-1 → Application Server

INVITE sip:B@as.foo.com SIP/2.0
Via: SIP/2.0/UDP a1.foo.com:5060;branch=m9hG4bK74bf9
Max-Forwards: 70
From: "Shared-A" <sip:shared-a@as.foo.com>;tag=5fxced76sl
To: "B" <sip:B@as.foo.com>
Call-ID: 9848276298220188511@a1.foo.com
Call-Info:  <sip:as.foo.com>;appearance-index=1
CSeq: 43234 INVITE
Contact: <sip:shared-a@a1.foo.com>
Content-Type: application/sdp
Content-Length: 143

v=0
o=UserA 2890844526 2890844526 IN IP4 a1.foo.com
s=-
c=IN IP4 192.0.2.101
t=0 0
m=audio 49172 RTP/AVP 0
a=rtpmap:0 PCMU/8000

[F2] 100 Trying Application Server → A-1

SIP/2.0 100 Trying
Via: SIP/2.0/UDP a1.foo.com:5060;branch=m9hG4bK74bf9
From: "Shared-A" <sip:shared-a@as.foo.com>;tag=5fxced76sl
To: "B" <sip:B@foo.com>; tag=4323z39
Call-ID: 9848276298220188511@a1.foo.com
Call-Info:  <sip:as.foo.com>;appearance-index=1
CSeq: 43234 INVITE
Contact: <sip:shared-a@as.foo.com>
Content-Length: 0

[F3] NOTIFY Application Server -> A-1
NOTIFY sip:shared-a@a1.foo.com SIP/2.0
Via: SIP/2.0/UDP as.foo.com:5060; branch=gsdf32nashds7
Max-Forwards: 70
From: "Shared-A" <sip:shared-a@as.foo.com>;tag=gsa3dszlfl
To: "Shared-A" <sip:shared-a@as.foo.com>;tag=423423233
Call-ID: 6j9FpLxk3uxtm8to@a1.foo.com
CSeq: 24 NOTIFY
Event: line-seize
Subscription-State: terminated; expires=0
Contact: sip:shared-a@as.foo.com
Content-Length: 0

[F4] 200 OK A-1 → Application Server
SIP/2.0 200 OK
Via: SIP/2.0/UDP a1.foo.com:5060;branch=gsdf32nashds7
From: "Shared-A" <sip:shared-a@as.foo.com>;tag=gsa3dszlfl
To: "Shared-A" <sip:shared-a@as.foo.com>;tag=423423233
Call-ID: 6j9FpLxk3uxtm8tn@a1.foo.com
```

```
CSeq: 24 NOTIFY
Content-Length: 0


[F5] INVITE Application Server → B

INVITE sip:B@b.foo.com SIP/2.0
Via: SIP/2.0/UDP as.foo.com:5060;branch=234adfrjksd
Max-Forwards: 70
From: "Shared-A" <sip:shared-a@as.foo.com>;tag=234wdkfj
To: "B" <sip:B@as.foo.com>
Call-ID: asdf2220188511@as.foo.com
CSeq: 12345 INVITE
Contact: <sip:shared-a@as.foo.com>
Content-Type: application/sdp
Content-Length: 143

v=0
o=UserA 2890844526 2890844526 IN IP4 a1.foo.com
s=-
c=IN IP4 192.0.2.101
t=0 0
m=audio 49172 RTP/AVP 0
a=rtpmap:0 PCMU/8000

[F6] 180 Ringing B → Application Server

SIP/2.0 180 Ringing
Via: SIP/2.0/UDP as.foo.com:5060;branch=234adfrjksd
From: "Shared-A" <sip:shared-a@as.foo.com>;tag=234wdkfj
To: "B" <sip:B@as.foo.com>;tag=1234dsf2
Call-ID: asdf2220188511@as.foo.com
CSeq: 12345 INVITE
Contact: <sip:B@b.foo.com>
Content-Length: 0

[F7] 180 Ringing Application Server → A-1

SIP/2.0 180 Ringing
Via: SIP/2.0/UDP a1.foo.com:5060;branch=m9hG4bK74bf9
From: "Shared-A" <sip:shared-a@as.foo.com>;tag=5fxced76sl
To: "B" <sip:B@as.foo.com>;tag=4323z39
Call-ID: 9848276298220188511@a1.foo.com
Call-Info:  <sip:as.foo.com>;appearance-index=1
CSeq: 43234 INVITE
Contact: <sip:B@as.foo.com>
Content-Length: 0
```

4.2.2.2 Application Server Notifies A-1 and A-2 that Call is Progressing

As the call progresses, the Application Server sends Call-Info NOTIFY-requests to all endpoints that have subscribed to the Call-Info event package on the shared line. In this example, IP phones A-1 and A-2 have both subscribed to the Call-Info event package, so the Application Server sends a NOTIFY-request to both A-1 and A-2, with a *Call-Info* header indicating a state of *Progressing* (events [F8-F11]).

```
[F8] NOTIFY Application Server -> A-1
NOTIFY sip:shared-a@a1.foo.com SIP/2.0
Via: SIP/2.0/UDP as.foo.com:5060;branch=gsdf32nashds7
Max-Forwards: 70
From: "Shared-A" <sip:shared-a@as.foo.com>;tag=dsa3dszlfl
To: "Shared-A" <sip:shared-a@as.foo.com>;tag=323423233
Call-ID: 5j9FpLxk3uxtm8tn@a1.foo.com
CSeq: 1344 NOTIFY
Event: call-info
Subscription-State: active; expires=3200
Call-Info:  <sip:as.foo.com>;appearance-index=1;appearance-
state=progressing,
            <sip:as.foo.com>;appearance-index=*;appearance-state=idle
Content-Length: 0

[F9] NOTIFY Application Server -> A-2
NOTIFY sip:shared-a@a2.foo.com SIP/2.0
Via: SIP/2.0/UDP as.foo.com:5060;branch=gsdf32nashds7
Max-Forwards: 70
From: "Shared-A" <sip:shared-a@as.foo.com>;tag=ggsa3dszlfl
To: "Shared-A" <sip:shared-a@as.foo.com>;tag=123456
Call-ID: 9k9FpLxk3uxtm8tn@a2.foo.com
CSeq: 1345 NOTIFY
Event: call-info
Subscription-State: active; expires=3100
Call-Info:  <sip:as.foo.com>;appearance-index=1;appearance-
state=progressing,
            <sip:as.foo.com>;appearance-index=*;appearance-state=idle
Content-Length: 0

[F10] 200 OK A-1 → Application Server
SIP/2.0 200 OK
Via: SIP/2.0/UDP as.foo.com:5060;branch=gsdf32nashds7
From: "Shared-A" <sip:shared-a@as.foo.com>;tag=dsa3dszlfl
To: "Shared-A" <sip:shared-a@as.foo.com>;tag=323423233
Call-ID: 5j9FpLxk3uxtm8tn@a1.foo.com
CSeq: 1344 NOTIFY
Content-Length: 0

[F11] 200 OK A-2 → Application Server
SIP/2.0 200 OK
Via: SIP/2.0/UDP as.foo.com:5060;branch=gsdf32nashds7
From: "Shared-A" <sip:shared-a@as.foo.com>;tag=ggsa3dszlfl
To: "Shared-A" <sip:shared-a@as.foo.com>;tag=123456
Call-ID: 9k9FpLxk3uxtm8tn@a2.foo.com
CSeq: 1345 NOTIFY
Content-Length: 0
```

4.2.2.3 B Answers Call

[F12 to F15] shows B answering the call using standard back-to-back user agent call flow. A-1 and B are now active in a call.

```
[F12] 200 OK B → Application Server

SIP/2.0 200 OK
Via: SIP/2.0/UDP as.foo.com:5060;branch=234adfrjksd
From: "Shared-A" <sip:shared-a@as.foo.com>;tag=234wdkfj
To: "B" <sip:B@as.foo.com>;tag=1234dsf2
Call-ID: asdf2220188511@as.foo.com
CSeq: 12345 INVITE
Contact: <sip:B@b.foo.com>
Content-Length: 0

[F13] 200 OK Application Server → A-1

SIP/2.0 200 OK
Via: SIP/2.0/UDP a1.foo.com:5060;branch=m9hG4bK74bf9
From: "Shared-A" <sip:shared-a@as.foo.com>;tag=5fxced76sl
To: "B" <sip:B@as.foo.com>; tag=4323z39
Call-ID: 9848276298220188511@a1.foo.com
Call-Info:  <sip:as.foo.com>;appearance-index=1
CSeq: 43234 INVITE
Contact: <sip:B@as.foo.com>
Content-Length: 0

[F14] ACK A-1 → Application Server

ACK sip:B@as.foo.com SIP/2.0
Via: SIP/2.0/UDP a1.foo.com:5060;branch=345rbK74bf9
Max-Forwards: 70
From: "Shared-A" <sip:shared-a@as.foo.com>;tag=234wdkfj
To: "B" <sip:B@as.foo.com>;tag=1234dsf2
Call-ID: asdf2220188511@foo.com
CSeq: 1093 ACK
Contact: <sip:shared-a@a1.foo.com>
Content-Length: 0

[F15] ACK Application Server → B

ACK sip:B@b.foo.com SIP/2.0
Via: SIP/2.0/UDP as.foo.com:5060;branch=23dsdf2ksd
Max-Forwards: 70
From: "Shared-A" <sip:shared-a@as.foo.com>;tag=234wdkfj
To: "B" <sip:B@as.foo.com>;tag=1234dsf2
Call-ID: asdf2220188511@foo.com
CSeq: 12345 ACK
Contact: <sip:shared-a@as.foo.com>
Content-Length: 0
```

4.2.2.4 Application Server Notifies A-1 and A-2 that Call is Now Active

In [F16 to F19] the Application Server sends a NOTIFY-request to IP phones A-1 and A-2 with a *Call-Info* header, indicating a call appearance state of *Active* for the first call appearance on the shared line.

```
[F16] NOTIFY Application Server -> A-1
NOTIFY sip:shared-a@a1.foo.com SIP/2.0
Via: SIP/2.0/UDP as.foo.com:5060;branch=gsdf32nashds7
Max-Forwards: 70
From: "Shared-A" <sip:shared-a@as.foo.com>;tag=dsa3dszlfl
To: "Shared-A" <sip:shared-a@as.foo.com>;tag=323423233
Call-ID: 5j9FpLxk3uxtm8tn@a1.foo.com
CSeq: 1344 NOTIFY
Event: call-info
Subscription-State: active; expires=3000
Call-Info:  <sip:as.foo.com>;appearance-index=1;appearance-state=active,
            <sip:as.foo.com>;appearance-index=*;appearance-state=idle
Content-Length: 0

[F17] NOTIFY Application Server -> A-2
NOTIFY sip:shared-a@a2.foo.com SIP/2.0
Via: SIP/2.0/UDP as.foo.com:5060;branch=gsdf32nashds7
Max-Forwards: 70
From: "Shared-A" <sip:shared-a@as.foo.com>;tag=ggsa3dszlfl
To: "Shared-A" <sip:shared-a@as.foo.com>;tag=123456
Call-ID: 9k9FpLxk3uxtm8tn@a2.foo.com
CSeq: 1345 NOTIFY
Event: call-info
Subscription-State: active; expires=2999
Call-Info:  <sip:as.foo.com>;appearance-index=1;appearance-state=active,
            <sip:as.foo.com>;appearance-index=*;appearance-state=idle
Content-Length: 0

[F18] 200 OK A-1 → Application Server
SIP/2.0 200 OK
Via: SIP/2.0/UDP as.foo.com:5060;branch=gsdf32nashds7
From: "Shared-A" <sip:shared-a@as.foo.com>;tag=dsa3dszlfl
To: "Shared-A" <sip:shared-a@as.foo.com>;tag=323423233
Call-ID: 5j9FpLxk3uxtm8tn@a1.foo.com
CSeq: 1344 NOTIFY
Content-Length: 0

[F19] 200 OK A-2 → Application Server
SIP/2.0 200 OK
Via: SIP/2.0/UDP as.foo.com:5060;branch=gsdf32nashds7
From: "Shared-A" <sip:shared-a@as.foo.com>;tag=ggsa3dszlfl
To: "Shared-A" <sip:shared-a@as.foo.com>;tag=123456
Call-ID: 9k9FpLxk3uxtm8tn@a2.foo.com
CSeq: 1345 NOTIFY
Content-Length: 0
```

4.2.3 Call Park Information Outgoing Call

The following figure shows the call flow when the Cisco BroadWorks Shared Call Appearance service has been configured to provide the Call Park state. The device usually subscribes to the Call-Info event package [F1]. The notifications [F3], [F5], [F7], [F9], and [F11] contain the Call Park information in the NOTIFY message body. Also, note that notifications [F9] and F11 are triggered by the Call Park state change, even if no call activity occurs on the line.

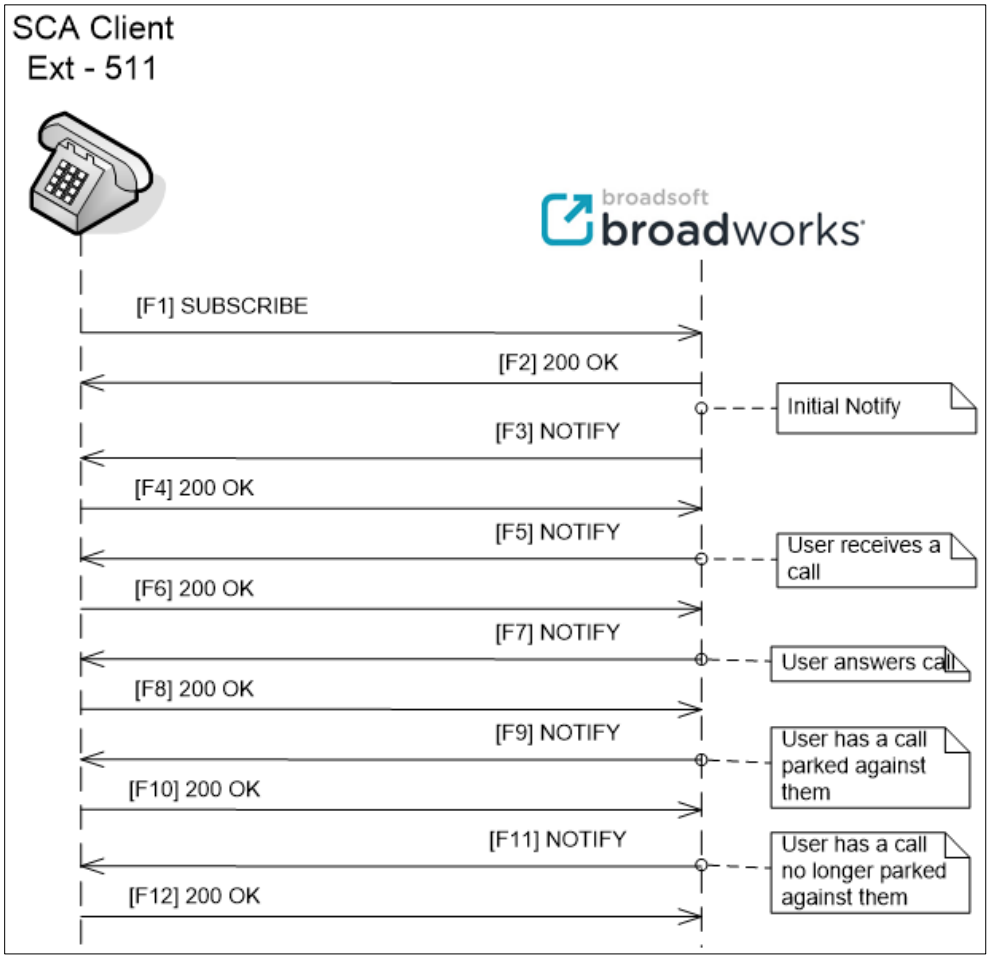


Figure 7 Shared Call Appearance (SCA) Message Flow

Initial Subscription

```
    [F1] Device 9726987511 -> Application Server
SUBSCRIBE sip:9726987511@as.bw.com:5060 SIP/2.0
Via: SIP/2.0/UDP c1.bw.com:5060;branch=z9hG4bK-76700-1-0
From: <sip:9726987511@txasdev87.net:5096>;tag=1
To: <sip:9726987511@as.bw.com:5060>
Call-ID: 1-76700@as.bw.com
CSeq: 1 SUBSCRIBE
Contact: <sip:9726987511@c1.bw.com:5060>
Expires:1800
Event:call-info
Accept: application/x-broadworks-callpark-info+xml
Max-Forwards:10
```

```
Content-Length:0

    [F2] Application Server -> Device 9726987511
SIP/2.0 200 OK
Via:SIP/2.0/UDP c1.bw.com:5060;branch=z9hG4bK-76700-1-0
From:<sip:9726987511@txasdev87.net:5096>;tag=1
To:<sip:9726987511@as.bw.com:5060>;tag=1381793443-1277837165411
Call-ID:1-76700@as.bw.com
CSeq:1 SUBSCRIBE
Expires:1798
Contact:<sip:as.bw.com:5060>
Content-Length:0

    [F3] Application Server -> Device 9726987511
NOTIFY sip:9726987511@c1.bw.com:5060 SIP/2.0
Via:SIP/2.0/UDP as.bw.com;branch=z9hG4bKBroadWorks.
From:<sip:9726987511@as.bw.com:5060>;tag=1381793443-1277837165411
To:<sip:9726987511@txasdev87.net:5096>;tag=1
Call-ID:1-76700@as.bw.com
CSeq:84394979 NOTIFY
Contact:<sip:as.bw.com:5060>
Call-Info:<sip:as.bw.com>;appearance-state=idle;appearance-index=*
Event:call-info
Subscription-State:active;expires=1798
Max-Forwards:10
Content-Type:application/x-broadworks-callpark-info+xml
Content-Length:149

<?xml version="1.0" encoding="UTF-8"?>
<x-broadworks-callpark-info
  xmlns="http://schema.broadsoft.com/callpark">
 <callpark/>
</x-broadworks-callpark-info>

    [F4] Device 9726987511 -> Application Server
SIP/2.0 200 OK
Via:SIP/2.0/UDP as.bw.com;branch=z9hG4bKBroadWorks.1lp1imq-
as.bw.comV5096-0-84394979-1381793443-1277837165411
From:<sip:9726987511@as.bw.com:5060>;tag=1381793443-1277837165411
To:<sip:9726987511@txasdev87.net:5096>;tag=1
Call-ID:1-76700@as.bw.com
CSeq:84394979 NOTIFY
Contact: <sip:c1.bw.com:5060;transport=UDP>
Content-Length: 0
```

User Bob (500) receives a call from Tom (503). NOTIFY has both the Call-Info and the Call Park message body.

```
    [F5] Application Server -> Device 9726987511
NOTIFY sip:c1.bw.com:5060;transport=UDP SIP/2.0
Via:SIP/2.0/UDP as.bw.com;branch=z9hG4bKBroadWorks
From:<sip:9726987511@as.bw.com:5060>;tag=1381793443-1277837165411
To:<sip:9726987511@txasdev87.net:5096>;tag=1
Call-ID:1-76700@as.bw.com
CSeq:84405932 NOTIFY
Contact:<sip:as.bw.com:5060>
Call-Info:<sip:as.bw.com>;appearance-state=alerting;
  appearance-uri="\"Tom north\"
  <sip:9726987503@as.bw.com;user=phone>";
  appearance-index=1,<sip:as.bw.com>;
  appearance-state=idle;appearance-index=*
Event:call-info
```

```
Subscription-State:active;expires=1787
Max-Forwards:10
Content-Type:application/x-broadworks-callpark-info+xml
Content-Length:149

<?xml version="1.0" encoding="UTF-8"?>
<x-broadworks-callpark-info
  xmlns="http://schema.broadsoft.com/callpark">
 <callpark/>
</x-broadworks-callpark-info>

    [F6] Device 9726987511 -> Application Server
SIP/2.0 200 OK
Via:SIP/2.0/UDP as.bw.com;branch=z9hG4bKBroadWorks.1lp1imq-
as.bw.comV5096-0-84405932-1381793443-1277837165411
From:<sip:9726987511@as.bw.com:5060>;tag=1381793443-1277837165411
To:<sip:9726987511@txasdev87.net:5096>;tag=1
Call-ID:1-76700@as.bw.com
CSeq:84405932 NOTIFY
Contact: <sip:c1.bw.com:5060;transport=UDP>
Content-Length: 0

    [F7] Application Server -> Device 9726987511
NOTIFY sip:c1.bw.com:5060;transport=UDP SIP/2.0
Via:SIP/2.0/UDP as.bw.com;branch=z9hG4bKBroadWorks
From:<sip:9726987511@as.bw.com:5060>;tag=1381793443-1277837165411
To:<sip:9726987511@txasdev87.net:5096>;tag=1
Call-ID:1-76700@as.bw.com
CSeq:84408388 NOTIFY
Contact:<sip:as.bw.com:5060>
Call-Info:<sip:as.bw.com>;appearance-state=active;
  appearance-uri="\"Tom north\"
  <sip:9726987503@as.bw.com;user=phone>";
  appearance-index=1,<sip:as.bw.com>;
  appearance-state=idle;appearance-index=*
Event:call-info
Subscription-State:active;expires=1785
Max-Forwards:10
Content-Type:application/x-broadworks-callpark-info+xml
Content-Length:149

<?xml version="1.0" encoding="UTF-8"?>
<x-broadworks-callpark-info
  xmlns="http://schema.broadsoft.com/callpark">
<callpark/>
</x-broadworks-callpark-info>

    [F8] Device 9726987511 -> Application Server
SIP/2.0 200 OK
Via:SIP/2.0/UDP as.bw.com;branch=z9hG4bKBroadWorks
From:<sip:9726987511@as.bw.com:5060>;tag=1381793443-1277837165411
To:<sip:9726987511@txasdev87.net:5096>;tag=1
Call-ID:1-76700@as.bw.com
CSeq:84408388 NOTIFY
Contact: <sip:c1.bw.com:5060;transport=UDP>
Content-Length: 0
```

User Bob (500) has a call parked against them. NOTIFY has both the Call-Info and the Call Park message body.

```
    [F9] Application Server -> Device 9726987511
NOTIFY sip:c1.bw.com:5060;transport=UDP SIP/2.0
Via:SIP/2.0/UDP as.bw.com;branch=z9hG4bKBroadWorks
From:<sip:9726987511@as.bw.com:5060>;tag=1381793443-1277837165411
To:<sip:9726987511@txasdev87.net:5096>;tag=1
Call-ID:1-76700@as.bw.com
CSeq:84447473 NOTIFY
Contact:<sip:as.bw.com:5060>
Call-Info:<sip:as.bw.com>;appearance-state=active;
  appearance-uri="\"Tom north\"
  <sip:9726987503@as.bw.com;user=phone>";
  appearance-index=1,<sip:as.bw.com>;
  appearance-state=idle;appearance-index=*
Event:call-info
Subscription-State:active;expires=1746
Max-Forwards:10
Content-Type:application/x-broadworks-callpark-info+xml
Content-Length:333

<?xml version="1.0" encoding="UTF-8"?>
<x-broadworks-callpark-info
  xmlns=http://schema.broadsoft.com/callpark>
 <callpark>
  <parked>
   <identity display="Alice south">
    sip:9726987601@as.bw.com;user=phone
   </identity>
  </parked>
 </callpark>
</x-broadworks-callpark-info>

    [F10] Device 9726987511 -> Application Server
SIP/2.0 200 OK
Via:SIP/2.0/UDP as.bw.com;branch=z9hG4bKBroadWorks
From:<sip:9726987511@as.bw.com:5060>;tag=1381793443-1277837165411
To:<sip:9726987511@txasdev87.net:5096>;tag=1
Call-ID:1-76700@as.bw.com
CSeq:84447473 NOTIFY
Contact: <sip:c1.bw.com:5060;transport=UDP>
Content-Length: 0
```

The parked user hangs up. NOTIFY has both the Call-Info and the Call Park message body.

```
    [F11] Application Server -> Device 9726987511
NOTIFY sip:c1.bw.com:5060;transport=UDP SIP/2.0
Via:SIP/2.0/UDP as.bw.com;branch=z9hG4bKBroadWorks
From:<sip:9726987511@as.bw.com:5060>;tag=1381793443-1277837165411
To:<sip:9726987511@txasdev87.net:5096>;tag=1
Call-ID:1-76700@as.bw.com
CSeq:84447475 NOTIFY
Contact:<sip:as.bw.com:5060>
Call-Info:<sip:as.bw.com>;appearance-state=active;
  appearance-uri="\"Tom north\"
  <sip:9726987503@as.bw.com;user=phone>";
  appearance-index=1,<sip:as.bw.com>;
  appearance-state=idle;appearance-index=*
Event:call-info
Subscription-State:active;expires=1740
```

```
Max-Forwards:10
Content-Type:application/x-broadworks-callpark-info+xml
Content-Length:149

<?xml version="1.0" encoding="UTF-8"?>
<x-broadworks-callpark-info
  xmlns="http://schema.broadsoft.com/callpark">
 <callpark/>
</x-broadworks-callpark-info>

