Event Notification in SIP

Status of this Memo

This document is an Internet-Draft and is in full conformance with all provisions of Section 10 of RFC2026.

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

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

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

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

This document is an individual submission to the IETF. Comments should be directed to the authors.

Abstract

This document describes an extension to the Session Initiation Protocol (SIP) [1]. The purpose of this extension is to provide a generic and extensible framework by which SIP nodes can request notification from remote nodes indicating that certain events have occurred.

Concrete uses of the mechanism described in this document may be standardized in the future.

1. Table of Contents

   1. Table of Contents .................................................. 1
   2. Introduction ....................................................... 2
   2.1. Overview of Operation .......................................... 3
   3. Extension Considerations ......................................... 3
   3.1. Appropriateness of Usage ...................................... 3
   3.2. Additional Guidelines ......................................... 4
   4. Syntax ............................................................. 4
2. Introduction

The ability to request asynchronous notification of events proves useful in many types of services for which cooperation between end-nodes is required. Examples of such services include automatic callback services (based on terminal state events), buddy lists (based on user presence events), message waiting indications (based on mailbox state change events), and PINT status (based on call state events).

The methods described in this document allow a framework by which notification of these events can be ordered.

Note that the event notification mechanisms defined herein are NOT intended to be a general-purpose infrastructure for all classes of event subscription and notification. Meeting
requirements for the general problem set of subscription and notification is far too complex for a single protocol. Our goal is to provide a general framework for event notification which is not so complex as to be unusable for simple features, but which is still flexible enough to provide powerful services. However, extensions based on this framework may define arbitrarily complex rules which govern the subscription and notification for the events or classes of events they describe.

Note that this draft does not describe an extension which may be used directly; it must be extended by other drafts (herein referred to as "extension drafts" and "event packages.") In object-oriented design terminology, it may be thought of as an abstract base class which must be derived into an instantiatable class by further extensions. Guidelines for creating these extensions are described in section 3.

2.1. Overview of Operation

The general concept is that entities in the network can subscribe to resource or call state for various resources or calls in the network, and those entities (or entities acting on their behalf) can send notifications when those states change.

A typical flow of messages would be:

<table>
<thead>
<tr>
<th>Subscriber</th>
<th>Notifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;-----SUBSCRIBE-----&quot;</td>
<td>Request state subscription</td>
</tr>
<tr>
<td>&quot;--------200--------&quot;</td>
<td>Return current state information</td>
</tr>
<tr>
<td>&quot;--------NOTIFY------&quot;</td>
<td>Return current state information</td>
</tr>
<tr>
<td>&quot;--------200--------&quot;</td>
<td></td>
</tr>
<tr>
<td>&quot;--------NOTIFY------&quot;</td>
<td>Return current state information</td>
</tr>
<tr>
<td>&quot;--------200--------&quot;</td>
<td></td>
</tr>
</tbody>
</table>

The subscriber and notifier entities need not necessarily be UAs, but often will be.

Subscriptions are expired and must be refreshed in exactly the same manner as registrations (see RFC 2543 [1]).

3. Extension Considerations

This section covers several issues which should be taken into consideration when SIP extensions based on SUBSCRIBE and NOTIFY are proposed.

3.1. Appropriateness of Usage

When using the methods described in this draft for event
notification, it is important to consider: is SIP an appropriate mechanism for the problem set? Is SIP being selected because of some unique feature provided by the protocol (e.g. user mobility), or merely because "it can be done?" If you find yourself defining SIP extensions for notifications related to, for example, network management or the temperature inside your car’s engine, you may want to reconsider your selection of protocols.

Those interested in extending the mechanism defined in this document are urged to read "Guidelines for Authors of SIP Extensions" [3] for further guidance regarding appropriate uses of SIP.

Further, it is expected that this mechanism is not to be used in applications where the frequency of reportable events is excessively rapid (e.g. more than about once per second). A SIP network is generally going to be provisioned for a reasonable signalling volume; sending a notification every time a user’s GPS position changes by one hundreth of a second could easily overload such a network.