    [F12] Device 9726987511 -> Application Server
SIP/2.0 200 OK
Via:SIP/2.0/UDP as.bw.com;branch=z9hG4bKBroadWorks
From:<sip:9726987511@as.bw.com:5060>;tag=1381793443-1277837165411
To:<sip:9726987511@txasdev87.net:5096>;tag=1
Call-ID:1-76700@as.bw.com
CSeq:84447475 NOTIFY
Contact: <sip:c1.bw.com:5060;transport=UDP>
Content-Length: 0
```

4.3 Event Package Name

The name of this event is “Call-Info”. It is carried in the *Event and Allow-Events* header, as defined in *RFC 6665* [3].

4.4 Event Package Parameters

This package does not define any event package parameters.

4.5 SUBSCRIBE Bodies

A SUBSCRIBE for a Call-Info package must not contain a body. A body is not required for this application of the Call-Info event package. Anybody sent in a SUBSCRIBE is ignored by the notifier.

4.6 Subscription Duration

Subscribers should specify the duration of a subscription with an *Expires* header in the SUBSCRIBE request. It is recommended that the subscription use the same refresh period as the REGISTER event. If the notifier changes the expiration period to a lower value (via an *Expires* header in the final response), the subscriber should honor it.

If the subscriber does not specify the duration of a subscription, then the notifier chooses the default expiration. The recommended default is 1800 seconds.

4.7 Subscription Termination

The Application Server supports only one subscription for each line-side Address of Record, that is, each line/port. If the Application Server receives a SUBSCRIBE request for a new subscription, it terminates any existing subscription for that line/port.

Normally, if a user agent at a different network address sends a SUBSCRIBE request, the Application Server implicitly terminates an existing subscription and does not send a NOTIFY request for the terminated subscription. However, the Application Server does support a mechanism to force the sending of a NOTIFY request for the terminated subscription. This mechanism is intended to support a single-login policy for a client such as UC-One Communicator. Suppose, for example, that a user uses the client application both at work and at home. The single-login policy allows only one login at a time, which means the user can be logged in from either work or home, but not both at the same time. If the user forgets to log out at home, then logs in at work, the single-login policy forces the Application Server to automatically log out the user's client at home. When the Application Server terminates the subscription, it sends a NOTIFY request with the *Subscription-State* header value of "terminated" and a *reason* parameter value of "duplicate". This *reason* parameter value is a signal to the client that it should not automatically try to log in again.

The Application Server sends this NOTIFY request for the terminated subscribe only if the following conditions hold:

- The *From* header URI in the SUBSCRIBE request has an *appid* parameter.
- The value of the *appid* URI parameter matches the value of the *appid* parameter for the existing unexpired subscription. The Application Server received this existing *appid* value in the previous SUBSCRIBE request. When comparing the two *appid* values, the Application Server does a case-sensitive comparison.

If these conditions are not met, then the Application Server terminates the older subscription but does not send a NOTIFY request.

The call flow in the following diagram shows how the single-login policy works. At the start of the scenario, Client A-2 has an unexpired subscription. Client A-1 sends a SUBSCRIBE request for the Call-Info package. The *From* header URI in the SUBSCRIBE request has an *appid* parameter with the value "myapp". When the Application Server processes the SUBSCRIBE request, it checks for an existing subscription with the same value for *appid* and it detects the duplicate subscription associated with Client A-2. The Application Server then terminates the subscription for Client A-2 and sends a NOTIFY request with subscription state *terminated*. The NOTIFY request has a *Subscription-State* header with the value "terminated" and a *reason* parameter with the value "duplicate".

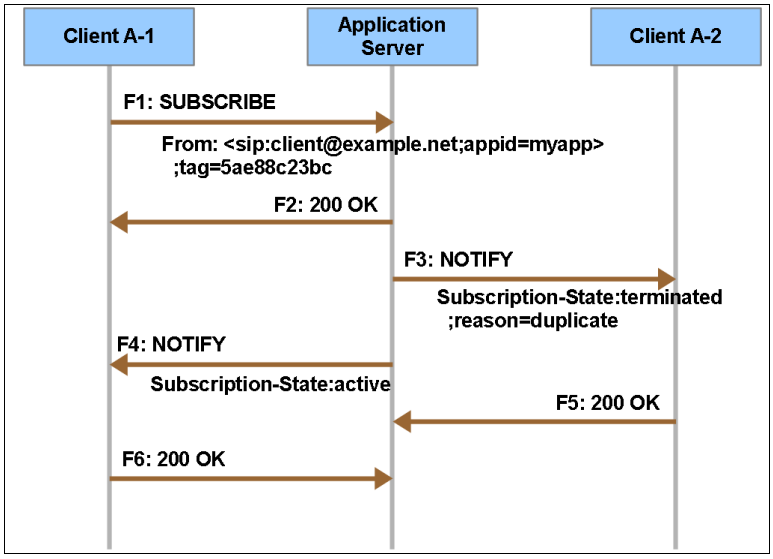


Figure 8 Termination of Duplicate Subscription

4.8 NOTIFY Bodies

A NOTIFY for a Call-Info package can optionally contain a message body. All call states are reflected through the *Call-Info* header, but the Shared Call Appearance can be optionally configured to provide Call Park information in the NOTIFY messages. When so configured, the NOTIFY message contains an *application/x-broadworks-callpark* message body as defined in section *12.7 NOTIFY Bodies*. If not configured to send Call Park notifications, then the NOTIFY messages do not contain a message body.

4.9 Subscriber Generation of SUBSCRIBE Requests

Subscriber generation of SUBSCRIBE requests must follow all rules with respect to subscriber behavior, as described in *RFC 6665* [3]. For this application, SIP phones that are configured with shared lines should generate SUBSCRIBE requests immediately after successfully registering a location against the provisioned address of record. The *Request-URI*, *From*, and *To* headers must be the same as the address of record of the shared line.

4.10 Notifier Processing of SUBSCRIBE Requests

Notifier processing of SUBSCRIBE requests must follow all rules with respect to notifier behavior, as described in *RFC 6665* [3]. The Call-Info package contains potentially sensitive information. Subscriptions should be authorized by the notifier. The notifier may perform implicit authorization by looking at the source IP address of the SUBSCRIBE event and matching it to any endpoints that have successfully been authenticated through the registration process. The notifier may choose to explicitly challenge the subscriber for authentication by using a *401 Unauthorized* response. If the subscriber has also registered a location for this address of record, then it should use the same credentials to answer the subscription challenge.

4.11 Notifier Generation of NOTIFY Requests

Notifier generation of NOTIFY requests must follow all rules with respect to notifier behavior, as described in *RFC 6665* [3]. Notifications are generated for the Call-Info package whenever the supplementary call information associated with an address of record (in this case a line) changes. As described in *RFC 3261* [2], supplementary call information can be almost anything. For the purposes of these applications, the call information includes a complete list of all the call appearances allowed on the address of record and the state of each call appearance. The notifier sends this information in a NOTIFY request by explicitly including a *Call-Info* header in addition to the required *NOTIFY request* headers. See the sections above for details about the content of the *Call-Info* header and how it should be used.

If the Shared Call Appearance service is configured to send Call Park information, then NOTIFY requests are also generated whenever the Call Park state changes.

4.12 Subscriber Processing of NOTIFY Requests

In general, subscriber processing of NOTIFY requests must follow all rules with respect to subscriber behavior, as described in *RFC 6665* [3]. The SIP specific event notification framework expects packages to specify how a subscriber processes NOTIFY requests, and in particular, how it uses the NOTIFY requests to construct a coherent view of the state of the subscribed resource.

To be specific, the Call-Info package itself has no state, but in this application, the state of the call appearances of a given address of record can be derived from the content of the *Call-Info* header that is delivered in the NOTIFY request. This state is described in detail in the following sections. Because these applications use the content of the NOTIFY's *Call-Info* header to visually present the call appearance state to the user, the subscriber (that is, the phone) should perform some level of authorization before processing NOTIFYs. The subscriber may perform implicit authorization by looking at the source IP address of the NOTIFY request event and matching it to any one of the IP addresses of the notifier, as resolved through Domain Name System (DNS). The subscriber may perform explicit authentication by challenging the notifier, by using a *401 Unauthorized* response. If the subscriber has also registered a location for this address of record, then the notifier should use those same credentials to answer the notification challenge.

If the Shared Call Appearance service is configured to send the Call Park information, then the NOTIFY messages contain the current Call Park state. The subscriber must discard any previous Call Park state information and replace it with the received information.

4.13 Handling of Forked Requests

Although it is theoretically possible for a SUBSCRIBE request to fork, this is not likely for the applications described in this document. Robust subscriber implementations should be prepared to install multiple Call-Info subscriptions, but practically speaking, this never happens.

4.14 Rate of Notifications

A very active phone line may result in many notifications sent to every endpoint in the shared call appearance group. Endpoints should be prepared to accept NOTIFY requests as frequently as a few times per second in burst scenarios (which should be infrequent).

5 Line-Seize Event Package

5.1 Overview

SIP has been employed as a line-side access protocol in voice networks. When compared to other line-side access protocols as Simple Conference Control Protocol (SCCP) or Media Gateway Control Protocol (MGCP), SIP is considered a “peer to peer” protocol. Because of this, Session Initiation Protocol - User Agents (SIP-UAs) embedded in IP phones typically provide local dial tone, digit collection, and essentially perform their own line-side call setup. This works well for lines that are private to the endpoint. However, this behavior can cause race conditions when emulating shared line functionality across multiple endpoints. Emulating shared-line functionality requires some central line controller to arbitrate which endpoint has “seized” the line. The purpose of the line-seize event package is to support the concept of a line-seizure between an endpoint and line controller, as found in a typical “shared line” solution.

The approach defined here requires that all endpoints that participate in a shared line obtain a subscription to the line-seize package for the shared line before presenting the user with dial tone or collecting digits. This means the endpoints must send a SUBSCRIBE-request for the line-seize event package whenever a user attempts to take the shared line off hook. The “line controller” guarantees that only one subscription to the line-seize event package can exist for a given shared line resource. When the line controller grants a subscription to the line-seize package, it responds to the SUBSCRIBE with a *200 OK* response and generates a NOTIFY, indicating that the subscription state is now *active*. This is an indicator to the endpoint that it can safely play dial tone to the user and collected digits. Once the digits have been collected, then the endpoint can send an INVITE request. While one endpoint has obtained the line-seize subscription, any requests from other endpoints to use that shared line are rejected with a “480 Temporarily Unavailable” response.

The line-seize event package is an instantiation of the SIP specific event notification framework (as defined in *RFC 6665* [3]). Following is a description of the event package details, as per *RFC 6665* [3].

To avoid scenarios where the endpoint seizes the line, but never actually uses it (that is, it never sends an INVITE request) and effectively hangs the line, the line-seize package must have a suitable duration associated with it. Fortunately, the SIP specific event notification framework, as defined in *RFC 6665* [3], explicitly describes a mechanism for negotiating subscription duration between the subscriber (the endpoint) and the notifier (the line controller). The SUBSCRIBE-request to the line-seize package contains an expiration time as offered by the endpoint – and in response the line controller can choose to set a smaller expiration time. If the endpoint has not collected digits and sent an INVITE request before the subscription has expired, then it must refresh the subscription or risk losing it to another endpoint. Endpoints should only collect digits for a suitable length of time before they let the subscription expire naturally. This avoids scenarios where one endpoint in the shared call appearance group goes off hook accidentally and effectively ties up the line. After a reasonable amount of time, the endpoint should stop refreshing the subscription, let it expire naturally, and then apply an appropriate local treatment (re-order tone), so the end user is aware that digits can no longer being collected.

If the user goes on hook before digit collection is complete, effectively terminating the line seizure without originating a call, then the endpoint must clear the line seizure by explicitly cancelling the subscription. This allows the line to be immediately seized by another endpoint. As per *RFC 3261* [2], this is effectively achieved by refreshing the subscription with an expiration of “0”.

5.2 Example Message Flows

The following figure shows the call flow as a result of a user at A-1 taking the shared line off hook. The line controller in this scenario is provided by a central Application Server (AS).

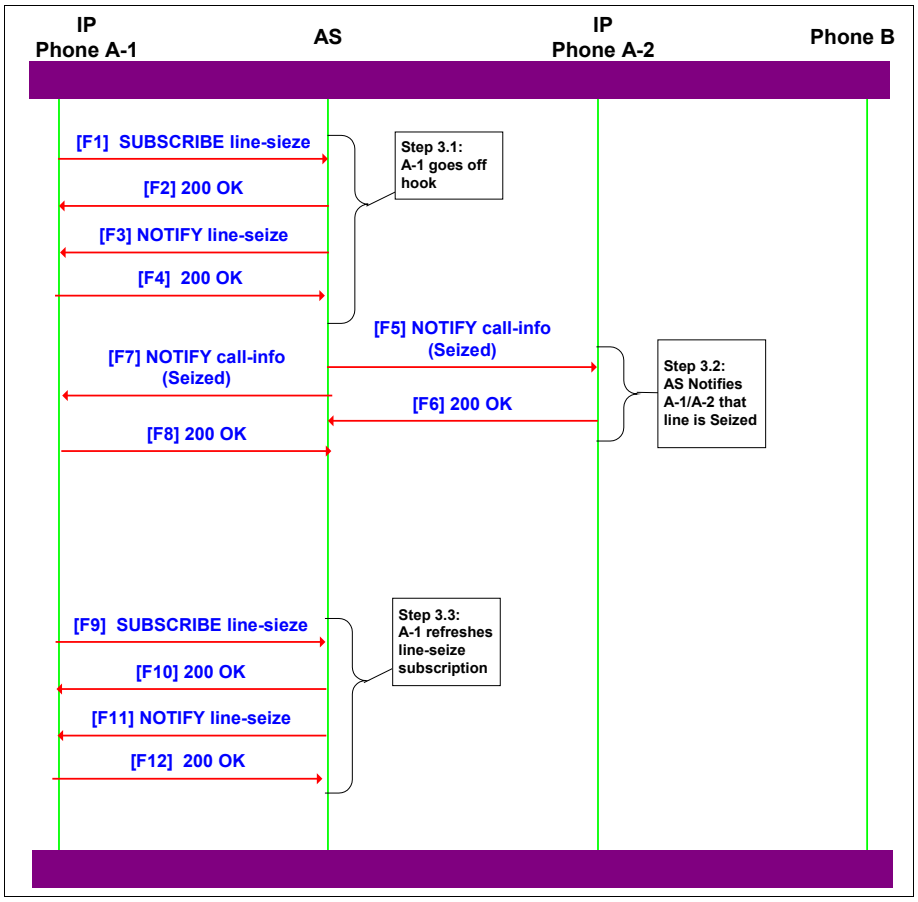


Figure 9 Call Flow as Result of User at A-1 Taking Shared Line Off Hook

In [F1 to F4], IP phone A-1 must be granted a subscription to the line-seize package. Only after the subscription has been granted, should the IP phone play dial tone to the user.

```
[F1] SUBSCRIBE A-1 -> Application Server
SUBSCRIBE sip:shared-a@as.foo.com SIP/2.0
Via: SIP/2.0/UDP a1.foo.com:5060;branch=gsdf32nashds7
Max-Forwards: 70
From: "Shared-A" <sip:shared-a@as.foo.com>;tag=gsa3dszlfl
To: "Shared-A" <sip:shared-a@as.foo.com>
Call-ID: 6j9FpLxk3uxtm8to@a1.foo.com
Call-Info:  <sip:as.foo.com>; appearance-index=1
CSeq: 7 SUBSCRIBE
Event: line-seize
Expires: 15
Contact: <sip:shared-a@a1.foo.com>
Content-Length: 0

[F2] 200 OK Application Server → A-1
SIP/2.0 200 OK
Via: SIP/2.0/UDP a1.foo.com:5060;branch=gsdf32nashds7
```

```
From: "Shared-A" <sip:shared-a@as.foo.com>;tag=gsa3dszlfl
To: "Shared-A" <sip:shared-a@as.foo.com>;tag=423423233
Call-ID: 6j9FpLxk3uxtm8tn@a1.foo.com
CSeq: 7 SUBSCRIBE
Contact: sip:shared-a@as.foo.com
Event: line-seize
Expires=15
Content-Length: 0

[F3] NOTIFY Application Server -> A-1
NOTIFY sip:shared-a@a1.foo.com SIP/2.0
Via: SIP/2.0/UDP as.foo.com:5060; branch=gfasdf237
Max-Forwards: 70
From: "Shared-A" <sip:shared-a@as.foo.com>;tag=gsa3dszlfl
To: "Shared-A" <sip:shared-a@as.foo.com>;tag=423423233
Call-ID: 6j9FpLxk3uxtm8to@a1.foo.com
Call-Info:  <sip:as.foo.com>; appearance-index=1
CSeq: 9 NOTIFY
Event: line-seize
Subscription-State: active; 14
Contact: sip:shared-a@as.foo.com
Content-Length: 0

[F4] 200 OK A-1 → Application Server
SIP/2.0 200 OK
Via: SIP/2.0/UDP a1.foo.com:5060; branch=gfasdf237
From: "Shared-A" <sip:shared-a@as.foo.com>;tag=gsa3dszlfl
To: "Shared-A" <sip:shared-a@as.foo.com>;tag=423423233
Call-ID: 6j9FpLxk3uxtm8tn@a1.foo.com
CSeq: 9 NOTIFY
Content-Length: 0
```

5.3 Event Package Name

The name of this event package is “line-seize”. It is carried in the *Event and Allow-Events* header, as defined in *RFC 6665* [3].

5.4 Event Package Parameters

This package does not define any event package parameters.

5.5 SUBSCRIBE Bodies

A SUBSCRIBE for a line-seize package must not contain a body. No body is required for this application of the line-seize event package. Any body sent in a SUBSCRIBE is ignored by the notifier.

5.6 Subscription Duration

Subscribers should specify the duration of a subscription with an *Expires* header in the SUBSCRIBE request. It is recommended that the subscription use a refresh period, equal to roughly how long it takes for a typical end user to dial a full E.164 number. If the notifier changes the expiration period to a lower value (via an *Expires* header in the final response), the subscriber should honor it.

If the subscriber does not specify the duration of a subscription, then the notifier chooses a default expiration value. The recommended default is 15 seconds.

5.7 NOTIFY Bodies

A NOTIFY for a line-seize package must not contain a body. No body is required for the line-seize event package, as it uses a very simple state machine of either *active* or *terminated*, which is easily reflected through the *Subscription-State* header. A body contained in a NOTIFY request should be ignored by the subscriber.

5.8 Subscriber Generation of SUBSCRIBE Requests

Subscriber generation of SUBSCRIBE requests must follow all rules with respect to subscriber behavior, as described in *RFC 3265* [3]. Typically, SIP phones that are configured with shared lines should generate SUBSCRIBE requests immediately after a user's attempts to go off hook on an idle shared line. The *Request-URI*, *From*, and *To* headers must be the same as the address of record of the shared line.

5.9 Notifier Processing of SUBSCRIBE Requests

Notifier processing of SUBSCRIBE requests must follow all rules with respect to notifier behavior, as described in *RFC 3265* [3]. The line-seize package contains potentially sensitive information. Subscriptions should be authorized by the notifier. The notifier may perform implicit authorization by looking at the source IP address of the SUBSCRIBE event and matching it to any endpoints that have successfully been authenticated through the registration process. The notifier may choose to explicitly challenge the subscriber for authentication, by using a *401 Unauthorized* response. If the subscriber has also registered a location for this address of record, then it should use the same credentials to answer the subscription challenge.

5.10 Notifier Generation of NOTIFY Requests

Notifier generation of NOTIFY requests must follow all rules with respect to notifier behavior, as described in *RFC 3265* [3]. Notifications are generated for the line-seize package immediately after a subscription has been accepted.

5.11 Subscriber Processing of NOTIFY Requests

In general, subscriber processing of NOTIFY requests must follow all rules with respect to subscriber behavior, as described in *RFC 3265* [3] The SIP specific event notification framework expects packages to specify how a subscriber processes NOTIFY requests, and in particular, how it uses the NOTIFY requests to construct a coherent view of the state of the subscribed resource.

The line-seize package has binary state. Either the line-seize subscription is *active* or it is *terminated*. This state is reflected in the *Subscription-State* header. The endpoint should perform some level of authorization before processing NOTIFYs. The subscriber may perform implicit authorization by looking at the source IP address of the NOTIFY request event and matching it to any one of the known IP addresses of the notifier (in this case the Application Server), as resolved through DNS. The subscriber may perform explicit authentication by challenging the notifier, by using a "401 Unauthorized" response. If the subscriber has also registered a location for this address of record, then the notifier should use those same credentials to answer the notification challenge.

5.12 Handling of Forked Requests

Although it is theoretically possible for a SUBSCRIBE request to fork, this is not likely for typical applications that use the line-seize event package. Since the nature of the subscription guarantees that only one subscription exist at a time, then the first forked request that arrives at the Application Server is granted a subscription, while the rest of the SUBSCRIBE-requests are rejected.

5.13 Rate of Notifications

Notifications are only sent when the endpoint explicitly refreshes the subscription (about every 15 seconds), when the subscription expires (also after about 15 seconds), or when the subscription is terminated.

6 Remote Control Talk Event Package

6.1 Overview

When providing CTI-based applications such as desktop call control clients, a service delivery platform must be able to request that an endpoint answer an incoming call. This extension provides the ability for an Application Server to send an asynchronous NOTIFY event to an endpoint, using an existing INVITE dialog. When received, the endpoint can elect to honor the request and answer the call, without local user intervention.

This extension can also be used to support remotely retrieving a call that has been previously held. When the endpoint receives the NOTIFY event and the associated dialog is in the *held* state, then the endpoint can elect to honor the request and retrieve the call from the *held* state.

The talk event package is an instantiation of the SIP specific event notification framework (as defined in *RFC 3265* [3]). Following are details of the event package, as per *RFC 3265* [3].

6.2 Example Message Flow

The following diagram shows the call flow as a result of a user performing a Click-To-Answer function on an incoming call from the Public Switched Telephone Network (PSTN).

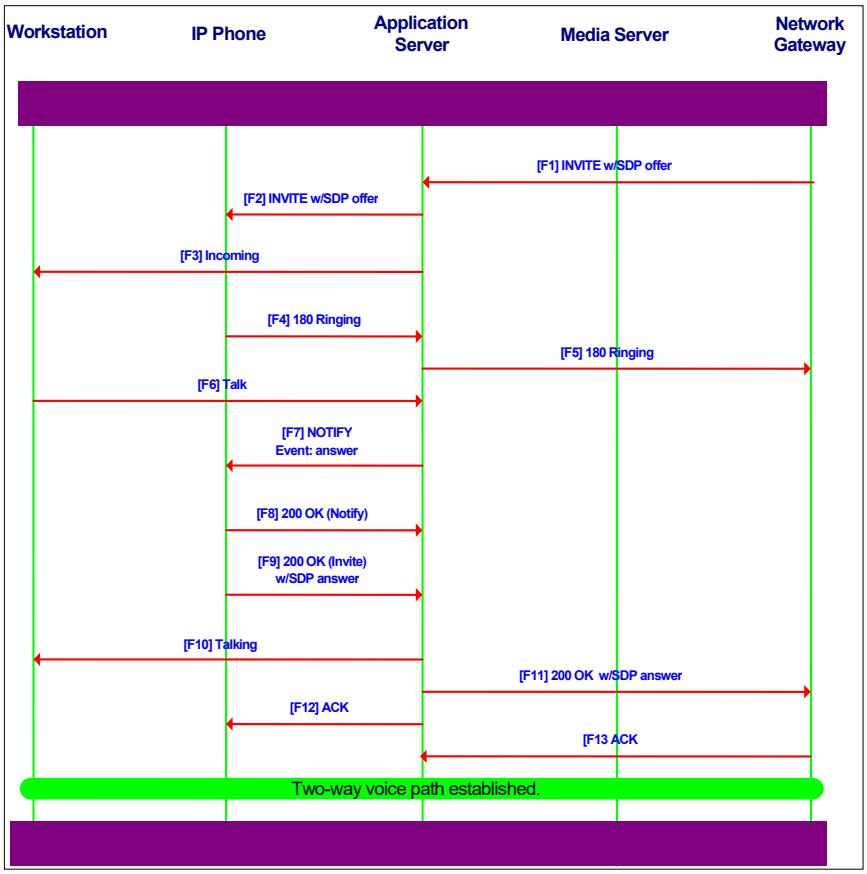


Figure 10 Click-To-Answer Call Flow

The call arrives at the Application Server in [F1] as usual. The Application Server sends an INVITE to the IP phone registered against the user profile in [F2].

```
[F2] INVITE Application Server -> IP phone
INVITE sip:3015551000@ipphone.com SIP/2.0
Via: SIP/2.0/UDP a1.foo.com:5060;branch=m9hG4bK74bf9
Max-Forwards: 70
From: "Joe" <sip:3015551212@a1.foo.com>;tag=5fxced76s2
To: "Jane" <sip:3015551000@a1.foo.com>
Call-ID: 984827629822018851 2@a1.foo.com
Call-Info:  <sip:as.foo.com>;appearance-index=1
CSeq: 43234 INVITE
Contact: <sip:a1.foo.com:5060>
Allow: ACK, BYE, CANCEL, INFO, INVITE, OPTIONS, PRACK, REFER
Supported:  100rel, timer
Content-Type:  application/sdp
Content-Length: 192

v=0
o=foo-UA 5184 14642 IN IP4 networkgateway.com
s=SIP Call
c=IN IP4 networkgateway.com
t=0 0
m=audio 27192 RTP/AVP 0 8 18
a=rtpmap :0 PCMU/8000
a=rtpmap :8 PCMA/8000
a=rtpmap:10 G729/0000
```

At about the same time, the Application Server sends a notification [F3] to the call client on the workstation, indicating that a call has arrived. The call client presents this to the end user. At about the same time as the user sees the call client present the incoming call on the desktop, the phone on his/her desk starts to alert and the phone responds to the INVITE with a *180 Ringing* in [F4].

```
[F4] SIP/2.0 180 Ringing
Via: SIP/2.0/UDP a1.foo.com:5060;branch=m9hG4bK74bf9
From: "Joe" <sip:3015551212@a1.foo.com>;tag=5fxced76s2
To: "Jane" <sip:3015551000@a1.foo.com>;tag=763jxd879sa
Call-ID: 984827629822018851 2@a1.foo.com
Call-Info:  <sip:as.foo.com>;appearance-index=1
CSeq: 43234 INVITE
Contact: <sip: 3015551000@ipphone.com:5060>
Allow-Events:hold,talk
Content-Length: 0
```

The *180 Ringing* progress is passed on to the network gateway in [F5]. The *Allow-Events* header in the *180 Ringing* indicates to the Application Server that this endpoint supports remote call control primitives. When the user selects the incoming call and clicks on **talk**, an event is sent from the call client to the Application Server in [F6], indicating that the user is requesting that the incoming call be answered. The Application Server reacts by sending a NOTIFY to the SIP phone in [F7].

```
[F7] NOTIFY Application Server -> IP Phone
NOTIFY sip:3015551000@ipphone.com SIP/2.0
Via: SIP/2.0/UDP a1.foo.com:5060;branch=m9hG4bK74bf9
From: "Joe" <sip:3015551212@a1.foo.com>;tag=5fxced76s2
To: "Jane" <sip:3015551000@a1.foo.com> ; tag=432d33
Call-ID: 984827629822018851 2@a1.foo.com
CSeq: 434 NOTIFY
Contact: <sip:a1.foo.com:5060>
Event: talk
```

```
Subscription-State: active
Contact: <sip:a1.foo.com:5060>
Content-Length: 0
```

The IP phone indicates that it honors the request by responding to the NOTIFY with a *200 OK*.

NOTE: For brevity, the authorization procedure is not shown in the figure. The phone should always challenge an incoming NOTIFY for credentials that authorize the notifier to perform remote call control.

```
[F8] 200 OK IP Phone -> Application Server
SIP/2.0 200 OK
Via: SIP/2.0/UDP a1.foo.com:5060;branch=m9hG4bK74bf9
From: "Joe" <sip:3015551212@a1.foo.com>;tag=5fxced76s2
To: "Jane" <sip:3015551000@a1.foo.com> ; tag=432d33
Call-ID: 984827629822018851 2@a1.foo.com
CSeq: 434 NOTIFY
Contact: <sip:a1.foo.com:5060>
Content-Length: 0
```

The phone then automatically answers the incoming call by forcing off-hook and activating the speaker.

6.3 Event Package Names

The event package name is called “talk”. It is carried in the *Event and Allow-Events* header, as defined in *RFC 3265* [3].

6.4 Event Package Parameters

This event package does not define any event package parameters.

6.5 SUBSCRIBE Bodies

The talk event package does not support dynamic subscriptions through SUBSCRIBE requests.

6.6 Subscription Duration

Subscriptions are implied by sessions established through INVITE requests and last throughout the duration of the session. When a UAC sends an INVITE to a UAS, it adds an *Allow-Events* header to the request, indicating all of the event packages it supports. This implicitly creates the subscription between the UAC and the UAS. When a UAS responds to the INVITE with an *18x* provisional response or a *200 OK* response, it adds an *Allow-Events* header indicating all of the event packages it supports. This implicitly creates the subscription between the UAS and the UAC. These subscriptions last until the session has been terminated with a BYE or CANCEL transaction.

6.7 NOTIFY Bodies

A NOTIFY for a talk package must not contain a body. No body is required for the talk event package as it has a trivial state.

6.8 Subscriber Generation of SUBSCRIBE Requests

Subscribers should not generate SUBSCRIBE requests. Subscriptions are granted to subscribers implicitly by the establishment of sessions between two user agents.

6.9 Notifier Processing of SUBSCRIBE Requests

Notifiers should not process SUBSCRIBE requests. If received, they should be rejected.

6.10 Subscriber Processing of NOTIFY Requests

In general, subscriber processing of NOTIFY requests must follow all rules with respect to subscriber behavior, as described in *RFC 3265* [3]. The SIP specific event notification framework expects packages to specify how a subscriber processes NOTIFY requests and, in particular, how it uses the NOTIFY requests to construct a coherent view of the state of the subscribed resource.

Because these event notifications have the ability to remotely control the user's handset behavior, the subscriber (for example, the phone) should perform some level of authorization before processing NOTIFYs. The subscriber may perform implicit authorization by looking at the source IP address of the NOTIFY request event and matching it to any one of the IP addresses of the notifier (in this case, the Application Server) as resolved through DNS. The subscriber may perform explicit authentication by challenging the notifier, by using a *401 Unauthorized* response. If the subscriber has also registered a location for this address of record, then the notifier should use those same credentials to answer the notification challenge.

When a subscriber receives a talk event notification for a particular dialog, it must apply the "talk" action to the local session in progress. If the session is in the *alerting* state, then the subscriber should attempt to answer the session, by forcing the call off hook and turning on the speakerphone (if supported). If the session is in the *held* state, then the subscriber should attempt to retrieve the call. If the subscriber receives a talk notification for a session that is not either in the *alerting* state or the *held* state, then the talk notification should be rejected.

Note that the subscriber should always respond to the NOTIFY transaction if it was received and before the action is complete. A *200 OK* response to a NOTIFY transaction only indicates that the notification has been received, and not that any action associated with the notification was successfully completed.

6.11 Handling of Forked Requests

NOTIFYs should not be forked.

6.12 Rate of Notifications

The rate of notifications is dictated by the end-user interface on the call control client and can vary from application to application. Endpoints should be prepared to accept NOTIFY requests as frequently as once every two or three seconds.

6.13 Notifier Generation of NOTIFY Requests

Notifications are generated for the talk package whenever a remote call control application requests that an alerting session be answered, or a held session be retrieved. Notifiers should not generate a talk event notification for a session that is already active and streaming media.

7 Remote Control Hold Event Package

7.1 Overview

When providing CTI-based applications such as desktop call control clients, a service delivery platform must be able to remotely request that an endpoint put a call on hold. This extension provides the ability for an Application Server to send an asynchronous NOTIFY event to an endpoint using an existing INVITE dialog. When received, the endpoint can elect to honor the request and put a call on hold, without local user intervention.

The hold event package is an instantiation of the SIP specific event notification framework (as defined in *RFC 3265* [3]). Following is a description of the event package, as per *RFC 3265* [3].

7.2 Example Message Flow

The following diagram shows the call flow as a result of a user performing a Click-To-Hold function on an active call from the PSTN.

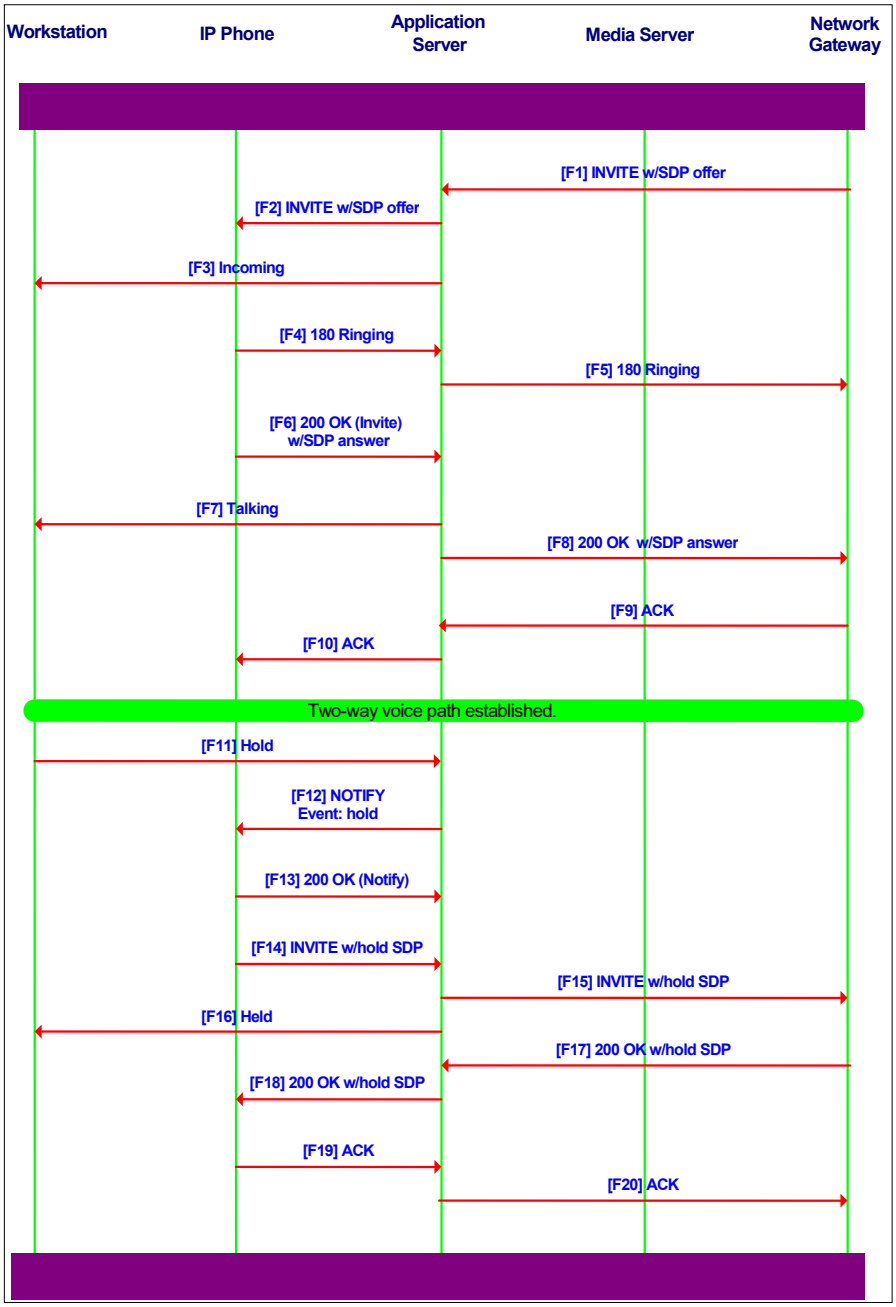


Figure 11 Click-To-Hold Call Flow

The call is set up as usual in [F1-F10] and a two-way audio is established. Now in [F11] the user at the call client decides to remotely hold the call and sends a "hold" request to the Application Server. The Application Server discovered that the endpoint supported the hold event package through an *Allow-Events* header in the *180 Ringing* provisional response at [F4]. The Application Server reacts by sending a NOTIFY with a hold event in [F12]. Note that this NOTIFY is sent using the same dialog as the session it is acting on.

```
[F12] NOTIFY Application Server -> IP phone
NOTIFY sip:3015551000@ipphone.com SIP/2.0
Via: SIP/2.0/UDP a1.foo.com:5060;branch=m9hG4bK74bf9
```

```
From: "Joe" <sip:3015551212@a1.foo.com>;tag=5fxced76s2
To: "Jane" <sip:3015551000@a1.foo.com> ; tag=432d33
Call-ID: 984827629822018851 2@a1.foo.com
CSeq: 444 NOTIFY
Contact: <sip:a1.foo.com:5060>
Event: hold
Subscription-State: active
Contact: <sip:a1.foo.com:5060>
Content-Length: 0
```

The IP phone indicates that it honors the request by responding to the NOTIFY with a *200 OK*.

> **NOTE**: For brevity, the authorization procedure is not shown in the diagram. The phone should always challenge an incoming NOTIFY for credentials that authorize the notifier to perform remote call control.

```
[F13] 200 OK IP Phone -> Application Server
SIP/2.0 200 OK
Via: SIP/2.0/UDP a1.foo.com:5060;branch=m9hG4bK74bf9
From: "Joe" <sip:3015551212@a1.foo.com>;tag=5fxced76s2
To: "Jane" <sip:3015551000@a1.foo.com> ; tag=432d33
Call-ID: 984827629822018851 2@a1.foo.com
CSeq: 444 NOTIFY
Contact: <sip:a1.foo.com:5060>
Content-Length: 0
```

The phone then performs the call-hold action.

7.3 Event Package Names

The event package name is called “hold”. It is carried in the *Event and Allow-Events* header, as defined in *RFC 3265* [3].

7.4 Event Package Parameters

This event package does not define any event package parameters.

7.5 SUBSCRIBE Bodies

The hold event package does not support dynamic subscriptions through SUBSCRIBE requests.

7.6 Subscription Duration

Subscriptions are implied by sessions established through INVITE requests and last throughout the duration of the session. When a UAC sends an INVITE to a UAS, it adds an *Allow-Events* header to the request, indicating all of the event packages it supports. This implicitly creates the subscription between the UAC and the UAS. When a UAS responds to the INVITE with an *18x* provisional response or *200 OK* response, it adds an *Allow-Events* header, indicating all of the event packages it supports. This implicitly creates the subscription between the UAS and the UAC. These subscriptions last until the session has been terminated with a BYE or CANCEL transaction.

7.7 NOTIFY Bodies

A NOTIFY for a hold package must not contain a body. No body is required as it has a trivial state.

7.8 Subscriber Generation of SUBSCRIBE Requests

Subscribers should not generate SUBSCRIBE requests. Subscriptions are granted to subscribers implicitly, by the establishment of sessions between two user agents.

7.9 Notifier Processing of SUBSCRIBE Requests

Notifiers should not process SUBSCRIBE requests. If received, they should be rejected.

7.10 Subscriber Processing of NOTIFY Requests

In general, subscriber processing of NOTIFY requests must follow all rules with respect to subscriber behavior, as described in *RFC 3265* [3]. The SIP specific event notification framework expects packages to specify how a subscriber processes NOTIFY requests and, in particular, how it uses the NOTIFY requests to construct a coherent view of the state of the subscribed resource.

Because these event notifications have the ability to remotely control the user's handset behavior, the subscriber (for example, the phone) should perform some level of authorization before processing NOTIFYs. The subscriber may perform implicit authorization, by looking at the source IP address of the NOTIFY request event and matching it to any one of the IP addresses of the notifier (in this case, the Application Server) as resolved through DNS. The subscriber may perform explicit authentication by challenging the notifier, by using a *401 Unauthorized* response. If the subscriber has also registered a location for this address of record, then the notifier should use those same credentials to answer the notification challenge.

When a subscriber receives a hold event notification for a particular dialog, it must apply the hold action to the local session in progress. If the session is in the *active* state, then it should be held. If it is in any other state, the request should be rejected.

Note that the subscriber should always respond to the NOTIFY transaction if it was received and before the action is complete. A *200 OK* response to a NOTIFY transaction only indicates that the notification has been received, and not that any action associated with the notification was successfully completed.

7.11 Handling of Forked Requests

SUBSCRIBEs and NOTIFYs should not be forked.

7.12 Rate of Notifications

The rate of notifications is dictated by the end-user interface on the call control client and can vary from application to application. Endpoints should be prepared to accept NOTIFY requests as frequently as once every two or three seconds.

7.13 Notifier Generation of NOTIFY Requests

Notifications are generated for the hold package whenever a remote call control application requests that an active session be held. Notifiers should not generate a hold event notification for a session that is already in the *held* state.

8 Application Server Feature Event Package

8.1 Overview

Many SIP phone users prefer to use the buttons on their phones to activate features, such as Do Not Disturb (DND), rather than through the Cisco BroadWorks web portal. This feature permits these SIP phone users to use the buttons on their phones in just this way.

Supported SIP phones can synchronize with the Cisco BroadWorks Application Server on the status of the following features:

- Do Not Disturb
- Call Forwarding Always (CFA)
- Call Forwarding Busy (CFB)
- Call Forwarding No Answer (CFNA)
- Call Center Agent ACD state
- Executive
- Executive-Assistant
- Security Classification
- Call Recording

If a user changes the status of one of these features via the web portal, a feature access code (FAC) or a Call Client, the Application Server notifies the phone about the status change. Conversely, if the user changes the feature status via a button on the user's phone, the phone notifies the Application Server of the status change request.

The synchronization protocol is based on the SIP-events framework. To use this capability, the phone user must have a SIP phone that supports the "as-feature-event" event package.

8.2 SIP Phone Behavior

When a SIP phone subscribes to the "as-feature-event" event package to control Cisco BroadWorks Application Server services, it must not attempt to offer these services itself. This behavior is mandatory to support the following scenarios:

Do Not Disturb

When Do Not Disturb is active, the Cisco BroadWorks Application Server can send an INVITE to the phone to play a ring reminder. This INVITE must cause the phone to ring. (It is incorrect for the phone to return a *486 Busy* in an attempt to implement the Do Not Disturb service.)

Call Forwarding (Always, Busy, No Answer)

- When Call Forwarding is active, the Cisco BroadWorks Application Server can send an INVITE to the phone to play a ring reminder. This INVITE must cause the phone to ring. (It is incorrect for the phone to return a *302 Moved Temporarily* or REFER in an attempt to implement the Call Forwarding service.)
- When Call Forwarding is active, Cisco BroadWorks Application Server can send an INVITE to the phone if the incoming call has the Diversion Inhibited property. This INVITE must cause the phone to ring. (It is incorrect for the phone to return a *302 Moved Temporarily* or REFER in an attempt to implement the Call Forwarding service.)

8.3 Example Message Flows

8.3.1 Initial SUBSCRIBE and NOTIFY

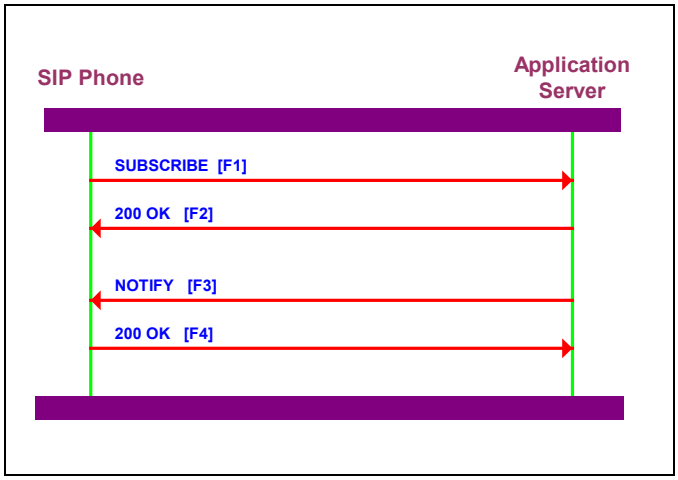


Figure 12 Initial SUBSCRIBE and NOTIFY

When a supported phone powers up, it sends a SUBSCRIBE request to the Application Server to create a subscription for the *as-feature-event* package as well as to get the full feature status. The following call flow shows the messages sent and received for this scenario.

To keep the flow simple, authentication is excluded. In most cases, the Application Server would require authentication from the phone in the SUBSCRIBE request.

In the first transaction, the phone sends a SUBSCRIBE request with an empty body to the Application Server, and receives a *200* response.

F1: Phone to Application Server

```
SUBSCRIBE  sip:2408881061@intas.broadworks.net SIP/2.0
Via:SIP/2.0/UDP 192.168.6.20;branch=z9hG4bK
From:<sip:2408881061@intas.broadworks.net>;tag=2006
To:"2408881061"<sip:2408881061@intas.broadworks.net>
Call-ID: 111111@10.10.1.21
CSeq: 8 SUBSCRIBE
Contact:<sip:2408881061@192.168.6.30>
Event: as-feature-event
Expires: 3600
Content-Length: 0
```

F2: Application Server to Phone

```
SIP/2.0 200 OK
From: <sip:2408881061@intas.broadworks.net>;tag=2006
To: "2408881061"<sip:2408881061@intas.broadworks.net>;tag=2007
Call-ID: 111111@10.10.1.21
CSeq: 8 SUBSCRIBE
Via: SIP/2.0/UDP 192.168.6.20;branch=z9hG4bK
Contact: <sip:intas.broadworks.net>
Event: as-feature-event
Expires: 3600
Content-Length: 0
```

Following this transaction, the Application Server sends a NOTIFY request to the phone with a body that contains the complete feature status, and the phone sends a *200* response. The body is a MIME "multi-part" body, containing an XML document for each feature assigned to the subscriber. If the size of NOTIFY is larger than 1500 bytes, it is recommended that the phone use Transmission Control Protocol (TCP) instead of User Datagram Protocol (UDP) as a transport mechanism. In the NOTIFY shown here, the subscriber has Do Not Disturb, Call Forwarding Always, Call Forwarding Busy, and Call Forwarding No Answer. The Application Server reports the status of all these features.

F3: Application Server to Phone

```
NOTIFY sip:2408881061@192.168.6.30 SIP/2.0
Via:SIP/2.0/UDP 192.168.6.20;branch=z9hG4bK
From:<sip:2408881061@intas.broadworks.net>;tag=2006
To:"2408881061"<sip:2408881061@intas.broadworks.net>;tag=2007
Call-ID: 111111@10.10.1.21
CSeq: 9 NOTIFY
Contact: <sip:intas.broadworks.net>
Max-Forwards: 10
Event: as-feature-event
Subscription-State: active, 3500
Content-Type: multipart/mixed; boundary=UniqueBroadWorksBoundary
Content-Length:2456

--UniqueBroadWorksBoundary
Content-Type:application/x-as-feature-event+xml
Content-Length:429
Content-ID:<sczbLo@broadworks>

<?xml version="1.0" encoding="ISO-8859-1"?>
<DoNotDisturbEvent xmlns="http://www.ecma-
international.org/standards/ecma-323/csta/ed3">
     <device>5559430902</device>
     <doNotDisturbOn>false</doNotDisturbOn>
</DoNotDisturbEvent>
--UniqueBroadWorksBoundary
Content-Type:application/x-as-feature-event+xml
Content-Length:458
Content-ID:<vZIKOJ@broadworks>

<?xml version="1.0" encoding="ISO-8859-1"?>
<ForwardingEvent xmlns="http://www.ecma-
international.org/standards/ecma-323/csta/ed3">
     <device>5559430902</device>
     <forwardingType>forwardImmediate</forwardingType>
     <forwardStatus>false</forwardStatus>
</ForwardingEvent>
--UniqueBroadWorksBoundary
Content-Type:application/x-as-feature-event+xml
Content-Length:463
Content-ID:<5U3iUD6@broadworks>

<?xml version="1.0" encoding="ISO-8859-1"?>
<ForwardingEvent xmlns="http://www.ecma-
international.org/standards/ecma-323/csta/ed3">
     <device>5559430902</device>
     <forwardingType>forwardBusy</forwardingType>
     <forwardStatus>false</activateForwardStatus>
     <forwardTo></forwardTo>
</ForwardingEvent>
--UniqueBroadWorksBoundary--
```

```
Content-Type:application/x-as-feature-event+xml
Content-Length:476
Content-ID:<5U3wsR@broadworks>

<?xml version="1.0" encoding="ISO-8859-1"?>
<ForwardingEvent xmlns="http://www.ecma-
international.org/standards/ecma-323/csta/ed3">
     <device>5559430902</device>
     <forwardingType>forwardNoAns</forwardingType>
     <forwardStatus>false</forwardStatus>
     <forwardTo></forwardTo>
     <ringCount></ringCount>
</ForwardingEvent>

--UniqueBroadWorksBoundary–
```

F4: Phone to Application Server

```
SIP/2.0 200 OK
From: <sip:2408881061@intas.broadworks.net>;tag=2006
To: "2408881061"<sip:2408881061@intas.broadworks.net>;tag=2007
Call-ID: 111111@10.10.1.21
CSeq: 9 NOTIFY
Via: SIP/2.0/UDP 192.168.6.20;branch=z9hG4bK
Contact: <sip:2408881061@192.168.6.30:5060;transport=UDP>
Event: as-feature-event
Content-Length: 0
```

8.3.2 Feature Status Update from Application Server to Phone

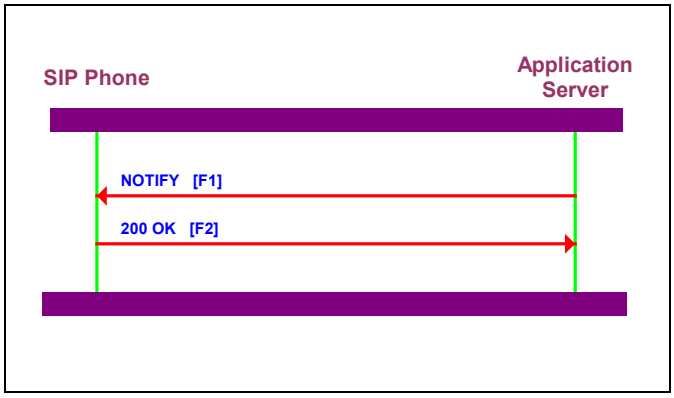


Figure 13 Feature Status Update from Application Server to Phone

When the user uses the web portal, an FAC or a client application to change the status of a synchronized feature, the Application Server sends a NOTIFY request to the phone to tell it about the changed status. The Application Server does not send the complete status information of all the features; it only sends the status of the changed feature. The phone sends a *200* response to complete the transaction.

In this particular example, the user activated Do Not Disturb.

F1: Application Server to Phone

```
NOTIFY sip:2408881061@192.168.6.30 SIP/2.0
Via:SIP/2.0/UDP 192.168.6.20;branch=z9hG4bK
From:<sip:2408881061@intas.broadworks.net>;tag=2006
```

```
To:"2408881061"<sip:2408881061@intas.broadworks.net>;tag=2007
Call-ID: 111111@10.10.1.21
CSeq: 10 NOTIFY
Contact: <sip:intas.broadworks.net>
Event: as-feature-event
Subscription-State: active, 2400
Content-type: application/x-as-feature-event+xml
Max-Forwards:10
Content-Length:200

<?xml version="1.0" encoding="ISO-8859-1"?>
<DoNotDisturbEvent xmlns="http://www.ecma-
international.org/standards/ecma-323/csta/ed3">
     <device>5559430902</device>
     <doNotDisturbOn>true</doNotDisturbOn>
</DoNotDisturbEvent>
```

F2: Phone to Application Server

```
SIP/2.0 200 OK
From: <sip:2408881061@intas.broadworks.net>;tag=2006
To: "2408881061"<sip:2408881061@intas.broadworks.net>;tag=20007
Call-ID: 111111@10.10.1.21
CSeq: 10 NOTIFY
Via: SIP/2.0/UDP 192.168.6.20;branch=z9hG4bK
Contact: <sip:2408881061@192.168.6.30:5060;transport=UDP>
Event: feature-event
Content-Length: 0
```

8.3.3 Feature Status Update from Phone to Application Server

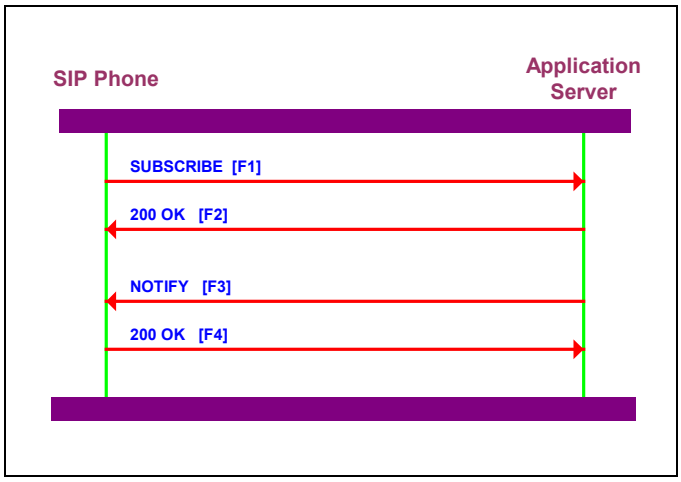


Figure 14 Feature Status Update from Phone to Application Server

When the user uses a button on the SIP phone to change the status of a synchronized feature, the phone sends a SUBSCRIBE request with a body to the Application Server to tell it about the changed feature status. The Application Server sends a *200* response to complete the transaction.

In this particular example, the user deactivated Do Not Disturb from the user's phone.

F1: Phone to Application Server

```
SUBSCRIBE  sip:2408881061@intas.broadworks.net SIP/2.0
Via:SIP/2.0/UDP 192.168.6.20;branch=z9hG4bK
From:<sip:2408881061@intas.broadworks.net>;tag=2006
To:"2408881061"<sip:2408881061@intas.broadworks.net>;tag=2007
Call-ID: 111111@10.10.1.21
CSeq: 11 SUBSCRIBE
Contact:<sip:2408881061@192.168.6.30>
Event: as-feature-event
Expires: 3600
Content-type: application/x-as-feature-event+xml
Content-Length: 120

<?xml version="1.0" encoding="ISO-8859-1"?>
<SetDoNotDisturb xmlns="http://www.ecma-
international.org/standards/ecma-323/csta/ed3">
     <device>5559430902</device>
     <doNotDisturbOn>false</doNotDisturbOn>
</SetDoNotDisturb>
```

F2: Application Server to Phone

```
SIP/2.0 200 OK
From: <sip:2408881061@intas.broadworks.net>;tag=2006
To: "2408881061"<sip:2408881061@intas.broadworks.net>;tag=2007
Call-ID: 111111@10.10.1.21
CSeq: 11 SUBSCRIBE
Via: SIP/2.0/UDP 192.168.6.20;branch=z9hG4bK
Contact: <sip:intas.broadworks.net>
Event: as-feature-event
Expires: 3600
Content-Length: 0
```

Immediately after the Application Server sends the *200* response, it sends a NOTIFY request with a body to tell the phone about the current status of the feature.

In most cases, the NOTIFY reflects the change just requested by the SUBSCRIBE request. However, in some cases, the NOTIFY request may indicate a status different from the SUBSCRIBE request. The phone must accept the status of the NOTIFY request if it is different from the SUBSCRIBE request.

F3: Application Server to Phone

```
NOTIFY sip:2408881061@192.168.6.30:5060 SIP/2.0
Via:SIP/2.0/UDP 192.168.6.20;branch=z9hG4bK
From:<sip:2408881061@intas.broadworks.net>;tag=2006
To:"2408881061"<sip:2408881061@intas.broadworks.net>;tag=2007
Call-ID: 111111@10.10.1.21
CSeq: 10 NOTIFY
Contact: <sip:intas.broadworks.net>
Event: as-feature-event
Subscription-State: active, 2400
Content-type: application/x-as-feature-event+xml
Max-Forwards:10
Content-Length:200

<?xml version="1.0" encoding="ISO-8859-1"?>
<DoNotDisturbEvent xmlns="http://www.ecma-
international.org/standards/ecma-323/csta/ed3">
```

```
    <device>5559430902</device>
    <doNotDisturbOn>true</doNotDisturbOn>
</DoNotDisturbEvent>
```

F8: Phone to Application Server

```
SIP/2.0 200 OK
From: <sip:2408881061@intas.broadworks.net>;tag=2006
To: "2408881061"<sip:2408881061@intas.broadworks.net>;tag=20007
Call-ID: 111111@10.10.1.21
CSeq: 10 NOTIFY
Via: SIP/2.0/UDP 192.168.6.20;branch=z9hG4bK
Contact: <sip:2408881061@192.168.6.30:5060;transport=UDP>
Event: feature-event
Content-Length: 0
```

8.4 Event Package Name

The name of this event package is “as-feature-event”. This name appears in the *Event* and *Allow-Events* header, as required by *RFC 6665* [3].

8.5 Event Package Parameters

This package does not define any event package parameters.

8.6 SUBSCRIBE Bodies

The message body in the SUBSCRIBE request is optional. The SIP phone may send a SUBSCRIBE request without a message body in order to refresh the subscription. Alternatively, the SIP phone may send a message body in order to change the status of a feature. The content of the message body depends on the specific feature being changed. However, in every case, the message body is an XML message and the declared media type is “application/x-as-feature-event”. The structure of the XML message is formally defined by an XML schema definition, which is one of the following:

- European Computer Manufacturers Association (ECMA) Computer Supported Telecommunications Applications (CSTA) XML schema definition, available for download from ECMA International [1]
- Cisco’s as-feature-event XML schema definition, which is included in this document

The following subsections describe the specific XML messages.

8.6.1 Set Do Not Disturb

The structure of the *Set Do Not Disturb* message is defined by the CSTA XML schema, an excerpt of which is provided as follows.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
    targetNamespace="http://www.ecma-international.org/standards/ecma-323/csta/ed3"
    xmlns:xsd=http://www.w3.org/2001/XMLSchema
    xmlns:csta="http://www.ecma-international.org/standards/ecma-323/csta/ed3"
    elementFormDefault="qualified"
    attributeFormDefault="unqualified">
  <xsd:annotation>
    <xsd:documentation>CSTA-set-do-not-disturb</xsd:documentation>
  </xsd:annotation>
  <xsd:include schemaLocation="device-identifiers.xsd"/>
  <xsd:include schemaLocation="device-feature-types.xsd"/>
  <xsd:include schemaLocation="extension-types.xsd"/>
```

```
  <xsd:element name="SetDoNotDisturb">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="device" type="csta:DeviceID"/>
        <xsd:element name="doNotDisturbOn" type="xsd:boolean"/>
        <xsd:element name="callOrigination" type="csta:CallOrigination"
          minOccurs="0"/>
        <xsd:element name="callingDeviceList" minOccurs="0">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="callingDevice" type="csta:DeviceID"
                minOccurs="0" maxOccurs="unbounded"/>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
        <xsd:element ref="csta:extensions" minOccurs="0"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

The SIP phone sends the *Set Do Not Disturb* message when changing the state of the Do Not Disturb feature.

Cisco BroadWorks supports the following fields:

- *device* – This is a mandatory field in the XML schema, but it is not used by Cisco BroadWorks. The phone can set this to any value.
- *doNotDisturbOn* – To activate Do Not Disturb, set to “true”. To deactivate Do Not Disturb, set to “false”.

8.6.2 Set Forwarding

The structure of the *Set Forwarding* message is defined by the CSTA XML schema, an excerpt of which is provided as follows.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
    targetNamespace="http://www.ecma-international.org/standards/ecma-323/csta/ed3"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:csta="http://www.ecma-international.org/standards/ecma-323/csta/ed3"
    elementFormDefault="qualified"
    attributeFormDefault="unqualified">
  <xsd:annotation>
    <xsd:documentation>CSTA-set-forwarding</xsd:documentation>
  </xsd:annotation>
  <xsd:include schemaLocation="device-identifiers.xsd"/>
  <xsd:include schemaLocation="device-feature-types.xsd"/>
  <xsd:include schemaLocation="extension-types.xsd"/>
  <xsd:element name="SetForwarding">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="device" type="csta:DeviceID"/>
        <xsd:element name="forwardingType" type="csta:ForwardingType"
          minOccurs="0"/>
        <xsd:element name="activateForward" type="xsd:boolean"/>
        <xsd:element name="forwardDN" type="csta:DeviceID" minOccurs="0"/>
        <xsd:element name="ringCount" minOccurs="0">
          <xsd:simpleType>
            <xsd:restriction base="xsd:long">
              <xsd:minInclusive value="1"/>
              <xsd:maxInclusive value="100"/>
            </xsd:restriction>
          </xsd:simpleType>
        </xsd:element>
        <xsd:element ref="csta:extensions" minOccurs="0"/>
      </xsd:sequence>
```

```
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="SetForwardingResponse">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="csta:extensions" minOccurs="0"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

The SIP phone sends the *Set Forwarding* message when changing the state or address of the Call Forwarding variants on Cisco BroadWorks (Call Forwarding Always, Call Forwarding Busy, and Call Forwarding No Answer).

Cisco BroadWorks supports the following fields:

- *device* – This is a mandatory field in the XML schema, but it is not used by Cisco BroadWorks. The phone can set this to any value.
- *forwardingType* – Cisco BroadWorks only supports the following forward types:
  - *forwardImmediate* – To set the Call Forwarding Always data on Cisco BroadWorks
  - *forwardBusy* – To set the Call Forwarding Busy data on Cisco BroadWorks
  - *forwardNoAns* – To set the Call Forwarding No Answer data on Cisco BroadWorks
- *activateForward* – To activate forwarding, set to “true”. To deactivate forwarding, set to “false”.
- *forwardDN* – This should be set to the forward address.
- *ringCount* – For Call Forwarding No Answer (CFNA), this element’s value indicates the number of rings allowed before the call should be forwarded. The SIP phone must provide a value for this element for CFNA; otherwise, the Application Server rejects the request.

8.6.3 Set Agent State

The structure of the *Set Agent State* message is defined by the CSTA XML schema, an excerpt of which is provided as follows.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
    targetNamespace="http://www.ecma-international.org/standards/ecma-323/csta/ed4"
    xmlns:xd="http://www.w3.org/2001/XMLSchema"
    xmlns:csta="http://www.ecma-international.org/standards/ecma-323/csta/ed4"
    elementFormDefault="qualified"
    attributeFormDefault="unqualified">
  <xsd:annotation>
    <xsd:documentation>CSTA-set-agent-state</xsd:documentation>
  </xsd:annotation>
  <xsd:include schemaLocation="device-identifiers.xsd"/>
  <xsd:include schemaLocation="device-feature-types.xsd"/>
  <xsd:include schemaLocation="extension-types.xsd"/>

  <xsd:element name="SetAgentState">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="device" type="csta:DeviceID"/>
        <xsd:element name="requestedAgentState" type="csta:ReqAgentState"/>
        <xsd:element name="agentID" type="csta:AgentID" minOccurs="0"/>
        <xsd:element name="password" type="csta:AgentPassword" minOccurs="0"/>
        <xsd:element name="group" type="csta:DeviceID" minOccurs="0"/>
```

```
        <xsd:element ref="csta:extensions" minOccurs="0"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:simpleType name="ReqAgentState">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="loggedOn"/>
      <xsd:enumeration value="loggedOff"/>
      <xsd:enumeration value="notReady"/>
      <xsd:enumeration value="ready"/>
      <xsd:enumeration value="workingAfterCall"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:schema>
```

The SIP phone sends the *Set Agent State* message when changing the *ACD* state of a Call Center Agent.

BroadWorks supports the following fields:

- *device* – This is a mandatory field in the XML schema, but it is not used by BroadWorks. The phone can set this to any value.
- *requestedAgentState* – Use one of the following values:
  - "loggedOn": Agent in the *Sign-In ACD* state
  - "loggedOff": Agent in the *Sign-Out ACD* state
  - "notReady": Agent in the *Unavailable ACD* state
  - "ready": Agent in the *Available ACD* state
  - workingAfterCall: *Agent in the Wrap-Up ACD* state

When the *requestedAgentState* is "notReady", the following optional extension is supported.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
    targetNamespace=http://schema.broadsoft.com/CSTAPrivateData
    xmlns:xsd=http://www.w3.org/2001/XMLSchema
    xmlns:pri=http://http://schema.broadsoft.com/CSTAPrivateData
    elementFormDefault="qualified"
    attributeFormDefault="unqualified">
  <xsd:annotation>
    <xsd:documentation>
       Private data schema for BroadSoft Call Center.
    </xsd:documentation>
  </xs:annotation>
  <xs:element name="AgentNotReadyReasonValue" type="xs:string"/>
</xs:schema>
```

The following is an example message body with the extension in use.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<SetAgentState
    xmlns="http://www.ecma-international.org/standards/ecma-323/csta/ed3">
  <device>5559430902</device>
  <requestedAgentState>notReady</requestedAgentState>
  <extensions>
    <privateData>
      <private xmlns:pri="http://schema.broadsoft.com/CSTAPrivateData">
        <pri:AgentNotReadyReasonValue>1
        </pri:AgentNotReadyReasonValue>
      </private>
    </privateData>
  </extensions>
```

```
</SetAgentState>
```

8.6.4 Set Executive Data

The structure of the *Set Executive Data* message is defined by the BroadWorks as-feature-event XML schema, an excerpt of which is provided as follows.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns="http://schema.broadsoft.com/as-feature-event"
    targetNamespace="http://schema.broadsoft.com/as-feature-event"
    elementFormDefault="qualified">

  <xs:element name="SetExecutiveData">
    <xs:annotation>
      <xs:documentation>
        Sets the Executive service data for the executive.
      </xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element name="callFilteringOn" type="xs:boolean">
          <xs:annotation>
            <xs:documentation>
              The call filtering status for the executive.
            </xs:documentation>
          </xs:annotation>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

The SIP phone sends the *Set Executive Data* message when changing the filtering status for BroadWorks' Executive service.