3.2. Additional Guidelines

When writing extensions based on SUBSCRIBE and NOTIFY, it is important to consider the type of information which will be conveyed during a notification.

A natural temptation is to convey merely the event (e.g. "a new voice message just arrived") without accompanying state (e.g. "7 total voice messages"). This complicates implementation of subscribing entities (since they have to maintain complete state for the entity to which they have subscribed), and also is particularly susceptible to synchronization problems.

It is therefore suggested that extensions are designed so as to notify of new state when an event occurs. In the circumstances that state may not be sufficient for a particular class of events, the extensions should include complete state information along with the event that occurred. (For example, "no customer service representatives available" may not be as useful "no customer service representatives available; representative sip:46@cs.xyz.int just logged off".)

4. Syntax

This section describes the syntax extensions required for event notification in SIP. Semantics are described in section 5.

4.1. New Methods

Roach [Page 4]
This document describes two new SIP methods: "SUBSCRIBE" and "NOTIFY."

This table expands on tables 4 and 5 in RFC 2543 [1].
<table>
<thead>
<tr>
<th>Header</th>
<th>Where</th>
<th>SUB NOT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accept</td>
<td>R</td>
<td>o o</td>
</tr>
<tr>
<td>Accept-Encoding</td>
<td>R</td>
<td>o o</td>
</tr>
<tr>
<td>Accept-Language</td>
<td>R</td>
<td>o o</td>
</tr>
<tr>
<td>Allow</td>
<td>200</td>
<td>- -</td>
</tr>
<tr>
<td>Allow</td>
<td>405</td>
<td>o o</td>
</tr>
<tr>
<td>Authorization</td>
<td>R</td>
<td>o o</td>
</tr>
<tr>
<td>Call-ID</td>
<td>gc</td>
<td>m m</td>
</tr>
<tr>
<td>Contact</td>
<td>R</td>
<td>m -</td>
</tr>
<tr>
<td>Contact</td>
<td>1xx</td>
<td>m o</td>
</tr>
<tr>
<td>Contact</td>
<td>2xx</td>
<td>m o</td>
</tr>
<tr>
<td>Contact</td>
<td>3xx</td>
<td>m m</td>
</tr>
<tr>
<td>Contact</td>
<td>485</td>
<td>o o</td>
</tr>
<tr>
<td>Content-Encoding</td>
<td>e</td>
<td>o o</td>
</tr>
<tr>
<td>Content-Length</td>
<td>e</td>
<td>o o</td>
</tr>
<tr>
<td>Content-Type</td>
<td>e</td>
<td>* *</td>
</tr>
<tr>
<td>CSeq</td>
<td>gc</td>
<td>m m</td>
</tr>
<tr>
<td>Date</td>
<td>g</td>
<td>o o</td>
</tr>
<tr>
<td>Encryption</td>
<td>g</td>
<td>o o</td>
</tr>
<tr>
<td>Expires</td>
<td>g</td>
<td>o o</td>
</tr>
<tr>
<td>From</td>
<td>gc</td>
<td>m m</td>
</tr>
<tr>
<td>Hide</td>
<td>R</td>
<td>o o</td>
</tr>
<tr>
<td>Max-Forwards</td>
<td>R</td>
<td>o o</td>
</tr>
<tr>
<td>Organization</td>
<td>g</td>
<td>o o</td>
</tr>
<tr>
<td>Priority</td>
<td>R</td>
<td>o o</td>
</tr>
<tr>
<td>Proxy-Authenticate</td>
<td>407</td>
<td>o o</td>
</tr>
<tr>
<td>Proxy-Authorization</td>
<td>R</td>
<td>o o</td>
</tr>
<tr>
<td>Proxy-Require</td>
<td>R</td>
<td>o o</td>
</tr>
<tr>
<td>Require</td>
<td>R</td>
<td>o o</td>
</tr>
<tr>
<td>Retry-After</td>
<td>R</td>
<td>- -</td>
</tr>
<tr>
<td>Retry-After</td>
<td>404,480,486</td>
<td>o o</td>
</tr>
<tr>
<td>Retry-After</td>
<td>503</td>
<td>o o</td>
</tr>
<tr>
<td>Retry-After</td>
<td>600,603</td>
<td>o o</td>
</tr>
<tr>
<td>Response-Key</td>
<td>R</td>
<td>o o</td>
</tr>
<tr>
<td>Record-Route</td>
<td>R</td>
<td>o o</td>
</tr>
<tr>
<td>Record-Route</td>
<td>2xx</td>
<td>o o</td>
</tr>
<tr>
<td>Route</td>
<td>R</td>
<td>o o</td>
</tr>
<tr>
<td>Server</td>
<td>r</td>
<td>o o</td>
</tr>
<tr>
<td>Subject</td>
<td>R</td>
<td>o o</td>
</tr>
<tr>
<td>Timestamp</td>
<td>g</td>
<td>o o</td>
</tr>
<tr>
<td>To</td>
<td>gc(1)</td>
<td>m m</td>
</tr>
<tr>
<td>Unsupported</td>
<td>420</td>
<td>o o</td>
</tr>
<tr>
<td>User-Agent</td>
<td>g</td>
<td>o o</td>
</tr>
<tr>
<td>Via</td>
<td>gc(2)</td>
<td>m m</td>
</tr>
<tr>
<td>Warning</td>
<td>r</td>
<td>o o</td>
</tr>
<tr>
<td>WWW-Authenticate</td>
<td>401</td>
<td>o o</td>
</tr>
</tbody>
</table>

4.1.1. SUBSCRIBE method
"SUBSCRIBE" is added to the definition of the element "Method" in the SIP message grammar.

Like all SIP method names, the SUBSCRIBE method name is case sensitive. The SUBSCRIBE method is used to request asynchronous notification of an event or set of events at a later time.

**4.1.2. NOTIFY method**

"NOTIFY" is added to the definition of the element "Method" in the SIP message grammar.

The NOTIFY method is used to notify a SIP node that an event which has been requested by an earlier SUBSCRIBE method has occurred. It may also provide further details about the event.

**4.2. New Headers**

This table expands on tables 4 and 5 in RFC 2543 [1], as amended by the changes described in section 4.1.

<table>
<thead>
<tr>
<th>Header field</th>
<th>where</th>
<th>proxy</th>
<th>ACK</th>
<th>BYE</th>
<th>CAN</th>
<th>INV</th>
<th>OPT</th>
<th>REG</th>
<th>SUB</th>
<th>NOT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allow-Events</td>
<td>g</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>Event</td>
<td>R</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>m</td>
<td>m</td>
<td></td>
</tr>
<tr>
<td>Event</td>
<td>r</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>m</td>
<td>m</td>
<td>-</td>
</tr>
</tbody>
</table>

**4.2.1. "Event" header**

The following header is defined for the purposes of this extension.

```
Event = "Event" ":" event-type
   *(( ":" parameter-name ["=" token] )
   ( ":" parameter-name ["=" quoted-string] ))
```

event-type = token

Event is added to the definition of the element "general-header" in the SIP message grammar.

This document does not define values for event-types. These values will be defined in further extensions that take advantage of the SUBSCRIBE and NOTIFY methods, and SHOULD be registered with the IANA.

Note that experimental event types may be created by prepending the organization’s internet domain, with the field order reversed.

Roach [Page 7]
(e.g. "Event: com.ericsson.foo").

4.2.2. "Allow-Events" Header

The following header is defined for the purposes of this extension.

Allow-Events = "Allow-Events" ":

Allow-Events is added to the definition of the element "general-header" in the SIP message grammar.

4.3. "489 Bad Event" Response Code

The 489 event response is added to the "Client-Error" header field definition:

Client-Error = "400" ; Bad Request

| "489" ; Bad Event

"489 Bad Event" is used to indicate that the server did not understand the event class specified in a "Event" header field.