This message has one field:

- *callFilteringOn* – This is a Boolean field. If the value is "true", the SIP phone is requesting the Application Server to turn on executive filtering.

Following is an example of the *Set Executive Data* message.

```
<?xml version="1.0" encoding="utf-8"?>
<bw:SetExecutiveData xmlns:bw="http://schema.broadsoft.com/as-feature-event">
  <bw:callFilteringOn>false</bw:callFilteringOn>
</bw:SetExecutiveData>
```

8.6.5 Set Executive-Assistant Filtering Data

The structure of the *Set Executive-Assistant Filtering Data* message is defined by the BroadWorks as-feature-event XML schema, an excerpt of which is provided as follows.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns="http://schema.broadsoft.com/as-feature-event"
    targetNamespace="http://schema.broadsoft.com/as-feature-event"
    elementFormDefault="qualified">

  <xs:element name="SetExecutiveAssistantFilteringData">
    <xs:annotation>
      <xs:documentation>  Sets the Executive service's call filtering status for an
        executive the assistant is assigned to.
      </xs:documentation>
    </xs:annotation>
    <xs:complexType>
```

```
    <xs:sequence>
      <xs:element name="execUserId" type="xs:string">
        <xs:annotation>
          <xs:documentation>
            The user id of the executive.
          </xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="callFilteringOn" type="xs:boolean">
        <xs:annotation>
          <xs:documentation>
            The call filtering status for the executive.
          </xs:documentation>
        </xs:annotation>
      </xs:element>
     </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

The SIP phone sends the *Set Executive-Assistant Filtering Data* message when changing the filtering status of the Cisco BroadWorks Executive-Assistant service.

This message has the following fields:

- *execUserId* – The value of this field is the Cisco BroadWorks user ID of the executive user.
- *callFilteringOn* – This is a Boolean field. If the value is "true", the SIP phone is requesting the Application Server to turn on executive filtering.

Following is an example of the *Set Executive-Assistant Filtering Data* message.

```
<?xml version="1.0" encoding="utf-8"?>
<bw:SetExecutiveAssistantFilteringData
  xmlns:bw="http://schema.broadsoft.com/as-feature-event">
  <bw:execUserId>CC-Bob@callcenter.test</bw:execUserId>
  <bw:callFilteringOn>false</bw:callFilteringOn>
</bw:SetExecutiveAssistantFilteringData>
```

8.6.6 Set Executive-Assistant Divert Data

The structure of the *Set Executive-Assistant Divert Data* message is defined by the Cisco BroadWorks as-feature-event XML schema, an excerpt of which is provided as follows.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns="http://schema.broadsoft.com/as-feature-event"
    targetNamespace="http://schema.broadsoft.com/as-feature-event"
    elementFormDefault="qualified">

  <xs:element name="SetExecutiveAssistantDivertData">
    <xs:annotation>
      <xs:documentation>
        Sets the Executive-Assistant service's divert data for an
        assistant.
      </xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element name="divertStatus" type="xs:boolean">
          <xs:annotation>
            <xs:documentation>
              The divert status for the assistant.
            </xs:documentation>
          </xs:annotation>
```

```
        </xs:element>
        <xs:element name="divertAddr" type="xs:string" minOccurs="0">
          <xs:annotation>
            <xs:documentation>
              The divert phone number or URL for the assistant.
            </xs:documentation>
          </xs:annotation>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

The SIP phone sends the *Set Executive-Assistant Divert Data* message when changing the divert status of the Cisco BroadWorks Executive-Assistant service.

This message has the following fields:

- *divertStatus* – This is a Boolean field. If the value is "true", then the SIP phone is requesting the Application Server to enable diversion for the assistant.
- *divertAddr* – The value of this field is the destination for the diversion.

Following is an example of the *Set Executive-Assistant Divert Data* message.

```
<?xml version="1.0" encoding="utf-8"?>
<bw:SetExecutiveAssistantDivertData
  xmlns:bw="http://schema.broadsoft.com/as-feature-event">
  <bw:divertStatus>true</bw:divertStatus>
  <bw:divertAddr>2131001035</bw:divertAddr>
</bw:SetExecutiveAssistantDivertData>
```

8.6.7 Set Security Class

The structure of the *Set Security Class* message is defined by the Cisco BroadWorks as-feature-event XML schema, an excerpt of which is provided as follows.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns="http://schema.broadsoft.com/as-feature-event"
    targetNamespace="http://schema.broadsoft.com/as-feature-event"
    elementFormDefault="qualified">

  <xs:element name="SetSecurityClass">
    <xs:annotation>
      <xs:documentation>
        Set the user's current classification level used in classifying
        active calls.
      </xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element name="overrideLevel" type="xs:string">
          <xs:annotation>
            <xs:documentation>
              The classification level name to use for classifying
              active calls.
            </xs:documentation>
          </xs:annotation>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

The SIP phone sends the *Set Security Class* message when changing the override security level.

This message has the following field:

- *overrideLevel* – The value of this field is the security level chosen by the user.

Following is an example of the *Set Security Class* message.

```
<?xml version="1.0" encoding="utf-8"?>
<bw:SetSecurityClass xmlns:bw="http://schema.broadsoft.com/as-feature-event">
<bw:overrideLevel>Secret</bw:overrideLevel>
</bw:SetSecurityClass>
```

8.7 Subscription Duration

Subscribers should specify the duration of a subscription with an *Expires* header in the SUBSCRIBE request. If the notifier changes the expiration period to a lower value (via an *Expires* header in the final response), the subscriber should honor it. If the subscriber does not specify the duration of a subscription, then the notifier chooses the default expiration.

8.8 NOTIFY Bodies

The NOTIFY message is sent by the Application Server to the phone. The message body is optional.

The Application Server sends a NOTIFY message if it receives a SUBSCRIBE message or if one of the monitored feature states changes while the subscription is active. Each reported feature state appears in a body of type *application/x-as-feature-event*, containing an XML message as specified in reference [1].

For the initial notification, it is possible for multiple features to be reported in a single NOTIFY message. When this occurs, each feature status appears in its own body of type *application/x-as-feature-event*. The multiple *application/x-as-feature-event* bodies are then combined into a multipart/mixed body.

The following subsections provide the schema for each feature status reported.

8.8.1 Do Not Disturb Event

The structure of the *Do Not Disturb Event* message is defined by the CSTA XML schema, an excerpt of which is provided as follows.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
    targetNamespace="http://www.ecma-international.org/standards/ecma-323/csta/ed3"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:csta="http://www.ecma-international.org/standards/ecma-323/csta/ed3"
    elementFormDefault="qualified"
    attributeFormDefault="unqualified">
  <xsd:annotation>
    <xsd:documentation>CSTA-do-not-disturb-event</xsd:documentation>
  </xsd:annotation>
  <xsd:include schemaLocation="device-identifiers.xsd"/>
  <xsd:include schemaLocation="status-reporting.xsd"/>
  <xsd:include schemaLocation="device-feature-types.xsd"/>
  <xsd:include schemaLocation="extension-types.xsd"/>
  <xsd:element name="DoNotDisturbEvent">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="csta:monitorCrossRefID"/>
        <xsd:element name="device" type="csta:SubjectDeviceID"/>
        <xsd:element name="doNotDisturbOn" type="xsd:boolean"/>
        <xsd:element name="callOrigination" type="csta:CallOrigination"
```

```
          minOccurs="0"/>
        <xsd:element name="callingDeviceList" minOccurs="0">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="callingDevice" type="csta:DeviceID"
                minOccurs="0" maxOccurs="unbounded"/>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
        <xsd:element ref="csta:extensions" minOccurs="0"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

The Application Server sends the *Do Not Disturb Event* message to indicate the current state of the Do Not Disturb feature on Cisco BroadWorks.

The supported fields are:

- *device* – This is a mandatory field in the XML schema, but it is not used by Cisco BroadWorks. The phone should ignore any value it receives from the Application Server.
- *doNotDisturbOn* – To activate Do Not Disturb, set to “true”. To deactivate Do Not Disturb, set to “false”.

8.8.2 Forwarding Event

The structure of the *Forwarding Event* message is defined by the CSTA XML schema, an excerpt of which is provided as follows.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
    targetNamespace="http://www.ecma-international.org/standards/ecma-323/csta/ed3"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:csta="http://www.ecma-international.org/standards/ecma-323/csta/ed3"
    elementFormDefault="qualified"
    attributeFormDefault="unqualified">
  <xsd:annotation>
    <xsd:documentation>CSTA-forwarding-event</xsd:documentation>
  </xsd:annotation>
  <xsd:include schemaLocation="device-identifiers.xsd"/>
  <xsd:include schemaLocation="status-reporting.xsd"/>
  <xsd:include schemaLocation="device-feature-types.xsd"/>
  <xsd:include schemaLocation="extension-types.xsd"/>
  <xsd:element name="ForwardingEvent">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="csta:monitorCrossRefID"/>
        <xsd:element name="device" type="csta:SubjectDeviceID"/>
        <xsd:element name="forwardingType" type="csta:ForwardingType"
          minOccurs="0"/>
        <xsd:element name="forwardStatus" type="xsd:boolean"/>
        <xsd:element name="forwardTo" type="csta:DeviceID" minOccurs="0"/>
        <xsd:element name="forwardDefault" type="csta:ForwardDefault"
          minOccurs="0"/>
        <xsd:element name="ringCount" minOccurs="0">
          <xsd:simpleType>
            <xsd:restriction base="xsd:short">
              <xsd:minInclusive value="1"/>
              <xsd:maxInclusive value="100"/>
            </xsd:restriction>
          </xsd:simpleType>
        </xsd:element>
        <xsd:element ref="csta:extensions" minOccurs="0"/>
      </xsd:sequence>
    </xsd:complexType>
```

```
  </xsd:element>
</xsd:schema>
```

The Application Server sends the *Forwarding Event* message when changing the state or address of the Call Forwarding variants on Cisco BroadWorks (Call Forwarding Always, Call Forwarding Busy, and Call Forwarding No Answer).

The supported fields are:

- *device* – This is a mandatory field in the XML schema, but it is not used by Cisco BroadWorks. The phone should ignore any value it receives from the Application Server.
- *forwardingType* – Cisco BroadWorks only supports the following forward types:
  - *forwardImmediate* – For Call Forwarding Always data on Cisco BroadWorks
  - *forwardBusy* – For Call Forwarding Busy data on Cisco BroadWorks
  - *forwardNoAns* – For Call Forwarding No Answer data on Cisco BroadWorks
- *forwardStatus* – If forwarding is activated, set to "true". If forwarding is deactivated, set to "false".
- *forwardTo* – Set to the current forward address on Cisco BroadWorks.

8.8.3 Agent Logged Off Event

The structure of the *Agent Logged Off Event* message is defined by the CSTA XML schema, an excerpt of which is provided as follows.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
    targetNamespace="http://www.ecma-international.org/standards/ecma-323/csta/ed4"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:csta="http://www.ecma-international.org/standards/ecma-323/csta/ed4"
    elementFormDefault="qualified"
    attributeFormDefault="unqualified">
  <xsd:annotation>
    <xsd:documentation>CSTA-agent-logged-off-event</xsd:documentation>
  </xsd:annotation>
  <xsd:include schemaLocation="event-cause.xsd"/>
  <xsd:include schemaLocation="device-identifiers.xsd"/>
  <xsd:include schemaLocation="status-reporting.xsd"/>
  <xsd:include schemaLocation="device-feature-types.xsd"/>
  <xsd:include schemaLocation="extension-types.xsd"/>

  <xsd:element name="AgentLoggedOffEvent">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="csta:monitorCrossRefID"/>
        <xsd:element name="agentDevice" type="csta:SubjectDeviceID"/>
        <xsd:element name="agentID" type="csta:AgentID" minOccurs="0"/>
        <xsd:element name="acdGroup" type="csta:DeviceID" minOccurs="0"/>
        <xsd:element name="agentPassword" type="csta:AgentPassword" minOccurs="0"/>
        <xsd:element ref="csta:cause" minOccurs="0"/>
        <xsd:element ref="csta:extensions" minOccurs="0"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

The Application Server sends the *Agent Logged Off Event* message to report that the Call Center Agent ACD state is *Sign-Out*.

The supported field is *agentDevice*. This is a mandatory field in the XML schema, but it is not used by Cisco BroadWorks. The phone should ignore any value it receives from the Application Server.

8.8.4 Agent Logged On Event

The structure of the *Agent Logged On Event* message is defined by the CSTA XML schema, an excerpt of which is provided as follows.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
    targetNamespace="http://www.ecma-international.org/standards/ecma-323/csta/ed4"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:csta="http://www.ecma-international.org/standards/ecma-323/csta/ed4"
    elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xsd:annotation>
    <xsd:documentation>CSTA-agent-logged-on-event</xsd:documentation>
  </xsd:annotation>
  <xsd:include schemaLocation="event-cause.xsd"/>
  <xsd:include schemaLocation="device-identifiers.xsd"/>
  <xsd:include schemaLocation="status-reporting.xsd"/>
  <xsd:include schemaLocation="device-feature-types.xsd"/>
  <xsd:include schemaLocation="extension-types.xsd"/>

  <xsd:element name="AgentLoggedOnEvent">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="csta:monitorCrossRefID"/>
        <xsd:element name="agentDevice" type="csta:SubjectDeviceID"/>
        <xsd:element name="agentID" type="csta:AgentID" minOccurs="0"/>
        <xsd:element name="acdGroup" type="csta:DeviceID" minOccurs="0"/>
        <xsd:element name="agentPassword" type="csta:AgentPassword" minOccurs="0"/>
        <xsd:element ref="csta:cause" minOccurs="0"/>
        <xsd:element ref="csta:extensions" minOccurs="0"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

The Application Server sends the *Agent Logged On Event* message to report that the Call Center Agent ACD state is *Sign-In.*

The supported field is *agentDevice.* This is a mandatory field in the XML schema; however, it is not used by Cisco BroadWorks. The phone should ignore any value it receives from the Application Server.

8.8.5 Agent Not Ready Event

The structure of the *Agent Not Ready Event* message is defined by the CSTA XML schema, an excerpt of which is provided as follows.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
    targetNamespace="http://www.ecma-international.org/standards/ecma-323/csta/ed4"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:csta="http://www.ecma-international.org/standards/ecma-323/csta/ed4"
    elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xsd:annotation>
    <xsd:documentation>CSTA-agent-not-ready-event</xsd:documentation>
  </xsd:annotation>
  <xsd:include schemaLocation="event-cause.xsd"/>
  <xsd:include schemaLocation="device-identifiers.xsd"/>
  <xsd:include schemaLocation="status-reporting.xsd"/>
  <xsd:include schemaLocation="device-feature-types.xsd"/>
  <xsd:include schemaLocation="extension-types.xsd"/>

  <xsd:element name="AgentNotReadyEvent">
```

```
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="csta:monitorCrossRefID"/>
        <xsd:element name="agentDevice" type="csta:SubjectDeviceID"/>
        <xsd:element name="agentID" type="csta:AgentID" minOccurs="0"/>
        <xsd:element name="acdGroup" type="csta:DeviceID" minOccurs="0"/>
        <xsd:element ref="csta:cause" minOccurs="0"/>
        <xsd:element ref="csta:extensions" minOccurs="0"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

The Application Server sends the *Agent Not Ready Event* message to report that the Call Center Agent ACD state is *Unavailable*.

The supported field is *agentDevice*. This is a mandatory field in the XML schema, but it is not used by Cisco BroadWorks. The phone should ignore any value it receives from the Application Server.

The *Agent Not Ready Event* message may include the optional extension defined in section *8.6.3 Set Agent State*, as in the following example.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<AgentNotReadyEvent
    xmlns="http://www.ecma-international.org/standards/ecma-323/csta/ed3">
  <monitorCrossRefID></monitorCrossRefID>
  <device><notKnown/></device>
  <extensions>
    <privateData>
      <private xmlns:pri="http://schema.broadsoft.com/CSTAPrivateData">
        <pri:AgentNotReadyReasonValue>1</pri:AgentNotReadyReasonValue>
      </private>
    </privateData>
  </extensions>
</AgentNotReadyEvent>
```

8.8.6 Agent Ready Event

The structure of the *Agent Ready Event* message is defined by the CSTA XML schema, an excerpt of which is provided as follows.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
    targetNamespace="http://www.ecma-international.org/standards/ecma-323/csta/ed4"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:csta="http://www.ecma-international.org/standards/ecma-323/csta/ed4"
    elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xsd:annotation>
    <xsd:documentation>CSTA-agent-ready-event</xsd:documentation>
  </xsd:annotation>
  <xsd:include schemaLocation="event-cause.xsd"/>
  <xsd:include schemaLocation="device-identifiers.xsd"/>
  <xsd:include schemaLocation="status-reporting.xsd"/>
  <xsd:include schemaLocation="device-feature-types.xsd"/>
  <xsd:include schemaLocation="extension-types.xsd"/>

  <xsd:element name="AgentReadyEvent">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="csta:monitorCrossRefID"/>
        <xsd:element name="agentDevice" type="csta:SubjectDeviceID"/>
        <xsd:element name="agentID" type="csta:AgentID" minOccurs="0"/>
        <xsd:element name="acdGroup" type="csta:DeviceID" minOccurs="0"/>
        <xsd:element ref="csta:cause" minOccurs="0"/>
        <xsd:element ref="csta:extensions" minOccurs="0"/>
      </xsd:sequence>
```

```
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

The Application Server sends the *Agent Ready Event* message to report that the Call Center Agent ACD state is *Available.*

The supported field is *agentDevice*: This is a mandatory field in the XML schema; however, it is not used by Cisco BroadWorks. The phone should ignore any value it receives from the Application Server.

8.8.7 Agent Working After Call Event

The structure of the *Agent Working After Call Event* message is defined by the CSTA XML schema, an excerpt of which is provided as follows.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
    targetNamespace="http://www.ecma-international.org/standards/ecma-323/csta/ed4"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:csta="http://www.ecma-international.org/standards/ecma-323/csta/ed4"
    elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xsd:annotation>
    <xsd:documentation>CSTA-agent-working-after-call-event</xsd:documentation>
  </xsd:annotation>
  <xsd:include schemaLocation="event-cause.xsd"/>
  <xsd:include schemaLocation="device-identifiers.xsd"/>
  <xsd:include schemaLocation="status-reporting.xsd"/>
  <xsd:include schemaLocation="device-feature-types.xsd"/>
  <xsd:include schemaLocation="extension-types.xsd"/>

  <xsd:element name="AgentWorkingAfterCallEvent">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="csta:monitorCrossRefID"/>
        <xsd:element name="agentDevice" type="csta:SubjectDeviceID"/>
        <xsd:element name="agentID" type="csta:AgentID" minOccurs="0"/>
        <xsd:element name="acdGroup" type="csta:DeviceID" minOccurs="0"/>
        <xsd:element name="pendingAgentState" minOccurs="0">
          <xsd:simpleType>
            <xsd:restriction base="xsd:string">
              <xsd:enumeration value="notReady"/>
              <xsd:enumeration value="ready"/>
              <xsd:enumeration value="null"/>
            </xsd:restriction>
          </xsd:simpleType>
        </xsd:element>
       <xsd:element ref="csta:cause" minOccurs="0"/>
       <xsd:element ref="csta:extensions" minOccurs="0"/>
     </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

The Application Server sends the *Agent Working After Call Event* message to report that the Call Center Agent ACD state is *Wrap-Up.*

The supported field is *agentDevice*. This is a mandatory field in the XML schema; however, it is not used by Cisco BroadWorks. The phone should ignore any value it receives from the Application Server.

8.8.8 Executive Event

The structure of the *Executive Event* message is defined by the Cisco BroadWorks as-feature-event XML schema, an excerpt of which is provided as follows.

```
  <xs:element name="ExecutiveEvent">
    <xs:annotation>
      <xs:documentation>
         Event sent to an executive with the Executive service data:
          - when the call filtering status is modified.
          - when the Executive service is assigned to or unassigned from an
executive.
      </xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element name="callFilteringOn" type="xs:boolean">
          <xs:annotation>
            <xs:documentation>
              The call filtering status for the executive.
            </xs:documentation>
          </xs:annotation>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
```

The Application Server sends the *Executive Event* message to notify the SIP phone of the filtering status for the executive in the Cisco BroadWorks Executive service.

Following is an example of the *Executive Event* message.

```
<?xml version="1.0" encoding="utf-8"?>
<bw:ExecutiveEvent xmlns:bw="http://schema.broadsoft.com/as-feature-event">
  <bw:callFilteringOn>false</bw:callFilteringOn>
</bw:ExecutiveEvent>
```

8.8.9 Executive-Assistant Event

The structure of the *Executive-Assistant Event* message is defined by the Cisco BroadWorks as-feature-event XML schema, an excerpt of which is provided as follows.

```
  <xs:element name="ExecutiveAssistantEvent">
    <xs:annotation>
      <xs:documentation>
        Event sent to an assistant with the Executive-Assistant
        service data and the call filtering status for the executives the
        assistant is assigned to:
          - when the divert status or divert address is modified.
          - when the Executive-Assistant service is assigned to or
            unassigned from an assistant.
          - when the call filtering status is modified for an executive
            the assistant is assigned to.
          - when the user ID or name is modified for an executive the
            assistant is assigned to.
          - when the assistant is assigned to or unassigned from an
            executive.
      </xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element name="divertStatus" type="xs:boolean">
          <xs:annotation>
            <xs:documentation>
              The divert status for the assistant.
            </xs:documentation>
          </xs:annotation>
        </xs:element>
        <xs:element name="divertAddr" type="xs:string" minOccurs="0">
          <xs:annotation>
            <xs:documentation>
              The divert phone number or URL for the assistant.
```

```
          </xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="execDataList" type="ExecutiveDataList" minOccurs="0">
        <xs:annotation>
          <xs:documentation>
            The information for the list of executives the assistant is
            assigned to.
          </xs:documentation>
        </xs:annotation>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>


<xs:complexType name="ExecutiveData">
  <xs:sequence>
    <xs:element name="execUserId" type="xs:string">
      <xs:annotation>
        <xs:documentation>
          The user id of the executive.
        </xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="callFilteringOn" type="xs:boolean">
      <xs:annotation>
        <xs:documentation>
          The call filtering status for the executive.
        </xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="execIds" type="ExecutiveIdentifier">
      <xs:annotation>
        <xs:documentation>
          The identifiers like name, DN and extn for the executive.
        </xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="ExecutiveIdentifier">
  <xs:sequence>
    <xs:element name="execName" type="xs:string">
      <xs:annotation>
        <xs:documentation>
          The name of the executive.
        </xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="execDN" type="xs:string" minOccurs="0">
      <xs:annotation>
        <xs:documentation>
          The primary DN for the executive.
        </xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="execExtn" type="xs:string" minOccurs="0">
      <xs:annotation>
        <xs:documentation>
          The primary extension for the executive.
        </xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="ExecutiveDataList">
  <xs:sequence>
```

```
      <xs:element name="execData" type="ExecutiveData" maxOccurs="unbounded">
        <xs:annotation>
          <xs:documentation>
            The data for an executive in the list.
          </xs:documentation>
        </xs:annotation>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
```

The Application Server sends the *Executive-Assistant Event* message to notify the SIP phone of the status of the assistant in Cisco BroadWorks' Executive-Assistant service.

Following is an example of the *Executive-Assistant Event* message.

```
<?xml version="1.0" encoding="utf-8"?>
<bw:ExecutiveAssistantEvent
    xmlns:bw="http://schema.broadsoft.com/as-feature-event">
  <bw:divertStatus>false</bw:divertStatus>
  <bw:divertAddr>2131001045</bw:divertAddr>
  <bw:execDataList>
    <bw:execData>
      <bw:execUserId>CC-Alice@callcenter.test</bw:execUserId>
      <bw:callFilteringOn>false</bw:callFilteringOn>
      <bw:execIds>
        <bw:execName>Alice Smith</bw:execName>
        <bw:execDN>2131001001</bw:execDN>
        <bw:execExtn>1001</bw:execExtn>
      </bw:execIds>
    </bw:execData>
  </bw:execDataList>
</bw:ExecutiveAssistantEvent>
```

8.8.10 Security Class Event

The structure of the *Security Class Event* message is defined by the Cisco BroadWorks as-feature-event XML schema, an excerpt of which is provided as follows.

```
  <xs:element name="SecurityClassEvent">
    <xs:annotation>
      <xs:documentation>Security Classification Event</xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element name="allowedLevels" type="LevelList" nillable="true">
          <xs:annotation>
              <xs:documentation>
                The list of security classification levels available for
                the user for updating its current security classification.
                The element is nil if the service is removed from the user.
              </xs:documentation>
            </xs:annotation>
        </xs:element>
        <xs:element name="overrideLevel" type="xs:string" nillable="true">
            <xs:annotation>
              <xs:documentation>
                The user override classification level.
                The element is nil if the user has not overrriden its assigned
                classification level.
              </xs:documentation>
            </xs:annotation>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:annotation>
    <xs:documentation>List of security classification levels. </xs:documentation>
```

```
  </xs:annotation>
  <xs:complexType name="LevelList">
    <xs:sequence>
      <xs:element name="level" type="xs:string" minOccurs="0"
            maxOccurs="unbounded">
          <xs:annotation>
            <xs:documentation>
              The security classification level name.
            </xs:documentation>
          </xs:annotation>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
```

The Application Server sends the *Security Class Event* message to notify the SIP phone of the security levels defined in Cisco BroadWorks and of the override classification level, if any.

Following is an example of the *Security Class Event* message.

```
<?xml version="1.0" encoding="utf-8"?>
<bw:SecurityClassEvent
    xmlns:bw="http://schema.broadsoft.com/as-feature-event"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <bw:allowedLevels>
    <bw:level>Top Secret</bw:level>
    <bw:level>Secret</bw:level>
    <bw:level>Restricted</bw:level>
    <bw:level>Unclassified</bw:level>
  </bw:allowedLevels>
  <bw:overrideLevel xsi:nil="true"></bw:overrideLevel>
</bw:SecurityClassEvent>
```

8.8.11 Call Recording Mode Event

The structure of the *Call Recording Mode Event* message is defined by the Cisco BroadWorks as-feature-event XML schema, an excerpt of which is provided as follows.

```
  <xs:annotation>
    <xs:documentation>Call Recording Mode Event</xs:documentation>
  </xs:annotation>
  <xs:element name="CallRecordingModeEvent">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="callRecordingMode" type="RecordingMode" nillable="true">
            <xs:annotation>
              <xs:documentation>
                The provisioned call recording mode for the user.
                The element is nil if the user does not have the service.
              </xs:documentation>
            </xs:annotation>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:simpleType name="RecordingMode">
    <xs:annotation>
      <xs:documentation>
        The recording mode of the Call Recording user:
        ALWAYS indicates calls originated by or received by the user are always
recorded and saved.
        ALWAYS-PAUSE-RESUME indicates that calls originated or received by the user
are always recorded and saved. The user also has the ability to pause and resume
recordings.
        ON-DEMAND indicates that calls can be selectively recorded and saved on an
on-demand basis. The user also has the ability to pause and resume recordings.
```

```
        ON-DEMAND-USER-START indicates that the call recording does not start until
initiated by the user. The user also has the ability to pause, resume and stop
recordings.
        NEVER indicates calls are never recorded.
      </xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:token">
      <xs:enumeration value="always" />
      <xs:enumeration value="always-pause-resume"/>
      <xs:enumeration value="on-demand" />
      <xs:enumeration value="on-demand-user-start"/>
      <xs:enumeration value="never" />
    </xs:restriction>
  </xs:simpleType>
```

The Application Server sends the *Call Recording Mode Event* message to notify the SIP phone of the current call recording mode for the user.

Following is an example of the *Call Recording Mode Event* message.

```
<?xml version="1.0" encoding="utf-8"?>
<bw:CallRecordingModeEvent xmlns:bw="http://schema.broadsoft.com/as-feature-event">
  <bw:callRecordingMode>always</bw:callRecordingMode>
</bw:CallRecordingModeEvent>
```

8.9 Subscriber Generation of SUBSCRIBE Requests

Subscriber (SIP phone) generation of SUBSCRIBE requests must follow all rules with respect to subscriber behavior, as described in *RFC 3265*.

SIP phones should generate SUBSCRIBE requests immediately after successfully registering. SUBSCRIBE requests should also be generated when the subscription approaches the expiration time. These SUBSCRIBE messages typically do not contain a message body.

SIP phones should also generate SUBSCRIBE requests with an *application/x-as-feature-event+xml* message body to configure a feature state. The SIP phone must not consider a *200* response from the Application Server as an acknowledgement that the feature state has changed. The *200* response from the Application Server is merely an indication that the subscription was accepted. The phone must wait for the subsequent NOTIFY message to synchronize with the actual feature state on the Application Server, which may or not be the state requested in the SUBSCRIBE request.

8.10 Notifier Processing of SUBSCRIBE Requests

Notifier (Application Server) processing of SUBSCRIBE requests must follow all rules with respect to notifier behavior, as described in *RFC 3265*. The notifier may choose to explicitly challenge the subscriber for authentication by using a *401 Unauthorized* response. If the subscriber has also registered a location for this address of record, then it should use the same credentials to answer the subscription challenge.