5. Node Behavior

Unless noted otherwise, call-member SUBSCRIBE and NOTIFY requests follow the same protocol rules governing the usage of tags, Route, Record-Route, Via handling, retransmission, reliability, CSeq handling, Contact handling, and message formatting as those defined in RFC 2543 [1] for BYE.

Similarly, unless noted otherwise, third-party SUBSCRIBE and NOTIFY requests follow the same protocol rules as those defined in RFC 2543 [1] for OPTIONS.

Note that neither SUBSCRIBE nor NOTIFY necessitate the use of "Require" or "Proxy-Require" headers; similarly, there is no token defined for "Supported" headers. If necessary, clients may probe for the support of SUBSCRIBE and NOTIFY using the OPTIONS request defined in RFC2543.

For the purposes of generality, both SUBSCRIBE and NOTIFY MAY be canceled; however, doing so is not recommended. Successfully cancelled SUBSCRIBE and NOTIFY requests MUST be completed with a "487 Request Cancelled" response; the server acts as if the request were never received.
SUBSCRIBE and NOTIFY have two slightly different uses: (1) subscription to and notification of events by nodes which are involved in an ongoing call with the node from which notification is being requested, and (2) subscription to and notification of events by nodes which are not actively involved as an endpoint in an ongoing call with the node from which notification is being ordered. For the purposes of brevity, these situations will be referred to as "call-member" and "third-party" respectively.

Further, third-party SUBSCRIBE and NOTIFY requests may relate to call-related events (e.g. "call terminated") or resource-related events (e.g. "terminal free," or "user present").

5.1. Description of SUBSCRIBE Behavior

5.1.1. Correlation to legs, calls, and terminals

A subscription is uniquely identified by the combination of the To, Call-ID and the From field in the SUBSCRIBE request. Refreshes of subscriptions SHOULD reuse the same Call-ID if possible, since subscriptions are uniquely identified at presence servers using the Call-ID. Two subscriptions from the same user, for the same user, but with different Call-IDs, are considered different subscriptions. Note this is exactly the same as usage of Call-ID in registrations.

Call-member SUBSCRIBE requests will be correlated in the same way that any other call-related request is (e.g. BYE) using To, From, and Call-ID; these subscriptions will generally be used to request information about the specific call.

Third-party SUBSCRIBE requests will not correlate to any previously-existing call leg in the server. The Call-ID of resource-related requests will be unique to the SUBSCRIBE and any subsequent NOTIFY requests.

Third-party SUBSCRIBE requests may also request information about call-related events by specifying a Call-ID that is known to have significance to the UAS. Proxies in the call-setup path may obtain this Call-ID by examining the messages they proxy; other methods of obtaining the Call-ID of an ongoing call are outside the scope of this document.

Note that third-party subscriptions have security implications; see section 7.

5.1.2. Subscription duration

All SUBSCRIBE requests are required to contain an "Expires" header. This header specifies for how long the subscription is to
remain in effect. The 200 response to a SUBSCRIBE request also MUST contain an "Expires" header. The period of time in the response MAY be shorter than specified in the request, but MUST NOT be longer. The period of time in the response is the one which defines the duration of the subscription.

Similar to REGISTER requests, SUBSCRIBE requests may be renewed at any time to prevent them from expiring at the end of the "Expires" period. These renewals will contain a the same "To," "From," and "Call-ID" as the original request, and an incremented "CSeq" number. Subscriptions may similarly be cancelled by re-issuing them with an "Expires: 0" header.

Further, call-related SUBSCRIBE requests (as opposed to resource-related) are automatically expired at the end of the call to which they relate (from the point of view of the device receiving the SUBSCRIBE request), regardless of the "Expires" values in the SUBSCRIBE response.

5.1.3. Identification of Subscribed Events and Event Classes

Identification of events is provided by three pieces of information: Request URI, Event Type, and (optionally) message body.