8.11 Notifier Generation of NOTIFY Requests

Notifier (Application Server) generation of NOTIFY requests must follow all rules with respect to notifier behavior, as described in *RFC 3265*.

If the SUBSCRIBE request contains a request to set a feature state, the Notifier may ignore the request to set the feature state:

- If the request may not be processed as this time, for example, if the database is locked, then the feature state is not changed and the existing feature state is reported in the NOTIFY message.
- If the user does not have the feature, a NOTIFY message with no message body is sent.
- If the SUBSCRIBE request contains no message body, the Notifier sends a NOTIFY message with all the feature states.

8.12 Subscriber Processing of NOTIFY Requests

In general, subscriber processing of NOTIFY requests must follow all rules with respect to subscriber behavior, as described in *RFC 3265*. The SIP-specific event notification framework expects packages to specify how a subscriber processes NOTIFY requests, and in particular, how it uses the NOTIFY requests to construct a coherent view of the state of the subscribed resource.

The NOTIFY messages received in response to SUBSCRIBE requests with no message body contain the complete set of feature status. These SUBSCRIBEs are typically the initial SUBSCRIBE message and subsequent SUBSCIBE refresh messages. When the corresponding NOTIFY message is received, the subscriber must discard any previous state information and replace it with the received information.

The NOTIFY messages received in response to SUBSCRIBE requests with a message body and the NOTIFY messages received following feature state change not related to a SUBSCRIBE message, contain only the feature state of the affected feature. The subscriber must replace the state of this feature with the new received state, but must not modify the state of any other feature.

During the life of the subscription, it is possible for features to be added or removed. The subscriber must be ready to add new features on any NOTIFY message. It must also be ready to remove features when NOTIFY messages with complete states are received.

8.13 Rate of Notifications

The rate of notifications is entirely dependent on the frequency at which the user feature state changes. Typically, this occurs at most a few times per minute.

8.14 Shared Lines

Cisco BroadWorks implements shared lines via the Shared Call Appearance feature. Shared Call Appearance allows a Cisco BroadWorks subscriber to have a primary phone and zero or more secondary phones. One common arrangement of shared lines is the executive/assistant arrangement, as shown in *Figure 15 Executive/Assistant Arrangement*.

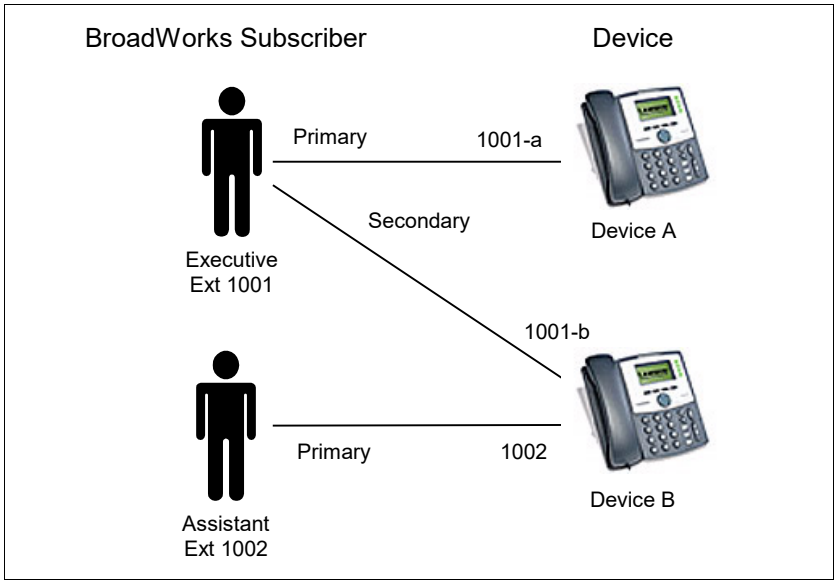


Figure 15 Executive/Assistant Arrangement

Feature status in Cisco BroadWorks is maintained according to the subscriber. This means that all Shared Call Appearance devices share the same feature status. In the executive/assistant case, if the executive were to enable Do Not Disturb, then incoming calls would get Do Not Disturb treatment and the assistant's phone would not ring. If feature synchronization were enabled, then both the executive's phone and the assistant's phone would show Do Not Disturb enabled for extension 1001.

To summarize the situation with shared lines, there may be only one subscription for each line-side address of record (AoR), that is, line/port. Because feature status is maintained according to the Cisco BroadWorks subscriber, all subscriptions for a subscriber's primary and secondary devices share the same feature status. In the arrangement shown in *Figure 15 Executive/Assistant Arrangement*, there may be three subscriptions, since there are three line-side AoRs (1001-a, 1001-b, and 1002). Two subscriptions, those for 1001-a and 1001-b, share the same feature status, which is the status of the executive subscriber. The Shared Line feature update is optional on phone.

NOTE: When a phone is managing multiple private lines, the phone can implement a Do Not Disturb and Call Forwarding state on a per-line basis or the Do Not Disturb and Call Forwarding state settings can apply to all lines configured on the phone. If the phone is synchronizing all lines on the phone, then the phone must ensure that all lines are synchronized when changes are made to one of the lines. When multiple lines are in use, each of the lines must subscribe to the as-feature-event package. If the Do Not Disturb feature is activated on Cisco BroadWorks for User 1, Cisco BroadWorks sends a NOTIFY message to the phone with the *doNotDisturbOn* field set to "true". The phone then detects a status change for the Do Not Disturb feature on line 1. The phone must then send a re-subscribe to Cisco BroadWorks for each of the other lines on the phone. This re-subscribe must include the Do Not Disturb feature event with the *doNotDisturbOn* field set to "true".

8.15 Cisco as-feature-event XML Schema Definition

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns="http://schema.broadsoft.com/as-feature-event"
  targetNamespace="http://schema.broadsoft.com/as-feature-event"
  elementFormDefault="qualified">

  <!--
    ****************************************
    *  Executive Event
    ****************************************
  -->
  <xs:element name="ExecutiveEvent">
    <xs:annotation>
      <xs:documentation>
         Event sent to an executive with the Executive service data:
          - when the call filtering status is modified.
          - when the Executive service is assigned to or unassigned from an
executive.
      </xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element name="callFilteringOn" type="xs:boolean">
          <xs:annotation>
            <xs:documentation>
              The call filtering status for the executive.
            </xs:documentation>
          </xs:annotation>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <!--
    ****************************************
    *  Executive-Assistant Event
    ****************************************
  -->
  <xs:element name="ExecutiveAssistantEvent">
    <xs:annotation>
      <xs:documentation>
        Event sent to an assistant with the Executive-Assistant
        service data and the call filtering status for the executives the
        assistant is assigned to:
          - when the divert status or divert address is modified.
          - when the Executive-Assistant service is assigned to or
            unassigned from an assistant.
          - when the call filtering status is modified for an executive
            the assistant is assigned to.
          - when the user ID or name is modified for an executive the
            assistant is assigned to.
          - when the assistant is assigned to or unassigned from an
            executive.
      </xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element name="divertStatus" type="xs:boolean">
          <xs:annotation>
            <xs:documentation>
              The divert status for the assistant.
            </xs:documentation>
          </xs:annotation>
        </xs:element>
        <xs:element name="divertAddr" type="xs:string" minOccurs="0">
          <xs:annotation>
            <xs:documentation>
              The divert phone number or URL for the assistant.
```

```
          </xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="execDataList" type="ExecutiveDataList" minOccurs="0">
        <xs:annotation>
          <xs:documentation>
            The information for the list of executives the assistant is
            assigned to.
          </xs:documentation>
        </xs:annotation>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:complexType name="ExecutiveData">
  <xs:sequence>
    <xs:element name="execUserId" type="xs:string">
      <xs:annotation>
        <xs:documentation>
          The user id of the executive.
        </xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="callFilteringOn" type="xs:boolean">
      <xs:annotation>
        <xs:documentation>
          The call filtering status for the executive.
        </xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="execIds" type="ExecutiveIdentifier">
      <xs:annotation>
        <xs:documentation>
          The identifiers like name, DN and extn for the executive.
        </xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="ExecutiveIdentifier">
  <xs:sequence>
    <xs:element name="execName" type="xs:string">
      <xs:annotation>
        <xs:documentation>
          The name of the executive.
        </xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="execDN" type="xs:string" minOccurs="0">
      <xs:annotation>
        <xs:documentation>
          The primary DN for the executive.
        </xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="execExtn" type="xs:string" minOccurs="0">
      <xs:annotation>
        <xs:documentation>
          The primary extension for the executive.
        </xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="ExecutiveDataList">
  <xs:sequence>
    <xs:element name="execData" type="ExecutiveData" maxOccurs="unbounded">
```

```
        <xs:annotation>
          <xs:documentation>
            The data for an executive in the list.
          </xs:documentation>
        </xs:annotation>
      </xs:element>
    </xs:sequence>
  </xs:complexType>

  <!--
    ****************************************
    *  Security Class Event
    ****************************************
  -->
  <xs:element name="SecurityClassEvent">
    <xs:annotation>
      <xs:documentation>Security Classification Event</xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element name="allowedLevels" type="LevelList" nillable="true">
          <xs:annotation>
              <xs:documentation>
                The list of security classification levels available for
                the user for updating its current security classification.
                The element is nil if the service is removed from the user.
              </xs:documentation>
            </xs:annotation>
        </xs:element>
        <xs:element name="overrideLevel" type="xs:string" nillable="true">
            <xs:annotation>
              <xs:documentation>
                The user override classification level.
                The element is nil if the user has not overrriden its
                assigned classification level.
              </xs:documentation>
            </xs:annotation>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:annotation>
    <xs:documentation>List of security classification levels. </xs:documentation>
  </xs:annotation>
  <xs:complexType name="LevelList">
    <xs:sequence>
      <xs:element name="level" type="xs:string" minOccurs="0"
maxOccurs="unbounded">
          <xs:annotation>
            <xs:documentation>
              The security classification level name.
            </xs:documentation>
          </xs:annotation>
      </xs:element>
    </xs:sequence>
  </xs:complexType>

  <!--
    ****************************************
    *  Call Recording Mode Event
    ****************************************
  -->
  <xs:annotation>
    <xs:documentation>Call Recording Mode Event</xs:documentation>
  </xs:annotation>
  <xs:element name="CallRecordingModeEvent">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="callRecordingMode" type="RecordingMode" nillable="true">
```

```
          <xs:annotation>
            <xs:documentation>
              The provisioned call recording mode for the user.
              The element is nil if the user does not have the service.
            </xs:documentation>
          </xs:annotation>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:simpleType name="RecordingMode">
    <xs:annotation>
      <xs:documentation>
        The recording mode of the Call Recording user:

        ALWAYS indicates calls originated by or received by the user are
        always recorded and saved.

        ALWAYS-PAUSE-RESUME indicates that calls originated or received by
        the user are always recorded and saved. The user also has the
        ability to pause and resume recordings.

        ON-DEMAND indicates that calls can be selectively recorded and
        saved on an on-demand basis. The user also has the ability to pause
        and resume recordings.

        ON-DEMAND-USER-START indicates that the call recording does not
        start until initiated by the user. The user also has the ability to
        pause, resume and stop recordings.

        NEVER indicates calls are never recorded.
      </xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:token">
      <xs:enumeration value="always" />
      <xs:enumeration value="always-pause-resume"/>
      <xs:enumeration value="on-demand" />
      <xs:enumeration value="on-demand-user-start"/>
      <xs:enumeration value="never" />
    </xs:restriction>
  </xs:simpleType>

  <!--
    ****************************************
    *  Set Executive Data
    ****************************************
  -->
  <xs:element name="SetExecutiveData">
    <xs:annotation>
      <xs:documentation>
        Sets the Executive service data for the executive.
      </xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element name="callFilteringOn" type="xs:boolean">
          <xs:annotation>
            <xs:documentation>
              The call filtering status for the executive.
            </xs:documentation>
          </xs:annotation>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <!--
    ****************************************
    *  Set Executive-Assistant Filtering Data
    ****************************************
```

```
-->
<xs:element name="SetExecutiveAssistantFilteringData">
  <xs:annotation>
    <xs:documentation>
      Sets the Executive service's call filtering status for an executive
      the assistant is assigned to.
    </xs:documentation>
  </xs:annotation>
  <xs:complexType>
  <xs:sequence>
    <xs:element name="execUserId" type="xs:string">
      <xs:annotation>
        <xs:documentation>
          The user id of the executive.
        </xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="callFilteringOn" type="xs:boolean">
      <xs:annotation>
        <xs:documentation>
          The call filtering status for the executive.
        </xs:documentation>
      </xs:annotation>
    </xs:element>
   </xs:sequence>
  </xs:complexType>
</xs:element>

<!--
  ****************************************
  *  Set Executive-Assistant Divert Data
  ****************************************
-->
<xs:element name="SetExecutiveAssistantDivertData">
  <xs:annotation>
    <xs:documentation>
      Sets the Executive-Assistant service's divert data for an
      assistant.
    </xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element name="divertStatus" type="xs:boolean">
        <xs:annotation>
          <xs:documentation>
            The divert status for the assistant.
          </xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="divertAddr" type="xs:string" minOccurs="0">
        <xs:annotation>
          <xs:documentation>
            The divert phone number or URL for the assistant.
          </xs:documentation>
        </xs:annotation>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<!--
  ****************************************
  *  Set Security Class
  ****************************************
-->
<xs:element name="SetSecurityClass">
  <xs:annotation>
    <xs:documentation>
      Set the user's current classification level used in classifying
      active calls.
```

```
      </xs:documentation>
      </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element name="overrideLevel" type="xs:string">
          <xs:annotation>
            <xs:documentation>
              The classification level name to use for classifying
              active calls.
            </xs:documentation>
          </xs:annotation>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

</xs:schema>
```

9 Alternate Signaling for Shared Call Appearance

Section *3 Call-Info Header Extensions* describes extensions to the *Call-Info* header to support Shared Call Appearance. This section describes an alternative signaling method that uses the dialog event notification package and the *Join/Replaces* SIP headers to achieve similar behaviors. This allows devices that do not support the Call-Info extensions, but do support the signaling that follows to provide enhanced capabilities for Shared Call Appearance.

The dialog event package is used to communicate appearance status to all devices.

- The SIP *Join* header, as defined in *RFC 3911* [4], is used to create a conference between SCA locations, also called Shared Call Appearance Bridging.
- The SIP *Replaces* header, as defined in *RFC 3891* [5], is used to retrieve calls between SCA locations.

The Cisco BroadWorks Shared Call Appearance service can also be optionally configured to provide Call Park information similarly to the Call Park Event Package (section *12 Call Park Event Package*) but without the requirement for a separate subscription.

9.1 Dialog Event Package

An SCA location receives dialog state information through subscriptions to the Dialog event package, as defined by *RFC 4235* [6]. Such devices are referred as “SCA location dialog subscribers”.

9.1.1 application/dialog-info+xml Message Body

The dialog event package reports the dialog state using the application/dialog-info+xml message body. The application/dialog-info+xml body contains a <dialog> element for each SIP dialog (or half-dialog if not fully specified yet) with the following information. The description of the message body is found in *RFC 4235* [6]. In the following table, the M/O indicates whether this element or attribute is mandatory (M) or optional (O).

Element/Attribute	M/O	Value format	Note	Example
id attribute	M	String	Generated by the Application Server according to *RFC 4235* [6].	id=“cm9ibk1C”
call-id attribute	O	String	Corresponding SIP dialog Call-ID. Single and double quotes are escaped with ' and " respectively. Only provided for “confirmed” dialogs.	call-id=“9b8e3369-b7f61357-ae78db88@192.168.22.75”
local-tag attribute	O	String	SIP tag of the party related to the Shared Call Appearance device for the SIP dialog between the SCA location and the Application Server (that is, *From* tag when SCA location originates a call; *To* tag when SCA location receives a call). Only provided for “confirmed” dialogs.	local-tag=”2062346240-1234474019093”

Element/Attribute	M/O	Value format	Note	Example
remote-tag attribute	O	String	SIP tag of the party related to the SCA location remote party on the SIP dialog between the SCA location and the Application Server (that is, the *To* tag when the SCA location originates a call, and the *From* tag when the SCA location receives a call). Only provided for “confirmed” dialogs.	remote-tag=”6C477E05-CC90A1D6”
direction attribute	M	One of “initiator”, “recipient”	Set to “initiator” if the SCA location initiated the SIP dialog; otherwise, set to “recipient”.	direction=”initiator”
<state> element	M	One of “proceeding”, “confirmed”, “terminated”	Indicates the SIP dialog state. The Application Server sends the *proceeding* state when a dialog is first created and before it is confirmed. The Application Server sends the *confirmed* state when it receives a 200-class response for the SIP INVITE request. The Application Server sends the *terminated* state when it receives a BYE for the SIP dialog. There is at most one notification for which a dialog is indicated as “terminated”. The Application Server does not send the *trying* or *early* states.	<state>confirmed</state>
<local> element	O	Contains an <identity> and optionally a <target> nested elements	Contains Shared Call Appearance user information. This does not correspond to the SIP dialog local information, but rather to user information managed by the Application Server. If present, the <target> is the Shared Call Appearance device contact from the SIP dialog.	
<remote> element	O	Contains an <identity> element.	Contains Shared Call Appearance user remote party information. This does not correspond to SIP dialog remote information, but rather the remote party information managed by the Application Server (evaluated against presentation services).	
<identity> element	O	URI	Optionally contains the *display* attribute. A <local> or <remote> element may contain 0 or 1 <identity> element.	<identity display="John Smith"> sip:5145551212@as.com </identity>
<target> element	O	URI	Contains the SCA location contact. Only included in the <local> element.	<target uri=”sip:5145551212@192.168.0.1”/>

Element/Attribute	M/O	Value format	Note	Example
<param> element	O	pname="+sip.rendering" pval="no"	Provided only when <target> is reported and the call appearance associated with the SIP dialog is either in *held, held-private*, or *bridge-held* states.	<param pname="+sip.rendering" pval="no"/>
<sa:appearance> element	O	Non-negative integer	From *RFC 7463* [9]. Corresponds to the appearance index associated with the SIP dialog. Many SIP dialogs may be associated to the same appearance (for example, SCA bridges).	<sa:appearance> 0 </sa:appearance>

Example message body

```
<?xml version="1.0" encoding="UTF-8"?>
<dialog-info
  xmlns="urn:ietf:params:xml:ns:dialog-info"
  xmlns:sa="urn:ietf:params:xml:ns:sa-dialog-info"
  version="2" state="partial" entity="sip:jsmith@as.com">
  <dialog id="SjRiWml4"
          direction="initiator"
          call-id="abc@192.168.0.1"
          local-tag="123"
          remote-tag="456">
    <state>confirmed</state>
    <local>
      <identity display="John Smith">sip:jsmith@as.com</identity>
      <target uri="sip:5145551212@192.168.0.1">
        <param pname="+sip.rendering" pval="no"/>
      </target>
    </local>
    <remote>
      <identity display="Alice Smith">sip:5145556666;user=phone</identity>
    </remote>
    <sa:appearance>0</sa:appearance>
  </dialog>
</dialog-info>
```

The Application Server sends "full" dialog information (*dialog-info state* attribute) upon initial SUBSCRIBEs or refresh re-SUBSCRIBEs; otherwise, the Application Server sends "partial" dialog information.

NOTE: SIP dialog identifiers (call-id, from-tag, and to-tag) are reported only for "confirmed" dialogs. The Application Server reports *Proceeding* state as a logical state implying that SIP dialog establishment (potentially with forking) is ongoing; it encompasses the *Trying, Proceeding*, and *Early* states defined in *RFC 4235* [6].

When the Shared Call Appearance service is configured to send the optional Call Park state information, Cisco BroadWorks uses the following extension.

```
<xs:element name="dialog">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="state" type="tns:state"
             minOccurs="1" maxOccurs="1"/>
      <xs:element name="local" type="tns:participant"
             minOccurs="0" maxOccurs="1"/>
      <xs:element name="remote" type="tns:participant">
             minOccurs="0" maxOccurs="1"/>
      <xs:element name="callpark" minOccurs="0" maxOccurs="1">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="parked" type="tns:participant"
                        minOccurs="0" maxOccurs="1"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

- If the *callpark* element is present, then the *local* and *remote* elements are not present.
- If a call is parked against the user, then the *state* element is populated with *confirmed* and the *parked* element is populated with the parked user's details.
- If a call is no longer parked against the user, then the *state* element is populated with *terminated* and the *parked* element is not present.

9.1.2 Subscribe to Dialog Event Package

The Application Server only supports a single dialog subscription according to Shared Call Appearance device AoR. Receiving a SUBSCRIBE for an SCA location, AoR automatically terminates any previous subscriptions for the same SCA location if SIP dialog information changed. The *id Event* header parameter is not supported and thus it is ignored.

The *Request-URI* in the SUBSCRIBE request must match the originator. If these two identities do not match, then the Application Server rejects the SUBSCRIBE request.

Following is an example of a successful subscription.

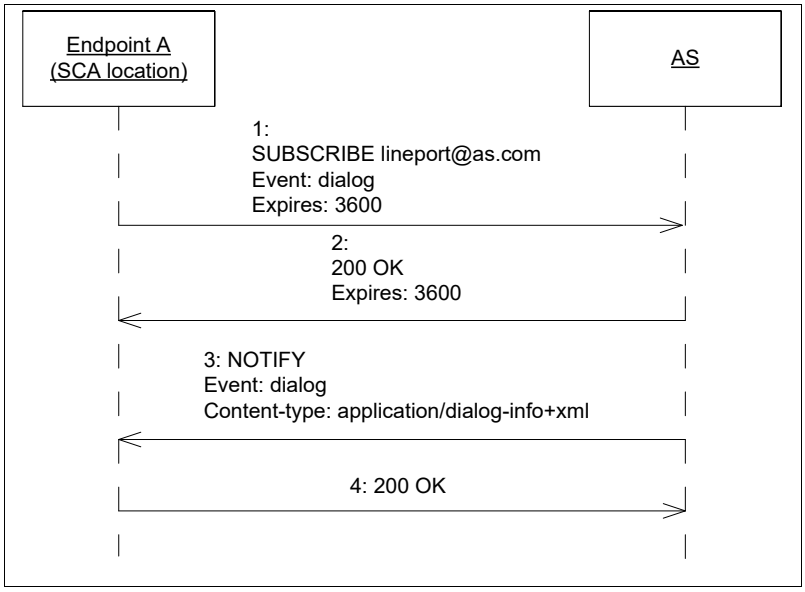


Figure 16 Example of Successful Subscription

1) An SCA location subscribes by sending the SUBSCRIBE SIP message to the Application Server with the following characteristics:
   - The *Request-URI* is the device's address of record (line/port).
   - The *Event* header is set to "dialog" and does not contain any parameter[1].
2) The Application Server validates the SUBSCRIBE and answers with *200 OK*, containing the confirmed subscription's expiration.
3) The Application Server sends an initial NOTIFY to the device containing the "full" current dialog state. The NOTIFY application/dialog-info+xml body is described in section *9.1.3 Receive Dialog Information Change Notifications*.
4) The device answers with a *200 OK*.

9.1.3 Receive Dialog Information Change Notifications

SCA dialog subscribers receive NOTIFY messages when dialog information for the user who owns the SCA device changes (that is, when a new call is initiated, answered, hung up, and so on). Upon such changes, the Application Server sends a NOTIFY message to all SCA device dialog event subscribers for the user. SCA location dialog subscribers receive dialog information for every INVITE-initiated SIP dialogs between the Application Server and any SCA location.

SIP dialog identifiers such as call-id, local-tag, and remote-tag are provided only for "confirmed" dialogs; this is to ensure devices are not trying to "join" or "retrieve" calls in inappropriate states. This also avoids sending many notifications when SIP dialogs are in the "proceeding" as well as for dialog information changes due to forking, redirection, and so on.

[1] More specifically, *id, call-id, to-tag, from-tag*, and *include-session-description* event package parameters are not supported. The *id* parameter is ignored; the presence of other parameters is treated as an invalid subscription request.

When a Share Call Appearance user receives a call, the Application Server creates separate SIP dialogs to each SCA location; each SIP dialog is reported to SCA location dialog subscribers.

Basic Shared Call Appearance call origination

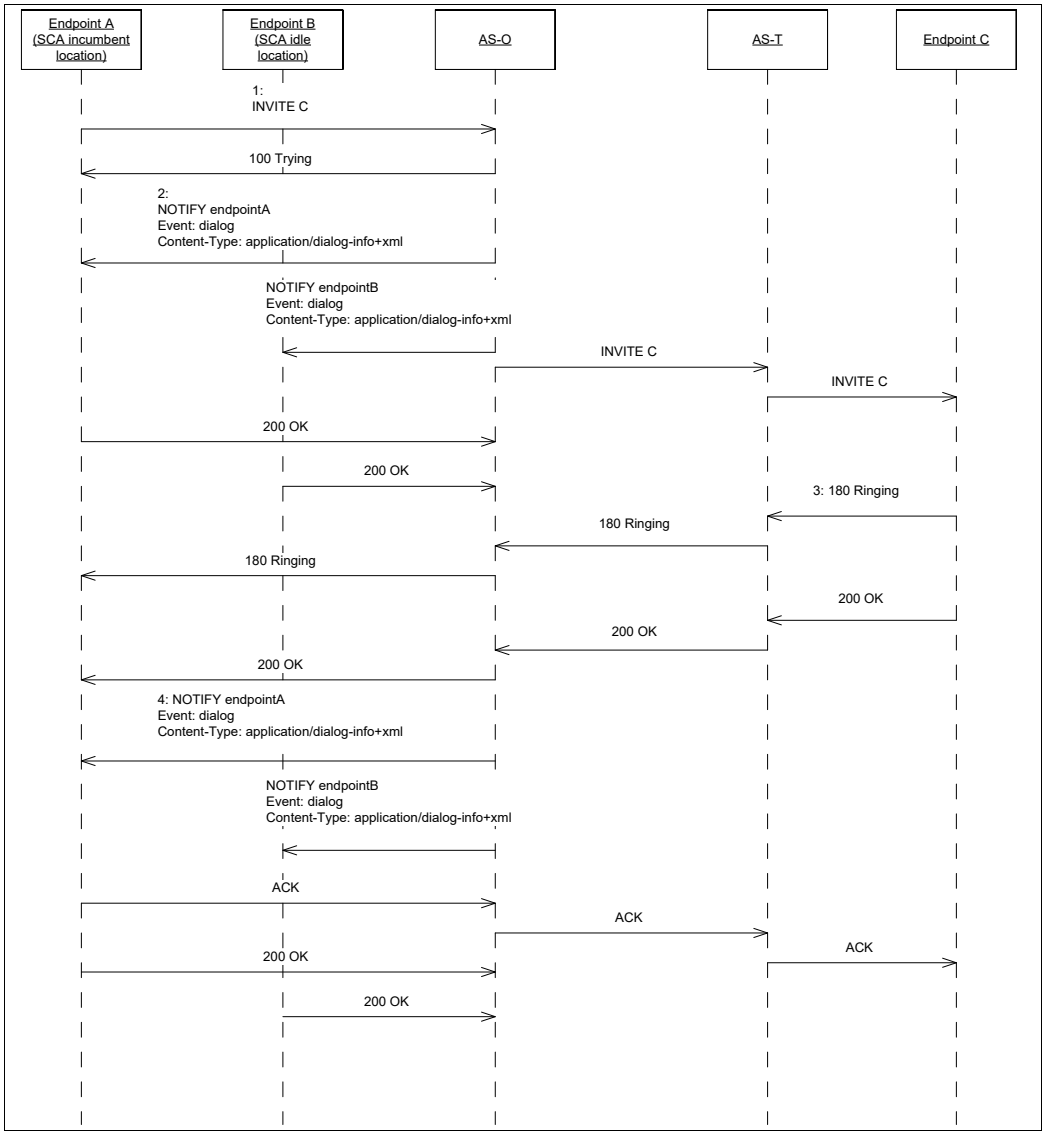


Figure 17 Basic Shared Call Appearance Call Origination

1) SCA location 1 initiates a new call by sending an INVITE to the Application Server; the Application Server responds with *100 Trying*.

2) The Application Server sends a NOTIFY to all SCA location dialog subscribers related to the user. The application/dialog-info+xml body contains a single <dialog> element and the dialog is in the *proceeding* state. The SIP dialog information (call-id, local-tag, and remote-tag) are not provided.

NOTIFY body sample

```
<dialog-info
  xmlns="urn:ietf:params:xml:ns:dialog-info"
  xmlns:sa="urn:ietf:params:xml:ns:sa-dialog-info"
  version="1" state="partial" entity="sip:jsmith@as.com">
  <dialog id="SjRiWml4"
          direction="initiator">
    <state>proceeding</state>
    <sa:appearance>0</sa:appearance>
  </dialog>
</dialog-info>
```

3) The remote party sends a *180 Ringing* and then a *200 OK* response.

4) The Application Server sends a NOTIFY to all SCA location subscribers with the dialog now in the *confirmed* state and full SIP dialog is provided. The INVITE response is "ACKed" by the originator devices; all SCA location dialog subscribers answer the NOTIFY with a *200 OK*.