The Request URI of a SUBSCRIBE request, most importantly, contains enough information to route the request to the appropriate entity. It also contains enough information to identify the resource for which event notification is desired, but not necessarily enough information to uniquely identify the nature of the event (e.g. "sip:adam.roach@ericsson.com" would be an appropriate URI to subscribe to for my presence state; it would also be an appropriate URI to subscribe to the state of my voice mailbox).

The "Event" header will contain a single opaque token which identifies the event or class of events for which a subscription is being requested. This token will be registered with the IANA and will correspond to an extension draft which further describes the semantics of the event or event class.

The "Event" header is considered mandatory for the purposes of this document. However, to maintain compatibility with PINT [4], servers MAY interpret a SUBSCRIBE request with no "Event" header as requesting a subscription to PINT events. If the servers do not support PINT, they SHOULD instead return "400 Bad Request."

If the extension draft to which the event token corresponds defines behavior associated with the body of its SUBSCRIBE
requests, those semantics apply. It is expected that most, but
not all, extension drafts will define syntax and semantics for
SUBSCRIBE method bodies; these bodies will typically modify,
expand, filter, throttle, and/or set thresholds for the class of
events being requested. Designers of extensions are strongly
encouraged to re-use existing MIME types for message bodies where
feasible.

5.1.4. Subscriber SUBSCRIBE Behavior

When a subscriber wishes to subscribe to (or refresh a
subscription to) an event class, he forms a SUBSCRIBE message. If
the subscription is call related, the call leg information (To,
From, Call-ID) are used in the SUBSCRIBE message as if it were
any other request related to the call.

For non-call-related subscriptions, the call leg information is
formed as if for an original INVITE: the Call-ID is a new call ID
with the syntax described in RFC 2543; the To: field indicates
the subscribed resource’s persistent address (which will generally
match the Request URI used to form the message); and the From:
field will indicate the subscriber’s persistent address
(typically sip:user@machine for UAs, or sip:machine for other
entities).

In both circumstances, the Contact: header(s) will contain
information about where resulting NOTIFY requests are to be sent.
Multiple Contact headers are allowed, and indicate multiple
destinations to which NOTIFY requests should be sent.

Subscribers MUST include an "Event" header in SUBSCRIBE requests,
indicating to which event or class of events they are
subscribing. This header contains a token which corresponds to an
extension draft; see section 5.1.3.

SUBSCRIBE requests MAY contain an "Accept" header. This header,
if present, indicates the body formats allowed in both the
SUBSCRIBE response and subsequent NOTIFY requests. Extensions
making use of SUBSCRIBE and NOTIFY MUST define the behavior for
SUBSCRIBE requests without "Accept" headers; usually, this will
connote a single, default body type.

SUBSCRIBE requests MUST contain either an "Expires" header or
"expires" parameters on each "Contact:" header. These expires
values indicate the duration of the subscription. The formatting
of these is described in RFC 2543. In order to keep subscriptions
effective beyond the duration communicated in the "Expires"
header, subscribers need to refresh subscriptions on a periodic
basis. This refreshing is performed in the same way as REGISTER
refreshes: the To, From, and Call-ID match those in the SUBSCRIBE
being refreshed, while the CSeq number is incremented.

Similar to REGISTER, a natural consequence of this scheme is that a SUBSCRIBE with an "Expires" of 0 constitutes a request to unsubscribe from an event.

5.1.5. Proxy SUBSCRIBE Behavior

Proxies need no additional behavior beyond that described in RFC 2543 [1] to support SUBSCRIBE. Note that SIP proxies may also act as subscribers or notifiers, as appropriate; under these circumstances, they will act as described in 5.1.4. and 5.1.6.

Due to CSeq space collisions, proxies are not allowed to initiate call-member SUBSCRIBE requests, and must instead use third-party SUBSCRIBE requests. They do so by including the Call-ID of an ongoing call. The "From:" field in this case, however, would indicate an address owned by the proxy, and thus be considered a new leg by the UAS.

5.1.6. Notifier SUBSCRIBE Behavior

Upon receipt of a SUBSCRIBE request, the notifier MAY send a 100 response.

The notifier SHOULD check that the event package specified in the "Event" header is understood. If not, the notifier SHOULD return a "489 Bad Event" response to indicate that the specified event/event class is not understood.

The notifier SHOULD also perform any necessary authentication per its local policy. SIP authentication mechanisms are discussed in RFC 2543. Note that, even if the notifier node typically acts as a proxy, authentication for SUBSCRIBE requests will always be performed via a "401" response, not a "407;" notifiers always act as a user agents when accepting subscriptions and sending notifications.

If authorization is not granted, the notifier SHOULD return a 600-class response.

Once the notifier has determined that it understands the event package and that the authenticated subscriber is authorized to subscribe, it returns a "200 OK" response. The "Expires" values present in this 200 response behave in the same way as they do in REGISTER responses: the server MAY shorten the interval, but MUST not increase it.

The exact contents of a 200 class response will vary according to the event package being used; in general, though, such responses
should contain the same amount of state information as in a
"NOTIFY" request. In other words, a SUBSCRIBE will serve both as
a subscription to event information and as an immediate fetch of
current related state information.

A natural consequence of the described behavior is that an
immediate fetch without a persistent subscription may be effected
by sending an appropriate SUBSCRIBE with an "Expires" of 0. Of
course, an immediate fetch while a subscription is pending may be
effected by sending an appropriate SUBSCRIBE with an "Expires"
greater than 0.

Note that privacy concerns may require that notifiers either use
access lists or ask the user on a per-subscription basis whether
a particular remote node is allowed to subscribe to a certain set
of events. If such authorization fails, the notifier should reply
to the request with a "403 Forbidden" request. See section 7.

The other response codes defined in RFC2543 may be used in
response to SUBSCRIBE requests, as appropriate.

When a notifier receives a subscription refresh, assuming that
the subscriber is still authorized, the notifier updates the
expiration time of each notification contact address in the
subscription. As with the initial subscription, the server MAY
lower the amount of time until expiration, but MUST NOT increase
it. The final expiration time is placed in the Expires header in
the response, or into the expires parameters of the Contact
headers in the response.

If no refresh for a notification address is received before its
expiration time, that address is removed from the list of
addresses. If all notification addresses are removed, the entire
subscription is deleted.

5.2. Description of NOTIFY Behavior

5.2.1. Correlation

NOTIFY requests MUST contain the same Call-ID, local URI, and
remote URI as the SUBSCRIBE request which ordered them. This is
the same set of criteria that define a call leg.

The From field of a NOTIFY request MUST contain a tag; this
allows for the subscriber to differentiate between events from
different notifiers.

Since forking proxies pass all 200 responses upstream, it is
expected that these "From" fields will not contain any tags which
are unknown to the subscriber. However, to make the subscription
mechanism more robust, subscribers SHOULD be prepared to receive notifications with previously unknown tags in the "From" field.

As expected, CSeq spaces are unique for each node; in other words, the notifier uses a different CSeq space than the subscriber and any other notifiers.

5.2.2. Identification of reported events, event classes, and current state

Identification of events being reported in a notification is very similar to that described for subscription to events (see section 5.1.3.).

The Request URI of a NOTIFY request contains enough information to route the request to the party which is subscribed to receive notifications. It is derived from the "Contact" header present in the corresponding SUBSCRIBE request.

If the same events for different resources are being subscribed to, implementors are expected to use different "Call Legs" (To, From, Call-ID) in order to be able to differentiate between notifications for them, unless the body for the event contains enough information for this correlation. Note that this shouldn’t often pose any difficulty, since the "To" field almost always matches the Request-URI on the originator’s side.

As in SUBSCRIBE requests, NOTIFY "Event" headers will contain a single opaque token which identifies the event or class of events for which a subscription is being requested.

If the extension draft to which the event token corresponds defines behavior associated with the body of its NOTIFY requests, those semantics apply. This information is expected to provide additional details about the nature of the event which has occurred and the resultant resource state.