NOTIFY body sample

```
<dialog-info
  xmlns="urn:ietf:params:xml:ns:dialog-info"
  xmlns:sa="urn:ietf:params:xml:ns:sa-dialog-info"
  version="2" state="partial" entity="sip:jsmith@as.com">
  <dialog id="SjRiWml4"
          direction="initiator"
          call-id="abc@192.168.0.1"
          local-tag="123"
          remote-tag="456">
    <state>confirmed</state>
    <local>
      <identity display="John Smith">sip:jsmith@as.com</identity>
      <target uri="sip:5145551212@192.168.0.1"/>
    </local>
    <remote>
      <identity display="Alice
Smith">sip:5145556666;user=phone</identity>
    </remote>
    <sa:appearance>0</sa:appearance>
  </dialog>
</dialog-info>
```

Remote party hangs up

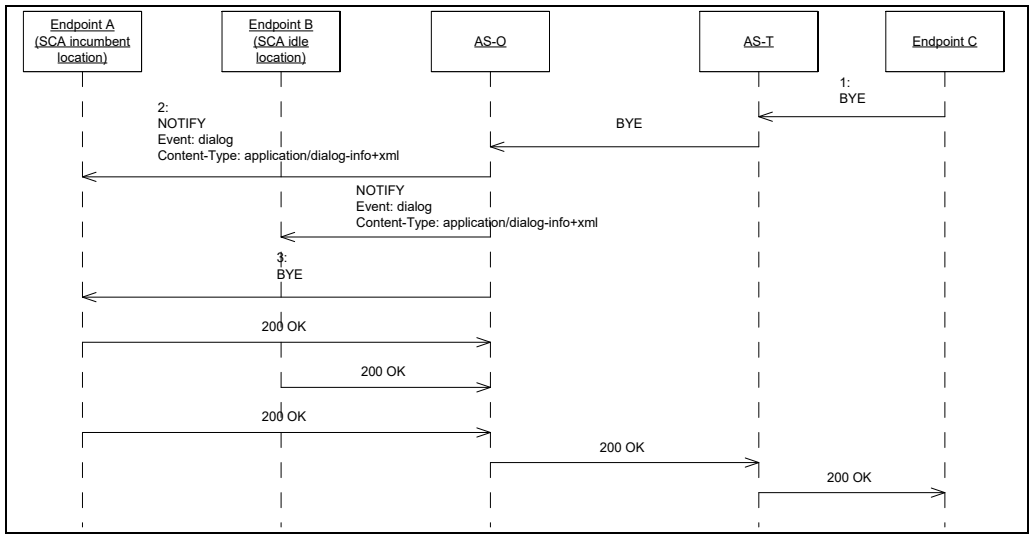


Figure 18 Remote Party Hangs Up

1) The remote party hangs up and sends a BYE.
2) The Application Server sends a NOTIFY to all SCA location subscribers with the dialog now in *terminated* state.
3) The BYE is responded to by the originator. All SCA location dialog subscribers answer to the NOTIFY.

NOTIFY body sample

```
<dialog-info
  xmlns="urn:ietf:params:xml:ns:dialog-info"
  xmlns:sa="urn:ietf:params:xml:ns:sa-dialog-info"
  version="3" state="partial" entity="sip:jsmith@as.com">
  <dialog id="SjRiWml4"
          direction="initiator">
    <state>terminated</state>
  </dialog>
</dialog-info>
```

9.1.4 Notification with Call Park State Information

Figure 19 Shared Call Appearance (SCA) Message Flow shows the message flow for a user with the Shared Call Appearance service configured to send Call Park state information. After the initial subscription, the user receives notifications when the call status or Call Park state changes.

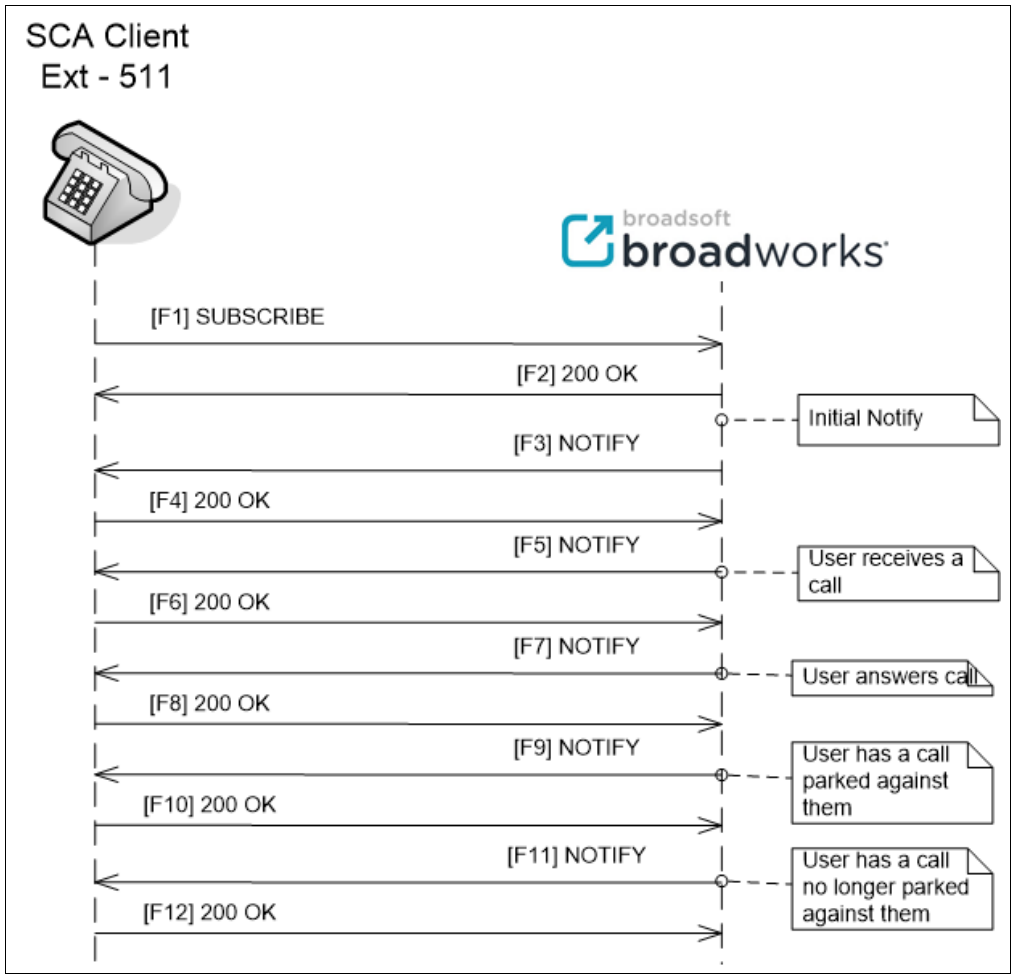


Figure 19 Shared Call Appearance (SCA) Message Flow

Initial Subscription

```
   [F1] Device 9726987511 -> Application Server
SUBSCRIBE sip:9726987511@as.bw.com:5060 SIP/2.0
Via: SIP/2.0/UDP c1.bw.com:5060;branch=z9hG4bK-74124-1-0
From: <sip:9726987511@txasdev87.net:5096>;tag=1
To: <sip:9726987511@as.bw.com:5060>
Call-ID: 1-74124@as.bw.com
CSeq: 1 SUBSCRIBE
Contact: sip:9726987511@c1.bw.com:5060
Expires:1800
Event:dialog
Accept: application/dialog-info+xml
Max-Forwards:10
Content-Length:0
```

```
    [F2] Application Server -> Device 9726987511
SIP/2.0 200 OK
Via:SIP/2.0/UDP c1.bw.com:5060;branch=z9hG4bK-74124-1-0
From:<sip:9726987511@txasdev87.net:5096>;tag=1
To:<sip:9726987511@as.bw.com:5060>;tag=1429831473-1277831629705
Call-ID:1-74124@as.bw.com
CSeq:1 SUBSCRIBE
Expires:1798
Contact:<sip:as.bw.com:5060>
Content-Length:0

    [F3] Application Server -> Device 9726987511
NOTIFY sip:9726987511@c1.bw.com:5060 SIP/2.0
Via:SIP/2.0/UDP as.bw.com;branch=z9hG4bKBroadWorks
From:<sip:9726987511@as.bw.com:5060>;tag=1429831473-1277831629705
To:<sip:9726987511@txasdev87.net:5096>;tag=1
Call-ID:1-74124@as.bw.com
CSeq:78859284 NOTIFY
Contact:<sip:as.bw.com:5060>
Event:dialog
Subscription-State:active;expires=1798
Max-Forwards:10
Content-Type:application/dialog-info+xml
Content-Length:219

<?xml version="1.0" encoding="UTF-8"?>
<dialog-info xmlns="urn:ietf:params:xml:ns:dialog-info"
  xmlns:sa="urn:ietf:params:xml:ns:sa-dialog-info"
  version="0" state="full"
  entity="sip:north00@txasdev87.net">
</dialog-info>

    [F4] Device 9726987511 -> Application Server
SIP/2.0 200 OK
Via:SIP/2.0/UDP as.bw.com;branch=z9hG4bKBroadWorks
From:<sip:9726987511@as.bw.com:5060>;tag=1429831473-1277831629705
To:<sip:9726987511@txasdev87.net:5096>;tag=1
Call-ID:1-74124@as.bw.com
CSeq:78859284 NOTIFY
Contact: <sip:c1.bw.com:5060;transport=UDP>
Content-Length: 0
```

User Bob (500) receives a call from Tom (503). A partial dialog-info message body is received with the new call.

```
    [F5] Application Server -> Device 9726987511
NOTIFY sip:c1.bw.com:5060;transport=UDP SIP/2.0
Via:SIP/2.0/UDP as.bw.com;branch=z9hG4bKBroadWorks
From:<sip:9726987511@as.bw.com:5060>;tag=1429831473-1277831629705
To:<sip:9726987511@txasdev87.net:5096>;tag=1
Call-ID:1-74124@as.bw.com
CSeq:78871703 NOTIFY
Contact:<sip:as.bw.com:5060>
Event:dialog
Subscription-State:active;expires=1786
Max-Forwards:10
Content-Type:application/dialog-info+xml
Content-Length:515

<?xml version="1.0" encoding="UTF-8"?>
<dialog-info xmlns="urn:ietf:params:xml:ns:dialog-info"
  xmlns:sa="urn:ietf:params:xml:ns:sa-dialog-info"
```

```
  version="1" state="partial"
  entity="sip:north00@txasdev87.net">
 <dialog id="Y2FsbGhhbGYtMTU6MQ==" direction="recipient">
  <state>proceeding</state>
  <local>
   <identity display="Bob Robert">sip:north00@txasdev87.net</identity>
  </local>
  <remote>
   <identity display="Tom north">sip:503@as.bw.com;user=phone</identity>
  </remote>
  <sa:appearance>1</sa:appearance>
 </dialog>
</dialog-info>

    [F6] Device 9726987511 -> Application Server
SIP/2.0 200 OK
Via:SIP/2.0/UDP as.bw.com;branch=z9hG4bKBroadWorks
From:<sip:9726987511@as.bw.com:5060>;tag=1429831473-1277831629705
To:<sip:9726987511@txasdev87.net:5096>;tag=1
Call-ID:1-74124@as.bw.com
CSeq:78871703 NOTIFY
Contact: <sip:c1.bw.com:5060;transport=UDP>
Content-Length: 0

    [F7] Application Server -> Device 9726987511
NOTIFY sip:c1.bw.com:5060;transport=UDP SIP/2.0
Via:SIP/2.0/UDP as.bw.com;branch=z9hG4bKBroadWorks
From:<sip:9726987511@as.bw.com:5060>;tag=1429831473-1277831629705
To:<sip:9726987511@txasdev87.net:5096>;tag=1
Call-ID:1-74124@as.bw.com
CSeq:78873416 NOTIFY
Contact:<sip:as.bw.com:5060>
Event:dialog
Subscription-State:active;expires=1784
Max-Forwards:10
Content-Type:application/dialog-info+xml
Content-Length:708

<?xml version="1.0" encoding="UTF-8"?>
<dialog-info xmlns="urn:ietf:params:xml:ns:dialog-info"
  xmlns:sa="urn:ietf:params:xml:ns:sa-dialog-info"
  version="2" state="partial"
  entity="sip:north00@txasdev87.net">
 <dialog id="Y2FsbGhhbGYtMTU6MQ==" direction="recipient"
   call-id="BW121402232290610661637564@as.bw.com"
   local-tag="1d8847e0" remote-tag="1035257616-1277831642232-">
  <state>confirmed</state>
  <local>
   <identity display="Bob Robert">sip:north00@txasdev87.net</identity>
   <target uri="sip:9726987500@c1.bw.com">
   </target>
  </local>
  <remote>
   <identity display="Tom north">sip:503@as.bw.com;user=phone</identity>
  </remote>
  <sa:appearance>1</sa:appearance>
 </dialog>
</dialog-info>
```

```
    [F8] Device 9726987511 -> Application Server
SIP/2.0 200 OK
Via:SIP/2.0/UDP as.bw.com;branch=z9hG4bKBroadWorks
From:<sip:9726987511@as.bw.com:5060>;tag=1429831473-1277831629705
To:<sip:9726987511@txasdev87.net:5096>;tag=1
Call-ID:1-74124@as.bw.com
CSeq:78873416 NOTIFY
Contact: <sip:c1.bw.com:5060;transport=UDP>
Content-Length: 0
```

User Bob (500) has a call parked against them. A partial dialog-info message body is received with only the parked call information.

```
    [F9] Application Server -> Device 9726987511
NOTIFY sip:c1.bw.com:5060;transport=UDP SIP/2.0
Via:SIP/2.0/UDP as.bw.com;branch=z9hG4bKBroadWorks
From:<sip:9726987511@as.bw.com:5060>;tag=1429831473-1277831629705
To:<sip:9726987511@txasdev87.net:5096>;tag=1
Call-ID:1-74124@as.bw.com
CSeq:78891596 NOTIFY
Contact:<sip:as.bw.com:5060>
Event:dialog
Subscription-State:active;expires=1766
Max-Forwards:10
Content-Type:application/dialog-info+xml
Content-Length:468

<?xml version="1.0" encoding="UTF-8"?>
<dialog-info xmlns="urn:ietf:params:xml:ns:dialog-info"
  xmlns:sa="urn:ietf:params:xml:ns:sa-dialog-info"
  xmlns:bw="http://schema.broadsoft.com/callpark"
  version="3" state="partial"
  entity="sip:north00@txasdev87.net">
 <dialog id="Y2FsbGhhbGYtMzM6MA==">
  <state>confirmed</state>
  <bw:callpark>
   <bw:parked>
    <identity display="Alice south">
     sip:9726987601@as.bw.com;user=phone
    </identity>
   </bw:parked>
  </bw:callpark>
 </dialog>
</dialog-info>

    [F10] Device 9726987511 -> Application Server
SIP/2.0 200 OK
Via:SIP/2.0/UDP as.bw.com;branch=z9hG4bKBroadWorks
From:<sip:9726987511@as.bw.com:5060>;tag=1429831473-1277831629705
To:<sip:9726987511@txasdev87.net:5096>;tag=1
Call-ID:1-74124@as.bw.com
CSeq:78891596 NOTIFY
Contact: <sip:c1.bw.com:5060;transport=UDP>
Content-Length: 0
```

The parked user hangs up. A partial dialog-info message body is received with just the parked call information.

```
    [F11] Application Server -> Device 9726987511
NOTIFY sip:c1.bw.com:5060;transport=UDP SIP/2.0
Via:SIP/2.0/UDP as.bw.com;branch=z9hG4bKBroadWorks
From:<sip:9726987511@as.bw.com:5060>;tag=1429831473-1277831629705
To:<sip:9726987511@txasdev87.net:5096>;tag=1
Call-ID:1-74124@as.bw.com
CSeq:78891597 NOTIFY
Contact:<sip:as.bw.com:5060>
Event:dialog
Subscription-State:active;expires=1760
Max-Forwards:10
Content-Type:application/dialog-info+xml
Content-Length:468

<?xml version="1.0" encoding="UTF-8"?>
<dialog-info xmlns="urn:ietf:params:xml:ns:dialog-info"
  xmlns:sa="urn:ietf:params:xml:ns:sa-dialog-info"
  xmlns:bw=http://schema.broadsoft.com/callpark
  version="4" state="partial"
  entity="sip:north00@txasdev87.net">
 <dialog id="Y2FsbGhhbGYtMzM6MA==">
  <state>terminated</state>
  <bw:callpark/>
 </dialog>
</dialog-info>

    [F12] Device 9726987511 -> Application Server
SIP/2.0 200 OK
Via:SIP/2.0/UDP as.bw.com;branch=z9hG4bKBroadWorks
From:<sip:9726987511@as.bw.com:5060>;tag=1429831473-1277831629705
To:<sip:9726987511@txasdev87.net:5096>;tag=1
Call-ID:1-74124@as.bw.com
CSeq:78891597 NOTIFY
Contact: <sip:c1.bw.com:5060;transport=UDP>
Content-Length: 0
```

9.2 Bridging (Barge-in) using Join Header

An SCA location can create a new SCA-Bridge or join an existing SCA-Bridge by creating a new SIP dialog with a SIP INVITE including the *Join* header as defined in *RFC 3911* [4]. Section *3.2.4 Extensions for Shared Call Appearance Bridging* describes an alternative method to perform the same functionality. Either method can be used, but they cannot be used simultaneously.

The following conditions must be met for the bridging request using the *Join* header to be successful:

- The SIP INVITE must come from an SCA location.
- SCA-Bridging capabilities must be enabled for the user.
- The *Join* header must specify a complete dialog (that is, it must contain *callid, to-tag*, and *from-tag*).
- The specified dialog refers to an existing INVITE-initiated SIP dialog between another SCA location for the same user and the Application Server. The dialog may or may not already be a participant in an SCA-Bridge.
- The referred dialog is in the *confirmed* state.

- The call appearance associated with the SIP dialog is not “privately held” (described in section *3.2.1 In INVITE Requests*).

When the conditions are met, the Application Server creates the SCA-Bridge or connects the SCA location to an existing SCA-Bridge if the joined dialog is already part of an SCA-Bridge. In terms of Shared Call Appearance terminology, the SCA location initiating SCA-Bridging is assigned the same call appearance as the call appearance associated with the joined SIP dialog.

If the device supports silent monitoring, it may add a *silent-monitor* parameter to the *Join* header to request it from Cisco BroadWorks. Silent monitoring is a special form of SCA bridging in which the barge-in party can listen to the other parties but cannot be heard by them.

SCA locations are aware of ongoing “confirmed” SIP dialogs through SCA location dialog event package notifications (see section *9.1 Dialog Event Package*).

Using the *Join* header is an explicit request to create/join a bridge; the Application Server does not interpret a SIP INVITE with the *Join* header to perform Shared Call Appearance retrieval, as is currently the case for devices supporting the *Call-Info* header.

When sending an INVITE to the Application Server with the *Join* header, the *Join* header takes precedence over the Request-URI (in which case the *Request-URI* becomes largely irrelevant). However, the *Join* header is ignored when the *Request-URI* is a conference URI configured on the Application Server.

The following is an example call flow. In the example, a call is already active between endpoints A and C. AS-O and AS-T represent the Application Server for the originating and terminating user respectively. Note that AS-T can be the same as AS-O, or a gateway to the PSTN without changing the call flow.

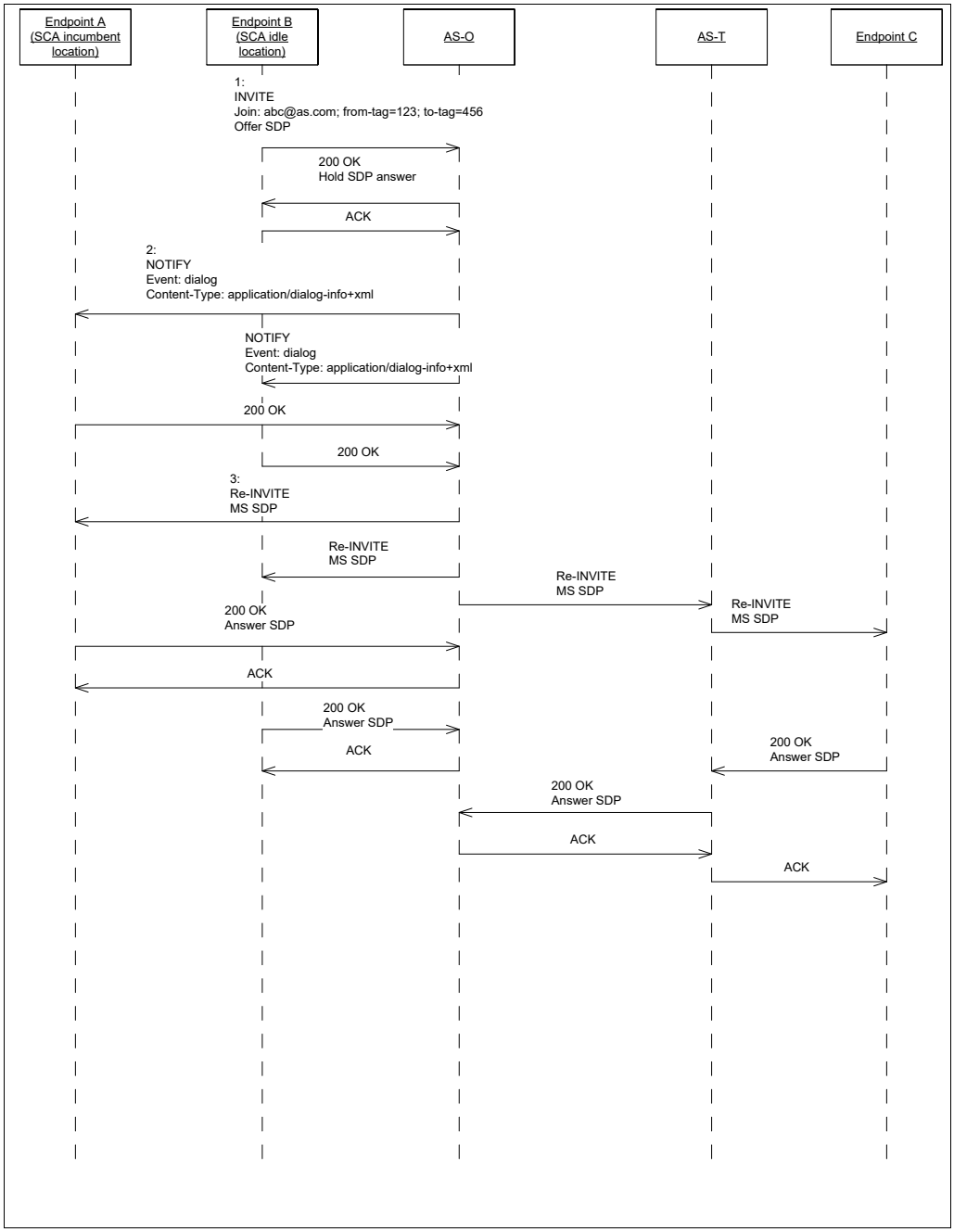


Figure 20 Example Call Flow – Bridging (Barge-in) using Join Header

1) The bridging SCA location sends an INVITE containing the *Join* header. The Application Server matches the *Join* header with the SIP dialog between the incumbent location and the Application Server. The Application Server returns a hold SDP to the bridging location and starts bridge creation in the meantime.

2) The Application Server sends NOTIFY to SCA location dialog subscribers reporting the new "confirmed" dialog between the Application Server and the bridging location.

3) Once the bridge is ready, the Application Server reconnects SCA location and the remote party to the SCA-Bridge.

Example SIP INVITE with the *Join* header

```
INVITE sip:5145551212@as.com SIP/2.0
Via:SIP/2.0/UDP 192.168.0.1;branch=z9hG4bKBroadWorks.-1su2ja6-192.168.22.75V5060-0-
559199717-1520248096-1233774013385-
From:"john smith"<sip:5145551212@192.168.0.1;user=phone>;tag=1520248096-
1233774013385-
To: sip:5145551212@as.com
Call-ID:BW140013385040209-1877405103@192.168.8.131
CSeq:559199717 INVITE
Contact:<sip:192.168.0.1>
Supported:100rel
Allow:ACK,BYE,CANCEL,INFO,INVITE,OPTIONS,PRACK,REFER,NOTIFY
Join: 12adf2f34456gs5;to-tag=12345;from-tag=54321
Max-Forwards:10
Content-Type:application/sdp
Content-Length:xxx
…
```

Example SIP INVITE with the *Join* header and *silent-monitor* parameter for silent monitoring

```
INVITE sip:5145551212@as.com SIP/2.0
Via:SIP/2.0/UDP 192.168.0.1;branch=z9hG4bKBroadWorks.-1su2ja6-192.168.22.75V5060-0-
559199717-1520248096-1233774013385-
From:"john smith"<sip:5145551212@192.168.0.1;user=phone>;tag=1520248096-
1233774013385-
To: sip:5145551212@as.com
Call-ID:BW140013385040209-1877405103@192.168.8.131
CSeq:559199717 INVITE
Contact:<sip:192.168.0.1>
Supported:100rel
Allow:ACK,BYE,CANCEL,INFO,INVITE,OPTIONS,PRACK,REFER,NOTIFY
Join: 12adf2f34456gs5;to-tag=12345;from-tag=54321;silent-monitor
Max-Forwards:10
Content-Type:application/sdp
Content-Length:xxx
…
```

9.3 Retrieve Call using Replaces Header

This section refers to the Shared Call Appearance feature in which a location can retrieve a call from another SCA location. When call retrieval occurs, the initial (target) location is released. While the main intended use for this function is to allow holding a call from an SCA location and retrieve it from another SCA location, it is also permitted to retrieve an active call (in all cases the call leg between the Application Server and the former SCA location is released).

The Application Server supports the *Replaces* header as a way to perform Shared Call Appearance call retrieval from an SCA location. The alternative uses the *Call-Info* header and is described in section *3 Call-Info Header Extensions*. Either method can be used, but they cannot be used simultaneously.

The following conditions must hold so the call retrieval request using the *Replaces* header is successful:

- The INVITE containing the *Replaces* header must come from an SCA location.
- The *Replaces* header must identify a complete SIP dialog (that is, it must contain *callid, to-tag*, and *from-tag*).
- The identified dialog must refer to an existing INVITE-initiated SIP dialog between another SCA location for the same user and the Application Server. The dialog must *not* be part of an SCA-Bridge.

- The identified dialog must be in the *confirmed* state.

When the conditions are met, the Application Server reconnects the remote party to the new SCA location and releases the target location.

SCA locations learn about current confirmed dialogs by subscribing to the dialog event package on the Application Server (see section *9.1 Dialog Event Package*) or via some proprietary ways not defined by Cisco BroadWorks.

> **NOTE**: Dialog information provided to SCA location dialog subscribers includes the appearance index. If more than one SIP dialog has the same appearance index, the dialogs are part of an SCA-Bridge. The SCA location shall not initiate call retrieval.

Figure 21 illustrates the call flow for retrieving a call. In this figure, A-1 and B are already connected together, and A-2 retrieves the call.

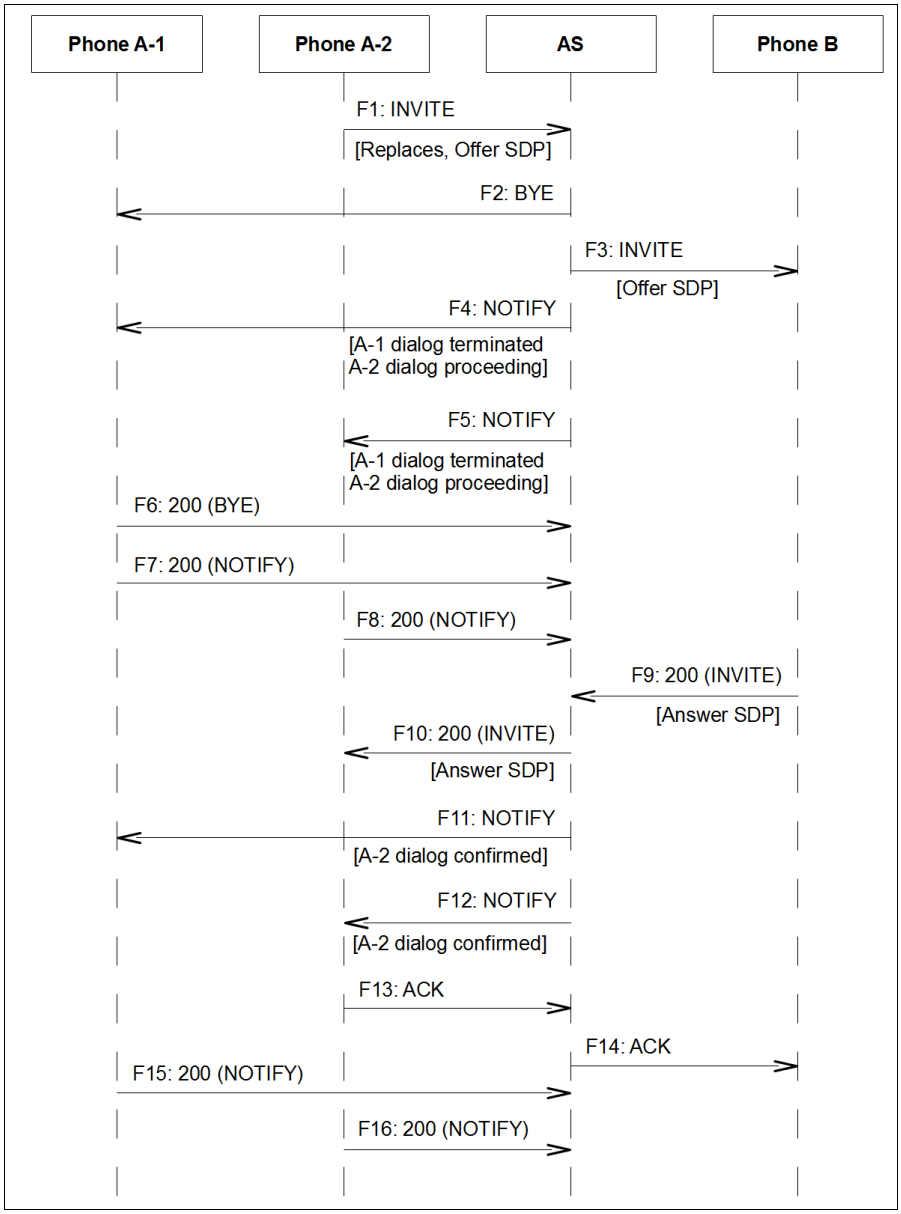


Figure 21 Call Flow – Retrieve Call using Replaces Header

1) The retrieving SCA endpoint (Phone A-2) sends an INVITE request (F1) to the Application Server. The INVITE request contains a *Replaces* header that matches the existing dialog between the Application Server and the incumbent SCA endpoint (Phone A-1).

2) The Application Server sends– a BYE request (F2) to Phone A-1 to release the existing call. At the same time, it terminates the associated SIP dialog.

3) The Application Server sends a re-INVITE request (F3) to Phone B to establish the media session with Phone A-2.

4) The Application Server sends NOTIFY requests (F4, F5) to all dialog subscribers, notifying them that the previously existing dialog is now terminated. The NOTIFY requests also provide information about the new unconfirmed dialog between the Application Server and Phone A-2. As seen in the call flow diagram, both Phone A-1 and Phone A-2 are subscribers, so both receive the NOTIFY request.

5) The Application Server receives a *200* response (F9) from the Phone B and sends a *200* response (F10) to Phone A-2. The new dialog is now confirmed, and the Application Server sends new NOTIFY requests (F11, F12) to all subscribers to notify them that the new dialog is confirmed.

Example SIP INVITE with *Replaces* header

```
INVITE sip:5145551212@as.com SIP/2.0
Via:SIP/2.0/UDP 192.168.0.1;branch=z9hG4bKBroadWorks.-1su2ja6-192.168.22.75V5060-0-
559199717-1520248096-1233774013385-
From:"john smith"<sip:5145551212@192.168.0.1;user=phone>;tag=1520248096-
1233774013385-
To: sip:5145551212@as.com
Call-ID:BW140013385040209-1877405103@192.168.8.131
CSeq:559199717 INVITE
Contact:<sip:192.168.0.1>
Supported:100rel
Allow:ACK,BYE,CANCEL,INFO,INVITE,OPTIONS,PRACK,REFER,NOTIFY
Replaces: 12adf2f34456gs5;to-tag=12345;from-tag=54321
Max-Forwards:10
Content-Type:application/sdp
Content-Length:xxx
…
```

10 Hoteling Event Package

10.1 Overview

The Hoteling Application Server function allows guest users to share one or more desk positions (also called host devices) that are designed for a common purpose. In the context of a call center, agents act as guests and log in to one of the host devices to perform their duties.

The guest may log in (that is, associate with the host device) from a client application (for example, the Cisco BroadWorks Call Center application), from the web portal, or by dialing into the voice portal from the host device. This extension provides a mechanism for the host device to synchronize with the Application Server when the host device's association status is modified. It also allows the SIP phone user to use keys on the phone as an alternative to logging in on the host device.

The synchronization protocol is based on the SIP-events framework and uses the *x-broadworks-hoteling* event package (referred as the hoteling event package in the text).

The hoteling event package is an instantiation of the SIP Specific Event Notification Framework (as defined in *RFC 3265*). For the applications described earlier, the Application Server acts as the "notifier", while the IP phones act as the "subscribers".

Following is a description of the event package details, as per *RFC 3265*.

10.2 Example Message Flow

10.2.1 Device Initial Subscription (Subscription with No Body)

When a supported phone powers up, it sends an initial subscription (SUBSCRIBE request) to the Application Server to subscribe to the *x-broadworks-hoteling* event package. The initial subscription to the *x-broadworks-hoteling* event package request is sent with an empty body. This initial subscription allows the receipt of the current Hoteling Host service status as well as the subscription to receive any change to the Hoteling Host service status.

A subscription is accepted only if Hoteling Host service is assigned and active (enabled).

Figure 22 shows the call flow of a scenario for the device initial subscription when the Hoteling Host service is configured properly.

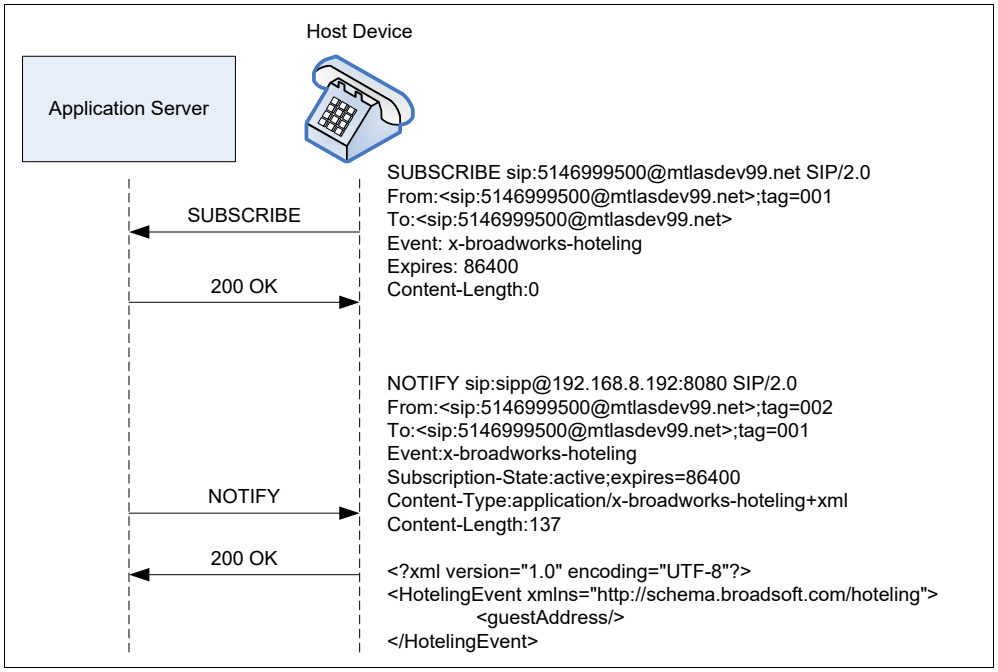


Figure 22 Device Initial Subscription

10.2.2 Host-Guest Association Created Notification

When the user uses the web portal, the voice portal, or a client application to create a host-guest association, the Application Server sends a NOTIFY request for the *x-broadworks-hoteling* event package, as shown in *Figure 23*.

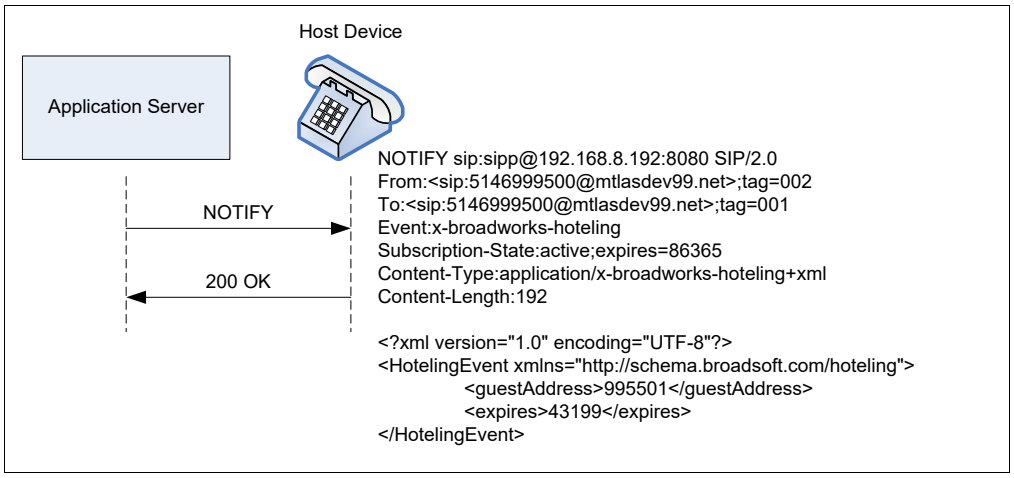


Figure 23 Host-Guest Association Created Notification

The *guestAddress* attribute contains the guest user primary phone number or the guest user ID. If there is no current host-guest association, the attribute is empty. The *guestAddress* attribute is formatted using the following rules (similar to the hoteling voice portal host-guest association):

- If location code and user extension are provisioned, *guestAddress* is set to location code + extension.
- If location code is not provisioned, *guestAddress* attribute is set to the user extension.

- If location code and guest extension are not provisioned, *guestAddress* attribute is set to the user phone number (national number).
- If user phone number is not provisioned, the *guestAddress* attribute is set to the userId.

In the example above, the *guestAddress* is set to the location code (995) + guest extension (501). In addition, the *expires* attribute contains the number of seconds before the host-guest association automatically expires on the Application Server. Note that this expiration value is different from the SIP subscription expiration and relates to the expiration of the Hoteling Host service itself. Note, that the *expires* attribute can be empty if the host-guest association has no expiration date.

10.2.3 Host-Guest Association Terminated Notification

When the user uses the web portal, the voice portal, or a client application to terminate a host-guest association, or it expires, the Application Server sends a NOTIFY request for the *x-broadworks-hoteling* event package, as shown in *Figure 24*.

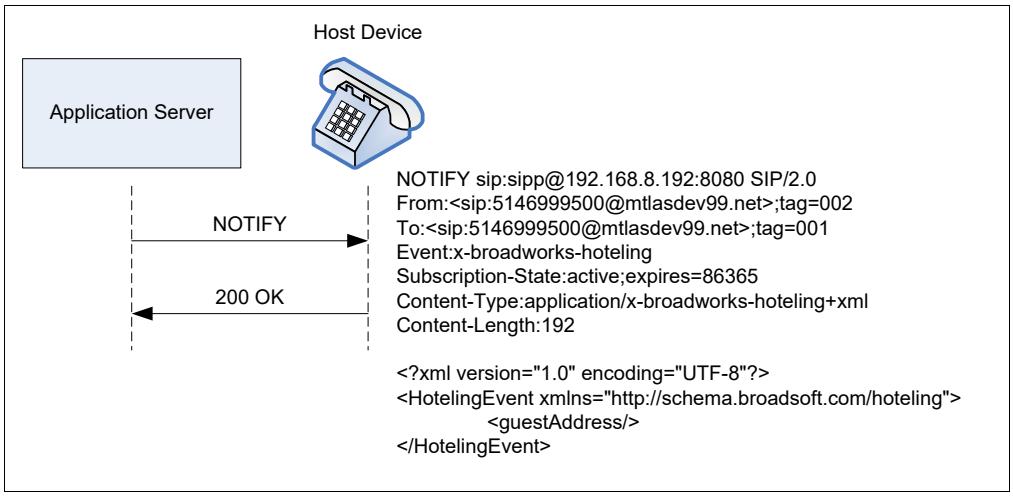


Figure 24 Host-Guest Association Terminated Notification

Note that *guestAddress* is empty to indicate that no guest user is currently associated with the host device.

10.2.4 Create Host-Guest Association

The device can use a SUBSCRIBE request with an application/x-broadworks-hoteling+xml message body to configure a new Host-Guest association. To create a host-guest association using a host device requires authentication at the application layer. For this reason, creating a host-guest association using the host device is done in two steps, association request and authentication.

Figure 25 shows the first step. A host-guest association request (SUBSCRIBE request with body without authentication data) is initiated by the host device. When the user uses the host device to create the host-guest association, the device sends a first SUBSCRIBE request with a body (without authentication attributes) to the Application Server to request the hoteling status change.

Immediately afterward, the Application Server requests for authentication (NOTIFY request including authentication attributes). The association status is not modified.

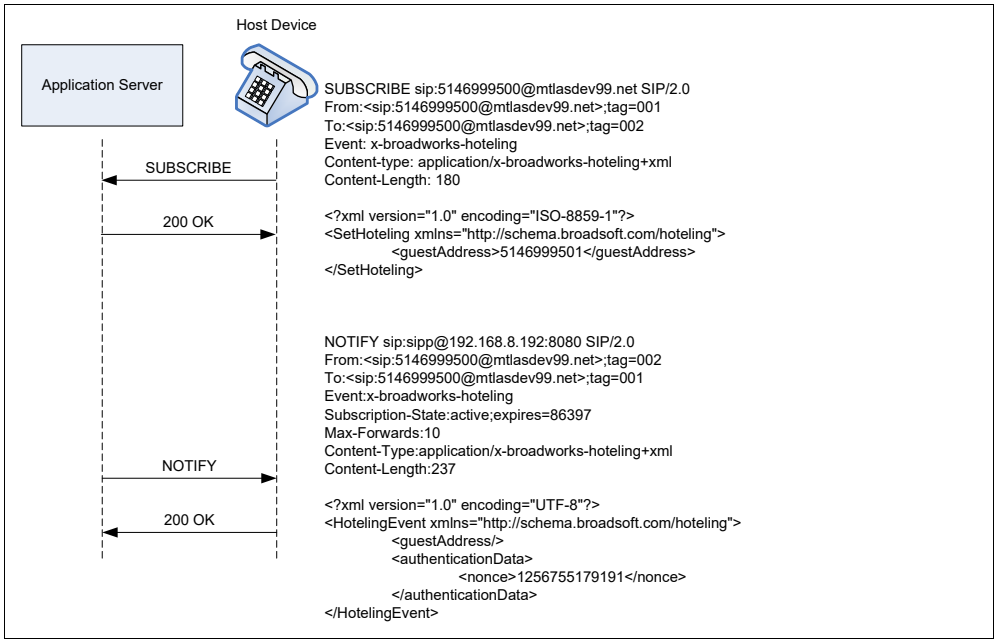


Figure 25 Host-Guest Association Request Step 1

In most cases, the authentication data to be used in the subsequent authentication request is provided. However, in the following cases, the Application Server may simply indicate an association request failure by sending the current hoteling host status (NOTIFY request without authentication data):

- The guest user provided in the *guestAddress* is unknown.
- The Hoteling Guest service is not assigned or disabled.
- The guest user provided in the *guestAddress* violates the hoteling host access level configuration (enterprise or group level).
- The host device already has a guest user assigned.

Figure 26 shows the second step. A host-guest authentication request (SUBSCRIBE request with body including authentication attributes) is initiated by the host device. Upon receiving the authentication data, the phone sends an authentication request to the Application Server that contains the host-guest association hoteling request data, but also the MD5 message digest of the combination of the nonce with the user's passcode (nonce + ":" + passcode).

Immediately afterward, the Application Server provides the hoteling host status update, including the current status.

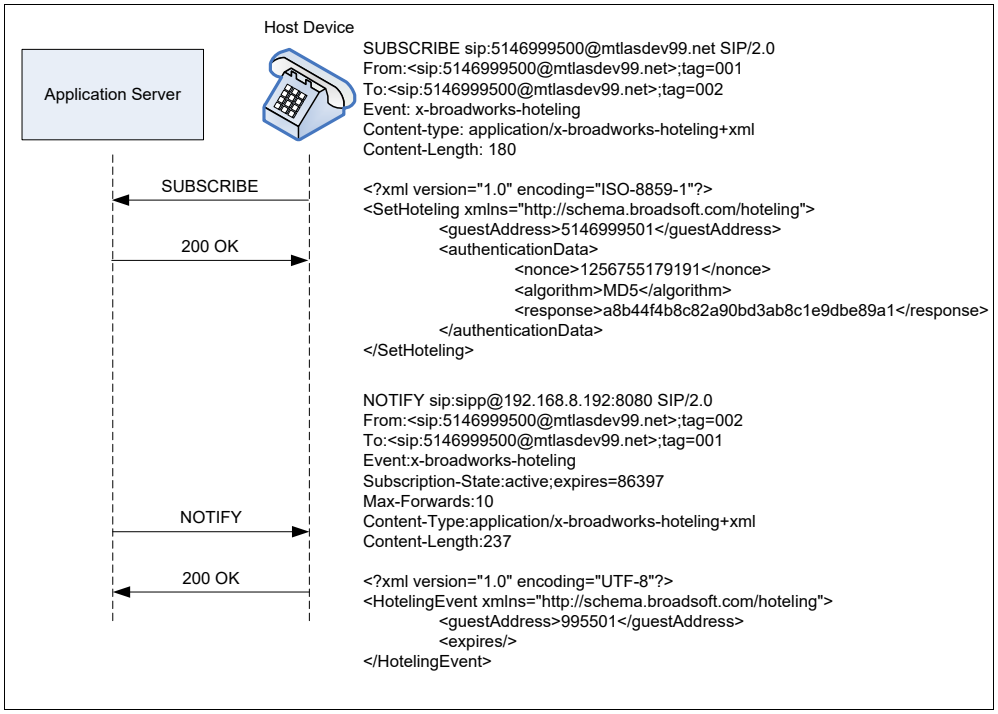


Figure 26 Host-Guest Authentication Request Step 2

10.2.5 Terminate Host-Guest Association

Terminating a host-guest association using the host device does not require authentication at the application layer. When the user uses the host device to terminate the current host-guest association, the device sends a request with a body including an empty *guestAddress*. Immediately after, the Application Server provides the hoteling host status update including the current status change, as shown in *Figure 27*.

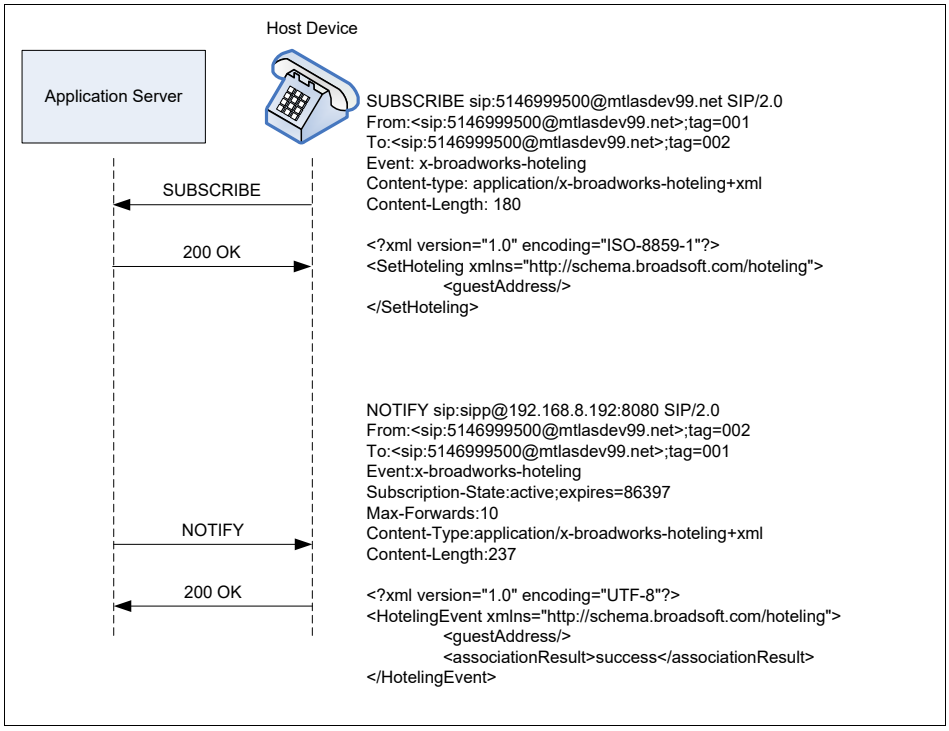


Figure 27 Terminate Host-Guest Association

10.3 Event Package Name

The name of this event is "*x-broadworks-hoteling*". It is carried in the *Event and Allow-Events* header, as defined in *RFC 3265*.

10.4 Event Package Parameters

This package does not define any event package parameters.

10.5 SUBSCRIBE Bodies

The SUBSCRIBE message is sent by the phone to the Application Server. The message body is optional.

The phone sends a SUBSCRIBE message without a message body to obtain the current status and request future feature status change notifications for the duration of the subscription.

The phone sends a SUBSCRIBE message with a message body to modify the host-guest association. The message body is of type *application/x-broadworks-hoteling* and contains an XML message. The following is the hoteling body format for the *application/x-broadworks-hoteling+xml* included in the SIP subscribe requests.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="http://schema.broadsoft.com/hoteling"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:hoteling="http://schema.broadsoft.com/hoteling"
            elementFormDefault="qualified"
            attributeFormDefault="unqualified">
  <xsd:annotation>
    <xsd:documentation>Hoteling-Set-Guest</xsd:documentation>
  </xsd:annotation>
  <xs:element name="SetHoteling">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="guestAddress" type="xs:string">
            <xs:annotation>
              <xs:documentation>
                The phone number or userID used to identify the guest
                user.  Empty to terminate a host-guest association.
              </xs:documentation>
            </xs:annotation>
        </xs:element>
        <xs:element name="authenticationData" type="AuthenticationData"
                    minOccurs="0"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xsd:schema>
```

Authentication data is included within the SUBSCRIBE request message body and provides the nonce and algorithm used to generate the secured password.

```
<xs:complexType name="AuthenticationData">
   <xs:sequence>
     <xs:element name="nonce" type="xs:string">
       <xs:annotation>
          <xs:documentation>
            The private key used to generate the password digest.
          </xs:documentation>
        </xs:annotation>
     </xs:element>
     <xs:element name="algorithm">
       <xs:annotation>
          <xs:documentation>
            The algorithm used to generate the password digest.
          </xs:documentation>
       </xs:annotation>
       <xs:simpleType>
         <xs:restriction base="xs:string">
           <xs:enumeration value="MD5"/>
         </xs:restriction>
       </xs:simpleType>
     </xs:element>
     <xs:element name="response" type="xs:string" minOccurs="0">
       <xs:annotation>
          <xs:documentation>
            The secured password to use for authentication.
          </xs:documentation>
       </xs:annotation>
     </xs:element>
   </xs:sequence>
</xs:complexType>
```

10.6 Subscription Duration

Subscribers should specify the duration of a subscription with an *Expires* header in the SUBSCRIBE request. It is recommended that the subscription use the same refresh period as the REGISTER event. If the notifier changes the expiration period to a lower value (via an *Expires* header in the final response), the subscriber should honor it.

If the subscriber does not specify the duration of a subscription, then the notifier chooses the default expiration.

10.7 NOTIFY Bodies

The NOTIFY message is sent by the Application Server to the phone. The NOTIFY message is triggered by a SUBSCRIBE message sent by the phone, or a Host-Guest association change.

The following is the hoteling body format for the *application/x-broadworks-hoteling+xml* included in the SIP notify request.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="http://schema.broadsoft.com/hoteling"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:hoteling="http://schema.broadsoft.com/hoteling"
            elementFormDefault="qualified"
            attributeFormDefault="unqualified">
  <xsd:annotation>
    <xsd:documentation>Hoteling-Status</xsd:documentation>
  </xsd:annotation>
  <xs:element name="HotelingEvent">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="guestAddress" type="xs:string">
            <xs:annotation>
              <xs:documentation>
                The phone number or userid of the current guest user
                associated to the host device.  Empty if there is no
                current host-guest association.
              </xs:documentation>
            </xs:annotation>
        </xs:element>
        <xs:element name="expires" type="xs:int" minOccurs="0">
            <xs:annotation>
              <xs:documentation>
                The number of seconds before the host-guest association
                automatically expires on the Application Server
              </xs:documentation>
            </xs:annotation>
        </xs:element>
        <xs:element name="authenticationData" type="AuthenticationData"
                    minOccurs="0"/>
        <xs:any namespace="##any" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xsd:schema>
```

Authentication data is included within the NOTIFY request message body and provides a nonce to be used for the message digest authentication. Its schema definition is included in section *10.5 SUBSCRIBE Bodies*.

10.8 Subscriber Generation of SUBSCRIBE Requests

Subscriber (SIP phone) generation of SUBSCRIBE requests must follow all rules with respect to subscriber behavior, as described in *RFC 3265*.

SIP phones should generate SUBSCRIBE requests immediately after successfully registering. SUBSCRIBE requests should also be generated when the subscription approaches the expiration time. These SUBSCRIBE messages typically do not contain a message body.

SIP phones should also generate SUBSCRIBE requests with an *application/x-broadworks-hoteling+xml* message body to configure a host-guest association. The SIP phone must not consider a *200* response from the Application Server as an acknowledgement that the association state has changed. The *200* response from the Application Server is merely an indication that the subscription was accepted. The phone must wait for the subsequent NOTIFY message to synchronize with the actual host-guest association state on the Application Server, which may or not be the state requested in the SUBSCRIBE request.

10.9 Notifier Processing of SUBSCRIBE Requests

Notifier (Application Server) processing of SUBSCRIBE requests must follow all rules with respect to notifier behavior, as described in *RFC 3265*. The notifier may choose to explicitly challenge the subscriber for authentication by using a *401 Unauthorized* response. If the subscriber has also registered a location for this address of record, then it should use the same credentials to answer the subscription challenge.

10.10 Notifier Generation of NOTIFY Requests

Notifier (Application Server) generation of NOTIFY requests must follow all rules with respect to notifier behavior, as described in *RFC 3265* [3].

10.11 Subscriber Processing of NOTIFY Requests

In general, subscriber processing of NOTIFY requests must follow all rules with respect to subscriber behavior, as described in *RFC 3265*. The SIP-specific event notification framework expects packages to specify how a subscriber processes NOTIFY requests, and in particular, how it uses the NOTIFY requests to construct a coherent view of the state of the subscribed resource.

The NOTIFY messages contain the current host-guest association status. The subscriber must discard any previous state information and replace it with the received information.

The NOTIFY messages received in response to SUBSCRIBE requests with a message body may contain authentication information as described in section *10.7 NOTIFY Bodies*. The subscriber should re-subscribe with the appropriate authentication information in the message body, as described in section *10.5 SUBSCRIBE Bodies*.

10.12 Rate of Notifications

The rate of notifications is entirely dependent on the frequency at which the host-guest association changes. Typically, this occurs at most a few times per hour.

11 Call Center Status Event Package

11.1 Overview

Incoming calls to a call center are queued and offered to the agent's device when the agent becomes available. Depending on various factors, the number of calls in queue and/or the average wait time of calls in queue vary over time, and directly affect the perceived quality of service received by callers.

This event package introduces a mechanism for the agent device to synchronize with the status of Premium call centers when the agent is assigned to and has joined the call center. This synchronization mechanism allows the agent to get a visual indicator of the call center status and modify the call handling behavior accordingly.

The status of a call center is determined by the Application Server via the implementation of thresholds related to the number of calls in queue and to the longest wait time of calls in queue. The Application Server reports the following call center status:

Status	Description
Empty	No calls are currently queued.
Normal	There is at least one call queued and it has not met any conditions for Threshold Exceeded.
Threshold Exceeded	The number of queued calls equals or exceeds the number of calls configured or a call in the queue exceeds the configured wait time.

The synchronization protocol is based on the SIP-events framework and uses the *x-broadworks-call-center-status* event package (referred as the call center status event package in the text).

The call center status event package is an instantiation of the SIP-specific event notification framework (as defined in *RFC 3265*). For the applications described earlier, the Application Server acts as the "notifier" while the IP phones act as the "subscribers".

Following is a description of the event package details, according to *RFC 3265*.

11.2 Example Message Flow

11.2.1 Subscription and Initial Notify

Figure 28 shows call flow for a scenario when Call Center – Premium service is configured properly for a user. In this scenario, the agent is assigned and has joined two different call centers (CallCenter and CallCenter2) and the queue status monitoring is enabled for both call centers.

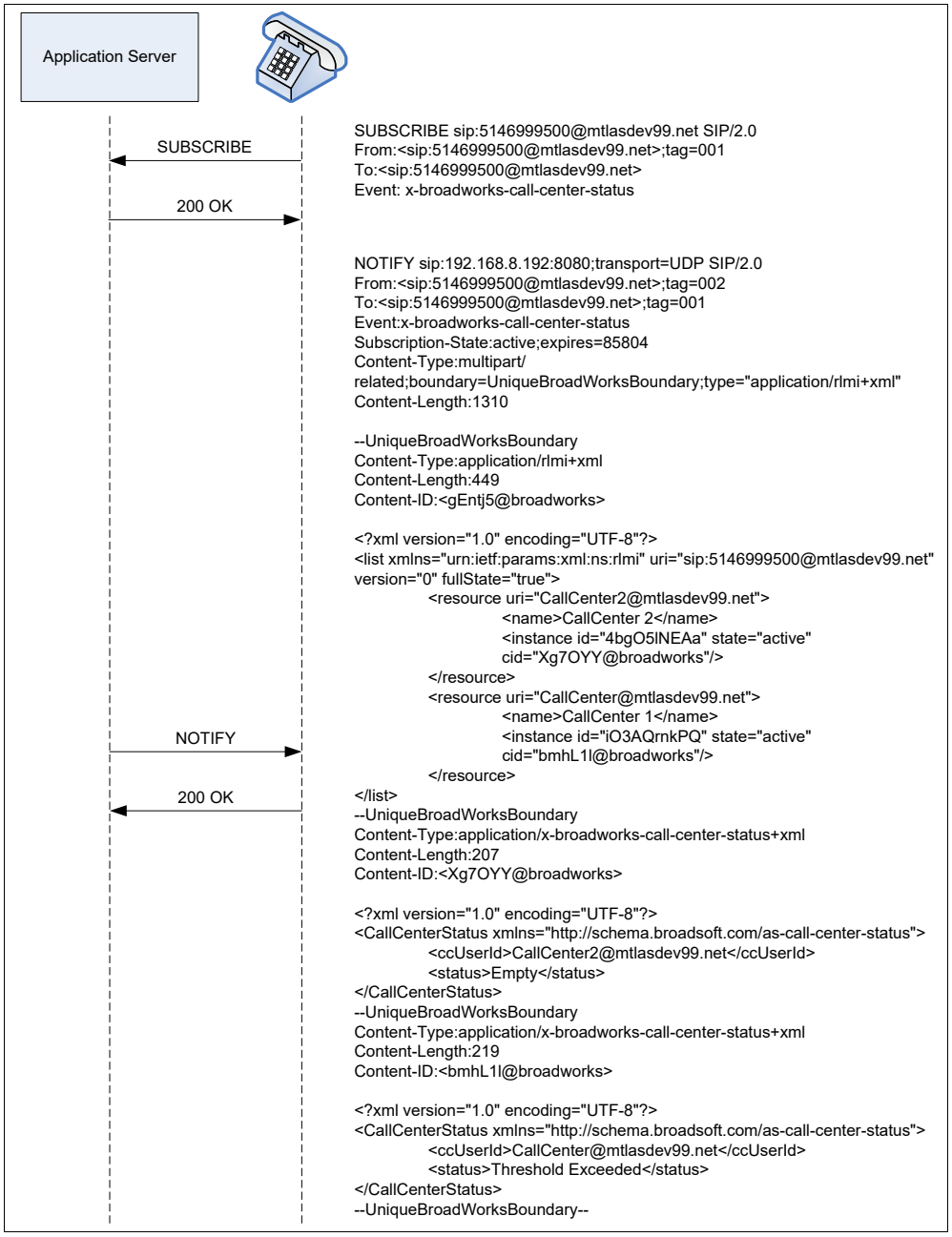


Figure 28 Subscription to Call Center Status Event Package

11.2.2 Partial Status Update Notify

A partial status update is sent to all subscribed agents that are assigned (and joined) the call center each time the call center status changes.