When present, the body of the NOTIFY request MUST be formatted into one of the body formats specified in the "Accept" header of the corresponding SUBSCRIBE request. The formatting rules and behavior when no "Accept" header is expected to be defined by the document which describes the relevant event package.

Note that sending a NOTIFY does not cancel the SUBSCRIBE which requested it; in other words, a single SUBSCRIBE request may trigger several NOTIFY requests.

Further, NOTIFY requests MAY be sent without a matching SUBSCRIBE under certain circumstances. It may make sense, for example, to set up a subscription using an out-of-band mechanism (e.g. HTTP,

static provisioning). A subscriber which is designed to operate in this fashion MUST be prepared to receive NOTIFY requests without a corresponding call leg.

5.2.3. Notifier NOTIFY Behavior

When a relevant change in the subscribed state occurs, the notifier will construct and send a NOTIFY request to the subscriber(s). Such a message should be sent in as timely a manner as is practical.

If the notifier is able, through any means, to determine that the subscriber is no longer available to receive notifications, it MAY elect to not send a notification. An example of a method by which such information may be known is the "SIP for Presence" event set (see [5] ).

If the original subscription contained a "Record-Route" and/or "Contact" headers, notifications are sent according to the rules outlined in RFC 2543 [1], as if the SUBSCRIBE were an INVITE, and the NOTIFY were any subsequent message (e.g. BYE).

Notify requests SHOULD NOT contain a "Contact" header.

A NOTIFY request is considered failed if the response times out, or a non-200 class response code is received which has no "Retry-After" header and no implied further action which can be taken to retry the request (e.g. "401 Authorization Required.").

If the response to a NOTIFY request fails, the notifier SHOULD remove the contact from the appropriate subscription. If removal of the contact leaves no remaining contacts, the entire subscription is removed.

5.2.4. Proxy NOTIFY Behavior

Proxies need no additional behavior beyond that described in RFC 2543 [1] to support NOTIFY.

5.2.5. Subscriber NOTIFY Behavior

Upon receiving a NOTIFY request, the subscriber should check that it matches at least one of its outstanding subscriptions; if not, it SHOULD return a "481 Call leg/transaction does not exist" response. It is RECOMMENDED that it also send a SUBSCRIBE request with "Expires: 0," "Contact: *," and "Event:" copied from the NOTIFY request to cancel the subscription which triggered the errant notification.

The previous paragraph notwithstanding, subscribers which are
designed to receive notifications for events subscribed to by an out-of-band mechanism MUST NOT return a 481 response code for any notifications, and should instead process them as if it had previous knowledge of the subscription.

To prevent spoofing of events, NOTIFY requests MAY be authenticated, using any defined SIP authentication mechanism.

Once the notification is deemed acceptable to the subscriber, the subscriber SHOULD return a 200 response. In general, it is not expected that NOTIFY responses will contain bodies; however, they MAY, if the NOTIFY request contained an "Accept" header.

Other responses defined in RFC 2543 [1] may also be returned, as appropriate.

5.3. Allow-Events header usage

The "Allow-Events" header, if present, includes a list of tokens which indicate the event packages supported by the client (if sent in a request) or server (if sent in a response).

Any node implementing one or more event packages SHOULD include an appropriate "Allow-Events" header indicating all supported events in INVITE requests and responses, OPTIONS responses, and REGISTER requests. "Allow-Events" headers MAY be included in any other type of request or response.

This information is very useful, for example, in allowing user agents to render particular interface elements appropriately according to whether the events required to implement the features they represent are supported by the appropriate nodes.

6. Open Issues

6.1. Resource identification for out-of-band subscriptions

In a SUBSCRIBE request, the request URI is used to identify the resource (although not the event) to which a subscription is requested. If there is no explicit SUBSCRIBE, this information doesn’t really exist anywhere. For certain event types, this may pose a problem. This hardly seems a reason to explicitly require a SUBSCRIBE to be sent prior to any NOTIFY, though.