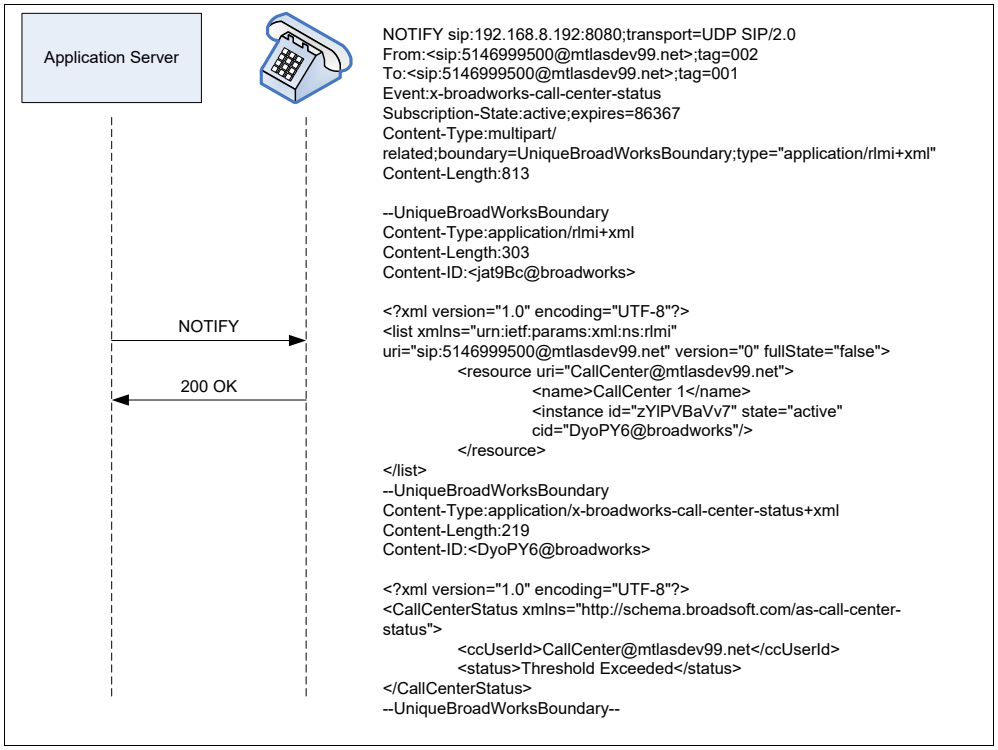


Figure 29 Partial Call Center Status Update

11.3 Event Package Name

The name of this event is "*x-broadworks-call-center-status*". It is carried in the *Event and Allow-Events* header, as defined in *RFC 3265*.

11.4 Event Package Parameters

This package does not define any event package parameters.

11.5 SUBSCRIBE Bodies

The SUBSCRIBE message is sent by the phone to the Application Server to obtain the current status and request future call center status change notifications for the duration of the subscription. The message does not contain a message body.

11.6 Subscription Duration

Subscribers **should** specify the duration of a subscription with an *Expires* header in the SUBSCRIBE request. It is recommended that the subscription use the same refresh period as the REGISTER event. If the notifier changes the expiration period to a lower value (via an *Expires* header in the final response), the subscriber should honor it.

If the subscriber does not specify the duration of a subscription, then the notifier chooses the default expiration.

11.7 NOTIFY Bodies

The NOTIFY message is sent by the Application Server to the phone. The NOTIFY message is triggered by a SUBSCRIBE message sent by the phone or a call center status change.

The NOTIFY request always includes one or more message bodies joined in a multipart/related construct. The first message body is of type *application/rlmi+xml* (defined in *RFC 4662* [7]) and provides the list of Premium call centers that the agent is currently assigned and has joined. The NOTIFY request may also include additional message bodies of type *application/x-broadsoft-call-center-status+xml*, which contain an XML message with the corresponding call center status, as specified in the following example.

The following is the call center status body format (*application/x-broadsoft-call-center-status+xml*).

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="http://schema.broadsoft.com/as-call-center"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:as-call-
center="http://schema.broadsoft.com/as-call-center"
elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xsd:annotation>
    <xsd:documentation>
      BroadWorks Call Center Status
    </xsd:documentation>
  </xsd:annotation>
  <xsd:element name="CallCenterStatus">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="id" type="xsd:string">
          <xsd:annotation>
            <xsd:documentation>
              The identifier of the call center.
            </xsd:documentation>
          </xsd:annotation>
        </xsd:element>
        <xsd:element name="status" type="CallCenterStatusValue">
          <xsd:annotation>
            <xsd:documentation>
              The status of the call center.
            </xsd:documentation>
          </xsd:annotation>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:simpleType name="CallCenterStatusValue">
    <xsd:annotation>
      <xsd:documentation>
        The possible values for the status of a Call Center.
      </xsd:documentation>
    </xsd:annotation>
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="Empty"/>
      <xsd:enumeration value="Normal"/>
      <xsd:enumeration value="Threshold Exceeded"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:schema>
```

As shown in the examples in section *11.2 Example Message Flow*, NOTIFY bodies can contain full or partial updates.

Full updates are sent in response to SUBSCRIBE messages. They contain the list of all Premium call centers the agent is assigned to and has joined as well as their status.

Partial updates are sent when a status change occurs for one of the Premium call centers the agent is assigned to and has joined. It only contains the information for the corresponding call center.

11.8 Subscriber Generation of SUBSCRIBE Requests

Subscriber (SIP phone) generation of SUBSCRIBE requests must follow all rules with respect to subscriber behavior, as described in *RFC 3265*.

SIP phones should generate SUBSCRIBE requests immediately after successfully registering. SUBSCRIBE requests should also be generated when the subscription approaches the expiration time. These SUBSCRIBE messages do not contain a message body.

11.9 Notifier Processing of SUBSCRIBE Requests

Notifier (Application Server) processing of SUBSCRIBE requests must follow all rules with respect to notifier behavior, as described in *RFC 3265*. The notifier may choose to explicitly challenge the subscriber for authentication by using a *401 Unauthorized* response. If the subscriber has also registered a location for this address of record, then it should use the same credentials to answer the subscription challenge.

11.10 Notifier Generation of NOTIFY Requests

Notifier (Application Server) generation of NOTIFY requests must follow all rules with respect to notifier behavior, as described in *RFC 3265*.

11.11 Subscriber Processing of NOTIFY Requests

In general, subscriber processing of NOTIFY requests must follow all rules with respect to subscriber behavior, as described in *RFC 3265*. The SIP-specific event notification framework expects packages to specify how a subscriber processes NOTIFY requests, and in particular, how it uses the NOTIFY requests to construct a coherent view of the state of the subscribed resource.

The NOTIFY messages contain the current call center status. The subscriber must discard any previous state information for the specified call center(s) and replace it with the received information.

11.12 Rate of Notifications

The rate of notifications is dependent on the frequency at which call center status changes. Notifications can be more frequent when the number of calls in queue and the longest wait time in queue oscillate near the thresholds configured. To prevent continually sending status updates, the Application Server implements a status downgrade delay period before downgrading the call center status (from Threshold Exceeded to Normal or from Normal to Empty).

The maximum rate of notification is three notifications per second per call center, but is typically much less. The rate of notification can be reduced by increasing the *bw.callcenter.statusDowngradeDelayPeriod* parameter at the Application Server command line interface (CLI).

```
AS_CLI/System/StartupParam> get
  bw.callcenter.statusDowngradeDelayPeriod = 1
```

12 Call Park Event Package

12.1 Overview

The Call Park service allows a “parking” user to park a call against a “parked against” extension. The “parked” user is placed on hold until a user retrieves the parked call. The Call Park event package improves the end-user experience by providing the user with an indication of calls currently parked.

Prior to this enhancement, when a call was parked against an extension, the user had to be notified by some external system, such as an intercom system; “Bob, pick up on Line 1”. If Bob stepped out or was on a call with another customer, Bob may not have realized that a call was parked against his extension.

This event package sends notifications to the user’s devices so that an indication can be provided to the user that a call has been parked against his extension.

The synchronization protocol is based on the SIP-events framework and uses the *x-broadworks-callpark* event package (referred as the call park event package in the text).

The call park event package is an instantiation of the SIP-specific event notification framework (as defined in *RFC 3265*). For the applications described earlier, the Application Server acts as the “notifier” whereas the IP phones act as the “subscribers”.

Following is a description of the event package details, according to *RFC 3265*.

12.1.1 Interaction with Shared Call Appearance

If the user device already uses a subscription for Shared Call Appearance, then the Call Park information is provided over the Shared Call Appearance subscription instead of the Call Park subscription, to optimize messaging. The addition of the Call Park data in the Shared Call Appearance notifications is done through configuration of the Shared Call Appearance service and does not require an additional subscription. For more information, see sections *4 Call-Info Event Package* and *9 Alternate Signaling for Shared Call Appearance*.

12.2 Example Message Flow

Figure 30 shows a typical call flow for a user device subscribing to the Call Park Event Package. The Application Server immediately sends a NOTIFY message with the current Call Park state. Subsequently, a NOTIFY message is sent each time the Call Park state changes.

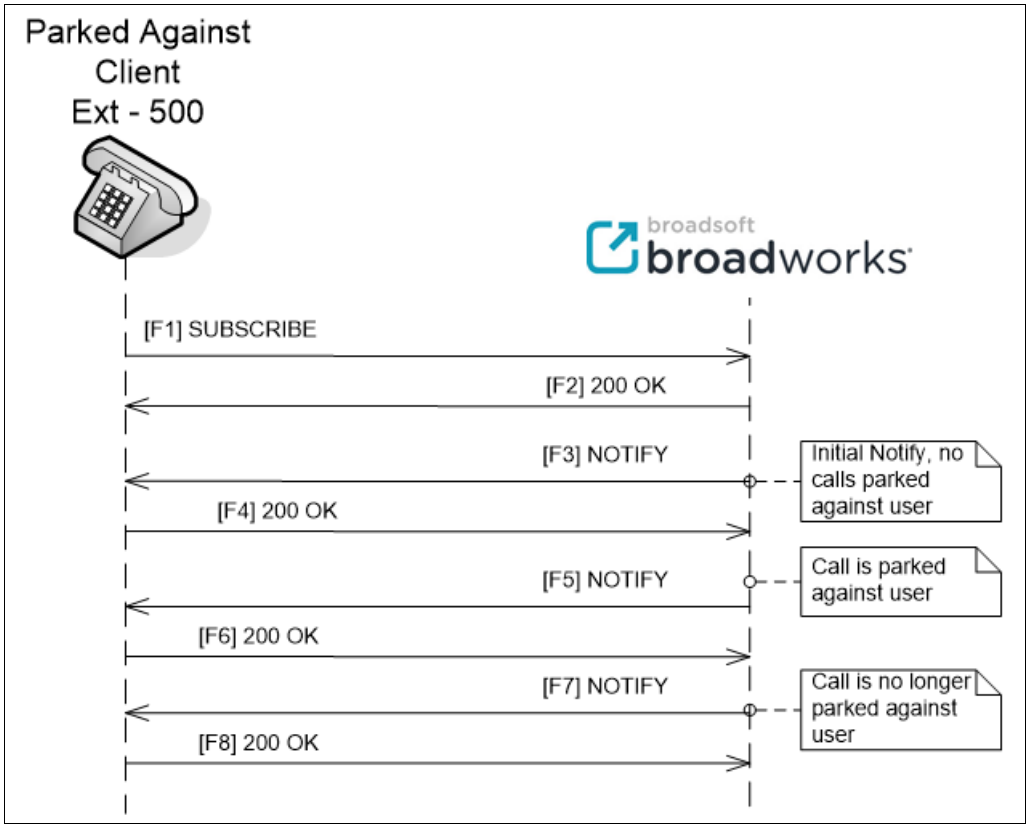


Figure 30 Subscription to Call Park Event Package

Initial subscription (the user does not have a call parked against them).

```
[F1] User 9726987500 -> Application Server
SUBSCRIBE sip:9726987500@as.bw.com:5060 SIP/2.0
Via: SIP/2.0/UDP c1.bw.com:5060;branch=z9hG4bK-76228-1-0
From: <sip:9726987500@txasdev87.net:5096>;tag=1
To: <sip:9726987500@as.bw.com:5060>
Call-ID: 1-76228@as.bw.com
CSeq: 1 SUBSCRIBE
Contact: sip:9726987500@c1.bw.com:5060
Expires:1800
Event:x-broadworks-callpark
Accept: application/x-broadworks-callpark-info+xml
Max-Forwards:10
Content-Length:0

    [F2] Application Server -> User 9726987500
SIP/2.0 200 OK
Via:SIP/2.0/UDP c1.bw.com:5060;branch=z9hG4bK-76228-1-0
From:<sip:9726987500@txasdev87.net:5096>;tag=1
To:<sip:9726987500@as.bw.com:5060>;tag=2036828995-1277837997983
Call-ID:1-76228@as.bw.com
CSeq:1 SUBSCRIBE
Expires:1798
Contact:<sip:as.bw.com:5060>
Content-Length:0
```

```
    [F3] Application Server -> User 9726987500
NOTIFY sip:9726987500@c1.bw.com:5060 SIP/2.0
Via:SIP/2.0/UDP as.bw.com;branch=z9hG4bKBroadWorks
From:<sip:9726987500@as.bw.com:5060>;tag=2036828995-1277837997983
To:<sip:9726987500@txasdev87.net:5096>;tag=1
Call-ID:1-76228@as.bw.com
CSeq:85227476 NOTIFY
Contact:<sip:as.bw.com:5060>
Event:x-broadworks-callpark
Subscription-State:active;expires=1798
Max-Forwards:10
Content-Type:application/x-broadworks-callpark-info+xml
Content-Length:215

<?xml version="1.0" encoding="UTF-8"?>
<x-broadworks-callpark-info
  xmlns="http://schema.broadsoft.com/callpark"
  version="0" state="FULL"
  entity="sip:north00@txasdev87.net">
 <callpark/>
</x-broadworks-callpark-info>

    [F4] User 9726987500-> Application Server
SIP/2.0 200 OK
Via:SIP/2.0/UDP as.bw.com;branch=z9hG4bKBroadWorks
From:<sip:9726987500@as.bw.com:5060>;tag=2036828995-1277837997983
To:<sip:9726987500@txasdev87.net:5096>;tag=1
Call-ID:1-76228@as.bw.com
CSeq:85227476 NOTIFY
Content-Length: 0
```

A call is parked against the user.

```
    [F5] Application Server -> User 9726987500
NOTIFY sip:9726987500@c1.bw.com:5060 SIP/2.0
Via:SIP/2.0/UDP as.bw.com;branch=z9hG4bKBroadWorks
From:<sip:9726987500@as.bw.com:5060>;tag=2036828995-1277837997983
To:<sip:9726987500@txasdev87.net:5096>;tag=1
Call-ID:1-76228@as.bw.com
CSeq:85260574 NOTIFY
Contact:<sip:as.bw.com:5060>
Event:x-broadworks-callpark
Subscription-State:active;expires=1765
Max-Forwards:10
Content-Type:application/x-broadworks-callpark-info+xml
Content-Length:329

<?xml version="1.0" encoding="UTF-8"?>
<x-broadworks-callpark-info
  xmlns="http://schema.broadsoft.com/callpark"
 <callpark>
  <parked>
   <identity display="Alice south">
    sip:876601@as.bw.com;user=phone
   </identity>
  </parked>
 </callpark>
</x-broadworks-callpark-info>
```

```
   [F6] User 9726987500-> Application Server
SIP/2.0 200 OK
Via:SIP/2.0/UDP as.bw.com;branch=z9hG4bKBroadWorks
From:<sip:9726987500@as.bw.com:5060>;tag=2036828995-1277837997983
To:<sip:9726987500@txasdev87.net:5096>;tag=1
Call-ID:1-76228@as.bw.com
CSeq:85260574 NOTIFY
Contact: <sip:c1.bw.com:5060;transport=UDP>
Content-Length: 0
```

A call is no longer parked against the user.

```
[F7] Application Server -> User 9726987500
NOTIFY sip:c1.bw.com:5060;transport=UDP SIP/2.0
Via:SIP/2.0/UDP as.bw.com;branch=z9hG4bKBroadWorks
From:<sip:9726987500@as.bw.com:5060>;tag=2036828995-1277837997983
To:<sip:9726987500@txasdev87.net:5096>;tag=1
Call-ID:1-76228@as.bw.com
CSeq:85267767 NOTIFY
Contact:<sip:as.bw.com:5060>
Event:x-broadworks-callpark
Subscription-State:active;expires=1758
Max-Forwards:10
Content-Type:application/x-broadworks-callpark-info+xml
Content-Length:215

<?xml version="1.0" encoding="UTF-8"?>
<x-broadworks-callpark-info
  xmlns="http://schema.broadsoft.com/callpark"
 <callpark/>
</x-broadworks-callpark-info>

   [F8] User 9726987500 -> Application Server
SIP/2.0 200 OK
Via:SIP/2.0/UDP as.bw.com;branch=z9hG4bKBroadWorks.
From:<sip:9726987500@as.bw.com:5060>;tag=2036828995-1277837997983
To:<sip:9726987500@txasdev87.net:5096>;tag=1
Call-ID:1-76228@as.bw.com
CSeq:85267767 NOTIFY
Contact: <sip:c1.bw.com:5060;transport=UDP>
Content-Length: 0
```

12.3 Event Package Name

The name of this event is "*x-broadworks-callpark*". It is carried in the *Event and Allow-Events* header, as defined in *RFC 3265*.

12.4 Event Package Parameters

This package does not define any event package parameters.

12.5 SUBSCRIBE Bodies

The SUBSCRIBE message is sent by the phone to the Application Server to obtain the current status and request future Call Park state change notifications for the duration of the subscription. The message does not contain a message body.

12.6 Subscription Duration

Subscribers should specify the duration of a subscription with an *Expires* header in the SUBSCRIBE request. It is recommended that the subscription use the same refresh period as the REGISTER event. If the notifier changes the expiration period to a lower value (via an *Expires* header in the final response), the subscriber should honor it.

If the subscriber does not specify the duration of a subscription, then the notifier chooses the default expiration.

12.7 NOTIFY Bodies

The NOTIFY message is sent by the Application Server to the phone. The NOTIFY message is triggered by a SUBSCRIBE message sent by the phone or a Call Park state change.

The NOTIFY request always includes one message body of type *application/x-broadworks-callpark-info+xml* defined as follows.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="http://schema.broadsoft.com/callpark"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:callpark="http://schema.broadsoft.com/callpark"
    elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xsd:annotation>
    <xsd:documentation>
      BroadWorks Call Park.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:element name="x-broadworks-callpark-info">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="callpark">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="parked" type="tns:participant"
                           minOccurs="0" maxOccurs="1"/>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    <xsd:complexType>
  </xsd:element>
</xsd:schema>
```

12.8 Subscriber Generation of SUBSCRIBE Requests

Subscriber (SIP phone) generation of SUBSCRIBE requests must follow all rules with respect to subscriber behavior, as described in *RFC 3265*.

SIP phones should generate SUBSCRIBE requests immediately after successfully registering. SUBSCRIBE requests should also be generated when the subscription approaches the expiration time. These SUBSCRIBE messages do not contain a message body.

12.9 Notifier Processing of SUBSCRIBE Requests

Notifier (Application Server) processing of SUBSCRIBE requests must follow all rules with respect to notifier behavior, as described in *RFC 3265*. The notifier may choose to explicitly challenge the subscriber for authentication by using a *401 Unauthorized* response. If the subscriber has also registered a location for this address of record, then it should use the same credentials to answer the subscription challenge.

12.10 Notifier Generation of NOTIFY Requests

Notifier (Application Server) generation of NOTIFY requests must follow all rules with respect to notifier behavior, as described in *RFC 3265* [3].

12.11 Subscriber Processing of NOTIFY Requests

In general, subscriber processing of NOTIFY requests must follow all rules with respect to subscriber behavior, as described in *RFC 3265*. The SIP-specific event notification framework expects packages to specify how a subscriber processes NOTIFY requests, and in particular, how it uses the NOTIFY requests to construct a coherent view of the state of the subscribed resource.

The NOTIFY messages contain the current Call Park state. The subscriber must discard any previous state information and replace it with the received information.

12.12 Rate of Notifications

The rate of notifications is dependent on the frequency at which the Call Park state changes. Typically, this occurs a few times per minute (at most).

13 Security Classification INFO Package

As part of the Security Classification service, Cisco BroadWorks can send an INFO message to an access device to indicate the current security classification for an active call. In accordance with RFC 6086 [8], these INFO requests operate within the definition of an INFO package. This section provides the specification of the Cisco BroadWorks Security Classification INFO package. For the details on Cisco BroadWorks Security Classification service, see the *Visual Security Classification for Active Call Feature Description, Release 20.0* [10].

The name of this INFO package for use in SIP messages is “x-broadworks-security-class”. This is the name that SIP user agents use in the *Recv-Info* header or *Info-Package* header.

If an access device is able to receive an INFO request for the security class INFO package, then either:

- When sending an initial INVITE request to Cisco BroadWorks, it must add a *Recv-Info* header with the value “x-broadworks-security-class”.

  Or,

- When sending a *200* response to an initial INVITE request from Cisco BroadWorks, it must add a *Recv-Info* header with the value “x-broadworks-security-class”.

When Cisco BroadWorks has security classification information to send to an access device, it sends an INFO request with that information, provided the access device indicated that it supports the security classification INFO package as described above. The INFO request contains an *Info-Package* header with the package name “x-broadworks-security-class”. Cisco BroadWorks adds the current classification level as the value of the *class* parameter in the *Info-Package* header.

RFC 6086 defines the following ABNF syntax for the *Info-Package* header and *Recv-Info* header.

```
Info-Package        =  "Info-Package" HCOLON Info-package-type
Recv-Info           =  "Recv-Info" HCOLON [Info-package-list]
Info-package-list   =  Info-package-type *( COMMA Info-package-type )
Info-package-type   =  Info-package-name *( SEMI Info-package-param )
Info-package-name   =  token
Info-package-param  =  generic-param
```

The syntax for Cisco's Security Classification INFO package is compatible with RFC syntax, and extends it as follows.

```
Info-package-name  /= "x-broadworks-security-class"
Info-package-param /= call-param
call-param          = "call" EQUAL quoted-string
```

Following is an example of the *Recv-Info* header.

```
Recv-Info: x-broadworks-security-class
```

Following is an example of the *Info-Package* header, with the current security level indicated as the value of the *class* parameter.

```
Info-Package: x-broadworks-security-class;class=secret
```

14 Client Session Information INFO Package

When a client application, such as Cisco's UC-One Communicator, participates in a call session involving the Cisco BroadWorks Application Server, there are occasions when the client needs to communicate application-level information to other clients. Cisco BroadWorks supports this client-to-client communication within a SIP dialog via INFO requests. In accordance with *RFC 6086*, these INFO requests operate within the definition of an INFO package. This section provides the specification of the Cisco BroadWorks Client Session Information INFO package. For the details of Cisco BroadWorks' support for client session information, see the *Business Communicator-Chat and Media Session Correlation Feature Description, Release 20.0* [11].

The name of this INFO package for use in SIP messages is “x-broadworks-client-session-info”. This is the name that SIP user agents use in the *Recv-Info* header or the *Info-Package* header.

An INFO request for the client session information package does not contain a message body. To send application-level information, the client adds an *X-BroadWorks-Client-Session-Info* header to the INFO request. Cisco BroadWorks stores the content of this header as ancillary information in the call session and may relay the information to other clients via an outgoing INFO request or through other means. However, Cisco BroadWorks does not interpret the content of the information, treating it as opaque information.

Although the purpose behind the client session information package is to facilitate the communication of application-level information from one client to another, Cisco BroadWorks does not proxy the INFO request between clients. Cisco BroadWorks does relay the client session information end to end, but it can use other means in addition to the INFO request to receive or send client session information. Therefore, the scope of the INFO request is the SIP dialog between Cisco BroadWorks and the device endpoint (that is, the client).

The following points clarify the use of the *Recv-Info* header:

- Cisco BroadWorks adds a *Recv-Info* header with “x-broadworks-client-session-info” to an initial INVITE request to an access device.
- When an access device sends a *200* response to an initial INVITE request from Cisco BroadWorks, it must include a *Recv-Info* header with “x-broadworks-client-session-info” if it is able to receive an INFO request for this package.
- When an access device sends an initial INVITE request to Cisco BroadWorks, it must include a *Recv-Info* header with “x-broadworks-client-session-info” if it is able to receive an INFO request for this package.
- When Cisco BroadWorks sends a *200* response to an initial INVITE request from an access device, it includes a *Recv-Info* header with “x-broadworks-client-session-info”, provided the INVITE request had a *Recv-Info* header with “x-broadworks-client-session-info”.

The following points clarify the use of the INFO request and *Info-Package* header:

- If Cisco BroadWorks has client session information to send to an access device, it may send that information in an INFO request to the access device. Cisco BroadWorks sends the INFO request only if it received an indication from the access device, via a *Recv-Info* header, that it supports the client session information INFO package. The INFO request contains an *Info-Package* header with “x-broadworks-client-session-info” and an *X-BroadWorks-Client-Session-Info* header that contains the client session information.

- If an access device has client session information to communicate to another client, it may send that information in an INFO request to Cisco BroadWorks. The access device should send the INFO request only if it received an indication, via a *Recv-Info* header, that Cisco BroadWorks supports the client session information INFO package. The INFO request must contain an *Info-Package* header with "x-broadworks-client-session-info" and must have an *X-BroadWorks-Client-Session-Info* header that contains the client session information.

RFC 6086 defines the following ABNF syntax for the *Info-Package* header and *Recv-Info* header.

```
Info-Package         =  "Info-Package" HCOLON Info-package-type
Recv-Info            =  "Recv-Info" HCOLON [Info-package-list]
Info-package-list    =  Info-package-type *( COMMA Info-package-type )
Info-package-type    =  Info-package-name *( SEMI Info-package-param )
Info-package-name    =  token
Info-package-param   =  generic-param
```

The syntax for Cisco's client session information INFO package is compatible with RFC syntax and extends it as follows.

```
Info-package-name  /= "x-broadworks-client-session-info"
```

Following is an example of the *Recv-Info* header.

```
Recv-Info: x-broadworks-client-session-info
```

Following is an example of the *Info-Package* header and the *X-BroadWorks-Client-Session-Info* header that convey the client session information.

```
Info-Package: x-broadworks-client-session-info
X-BroadWorks-Client-Session-Info: qwerty-asdfg
```

Acronyms and Abbreviations

ABNF	Augmented Backus-Naur Format
ACD	Automatic Call Distribution
AoR	Address of Record
AS	Application Server
AS-O	Application Server - Originating
AS-T	Application Server - Terminating
CFA	Call Forwarding Always
CFB	Call Forwarding Busy
CFNA	Call Forwarding No Answer
CLI	Command Line Interface
CSTA	Computer Supported Telecommunications Applications
CTI	Computer Telephony Integration
DN	Directory Number
DND	Do Not Disturb
DNS	Domain Name System
ECMA	European Computer Manufacturers Association
FAC	Feature Access Code
IP	Internet Protocol
LCD	Liquid Crystal Display
LED	Light-Emitting Diode
MGCP	Media Gateway Control Protocol
PBX	Private Branch Exchange
PSTN	Public Switched Telephone Network
RFC	Request for Comments
SCA	Shared Call Appearance
SCCP	Simple Conference Control Protocol
SDP	Session Description Protocol
SIP	Session Initiation Protocol
TCP	Transmission Control Protocol
UA	User Agent
UAC	User Agent Client
UAS	User Agent Server
UDP	User Datagram Protocol
URI	Uniform Resource Identifier

References

[1] Standard ECMA-323. 2006. *XML Protocol for Computer Supported Telecommunications Applications (CSTA) Phase III.* Available from http://www.ecma-international.org/publications/standards/Ecma-323.htm.

[2] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J, Sparks R., Handley, M., and Schooler, E., "SIP: Session Initiation Protocol", RFC 3261, Internet Engineering Task Force, June 2002. Available from http://www.ietf.org/.

[3] Roach, A.B., "SIP-Specific Event Notification", RFC 6665, Internet Engineering Task Force, July 2012. Available from http://www.ietf.org/.

[4] Mahy, R., Petrie, D., "The Session Initiation Protocol (SIP) "Join" Header", RFC 3911, Internet Engineering Task Force, October 2004. Available from http://www.ietf.org/.

[5] Mahy, R. & al., "The Session Initiation Protocol (SIP) *Replaces* Header", RFC 3891, Internet Engineering Task Force, September 2004. Available from http://www.ietf.org/.

[6] Rosenberg & al., "An INVITE-Initiated Dialog Event Package for the Session Initiation Protocol (SIP)", RFC 4235, Internet Engineering Task Force, November 2005. Available from http://www.ietf.org/.

[7] Roach, A.B., Campbell, B., Rosenberg, J., "A Session Initiation Protocol (SIP) Event Notification Extension for Resource Lists", RFC 4662, Internet Engineering Task Force, August 2006. Available from http://www.ietf.org/.

[8] Holmberg, C., Burger, E., Kaplan, H., "Session Initiation Protocol (SIP) INFO Method and Package Framework", RFC 6086, Internet Engineering Task Force, January 2011. Available from http://www.ietf.org/.

[9] Johnston, A., Ed, Soroushnejad, M., Ed., Venkataramanan, V., "Shared Appearances of a Session Initiation Protocol (SIP) Address of Record (AOR)", RFC 7463, Internet Engineering Task Force, March 2015. Available from http://www.ietf.org/.

[10] Cisco Systems, Inc. 2013. *Cisco BroadWorks Visual Security Classification for Active Call Feature Description, Release 20.0.* Available from Cisco at cisco.com.

[11] Cisco Systems, Inc. 2013. *Business Communicator-Chat and Media Session Correlation Feature Description, Release 20.0.* Available from Cisco at cisco.com.

Index