This can be solved by adding something like a "resource=URI" parameter to the "Event:" header (or an additional header, such as "Event-URI") in NOTIFY requests. This approach also lends a certain robustness to the SUBSCRIBE/NOTIFY mechanism, since any lost subscription state information can be reconstructed by the subscriber as NOTIFY requests arrive. It also allows
differentiation of resources for subscriptions that share a common To, From, and Call-ID.

The author would appreciate feedback on whether such a mechanism is considered appropriate and useful.

6.2. Expiration of call-related subscriptions when the call is over

Call-related subscriptions disappear at the end of the call to which they relate. Is this an unnecessary restriction? It seems to make the implementation in the notifier simpler. Obviously, this doesn’t preclude end-of-call statistics (which can be sent right at the end of the call, just as the subscription is about to disappear). Are there any realistic uses for notification of call-related events long after the call to which they relate is terminated?

6.3. Forking behavior

Forking of a SUBSCRIBE request may have unintended consequences; however, they don’t seem to normally pose any undesirable behavior. Under normal proxy processing, the SUBSCRIBE request will be cancelled for all outstanding branches once the first 200 response is received. Since many notifiers will respond to a SUBSCRIBE immediately (and proxies are required to pass all 200s, not just the first one they see), it is most likely that forked SUBSCRIBE requests will cause the subscriber to receive a 200 response for every branch. The exceptions to this assumption will be, for example, notifiers that interactively ask the user for permission before accepting a subscription.

The only really undesirable behavior arises from the fact that SUBSCRIBE requests can be cancelled, so it is possible that not all requests will complete. So, here’s the open issue: should we address this by saying that SUBSCRIBE requests can’t be cancelled, should we try to find another solution to this problem, or should we just let things behave the way described here?

Note that forking shouldn’t usually cause problems for NOTIFY requests, since they will typically be sent to the URI indicated in the SUBSCRIBE request Contact: header. The Contact: header in a request is generally expected to point directly to the originator of the SUBSCRIBE request, thereby eliminating the possibility of forking it. Even if NOTIFY requests are forked, the consequences will be minimal. The 200 responses to NOTIFY requests are only included as confirmation, and aren’t expected to contain useful information.

6.4. Event Agents
The SIP for Presence draft (draft-rosenberg-impp-presence-00.txt) describes a mechanism by which presentities having access to registration information can accept registrations on behalf of user agents incapable of processing SUBSCRIBE requests. This is a very useful concept; however, it does not seem to be generalizable to all classes of events. Should this draft make explicit provisions for this capability, or should it remain defined in the SIP for Presence draft as behavior specific to the "presence" event package?

6.5. Is "489 Bad Event" strictly necessary?

Is a new response code really necessary, or is there an appropriate one we can re-use? No existing codes seem applicable.

7. Security Considerations

The ability to accept third-party subscriptions should be under the direct control of the user, since many types of events may be considered sensitive for the purposes of privacy. Similarly, the user agent should have the ability to selectively reject subscriptions based on the calling party (using either a white-list or black-list functionality), and/or using standard SIP authentication mechanisms.

The mere act of returning a "403 Forbidden" response code to a SUBSCRIBE request may, under certain very rare circumstances, pose privacy concerns. In these cases, the notifier may elect to return a 200 response with (possibly inaccurate) state, and never notify the subscriber of events. Note that this behavior is a rare exception, and should not be exhibited without justification.

8. References


9. Credits

Thanks to the participants in the Events BOF at the 48th IETF meeting in Pittsburgh, as well as those who gave ideas and suggestions on the SIP Events mailing list. In particular, I wish to thank Henning Schulzrinne of Columbia University for coming up with the final three-tiered event identification scheme, Sean Olson of Ericsson for miscellaneous guidance, and the authors of the "SIP Extensions for Presence" draft for their input to SUBSCRIBE and NOTIFY request semantics.

10. Author’s Address

Adam Roach
Ericsson Inc.
Mailstop L-04
851 International Pkwy.
Richardson, TX 75081
USA
Phone: +1 972 583 7594
Fax: +1 972 669 0154
E-Mail: adam.roach@ericsson.com