Internet Engineering Task Force
Internet Draft
Category: Standards Track

Adam Roach
Ericsson Inc.

February 2001
Expires August 2001
<draft-roach-sip-subscribe-notify-03.txt>

Event Notification in SIP

Status of this Memo

Abstract

This document describes an extension to the Session Initiation Protocol (SIP). 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

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.

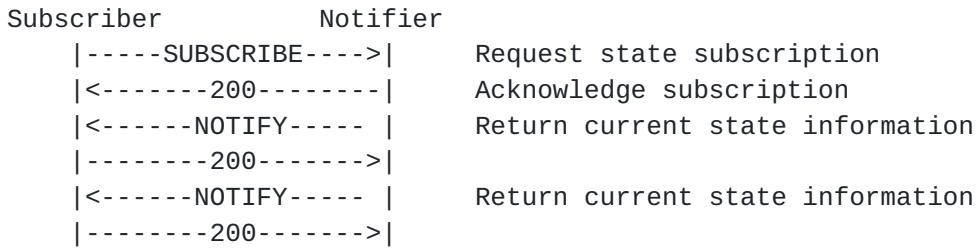
Note that the event notification mechanisms defined herein are NOT intended to be a general-purpose infrastructure for all classes of event subscription and notification. Meeting requirements for the general problem set of subscription and notification is far too complex for a single protocol. Our goal 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:



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

This document describes two new SIP methods: "SUBSCRIBE" and "NOTIFY."

This table expands on tables 4 and 5 in RFC 2543 [1] .

Header	Where	SUB	NOT
Accept	R	o	o
Accept-Encoding	R	o	o
Accept-Language	R	o	o
Allow	200	-	-
Allow	405	o	o
Authorization	R	o	o
Call-ID	gc	m	m
Contact	R	m	m
Contact	1xx	m	o
Contact	2xx	m	o
Contact	3xx	m	m
Contact	485	o	o
Content-Encoding	e	o	o
Content-Length	e	o	o
Content-Type	e	*	*
CSeq	gc	m	m
Date	g	o	o
Encryption	g	o	o
Expires	g	m	o
From	gc	m	m
Hide	R	o	o
Max-Forwards	R	o	o
Organization	g	o	o
Priority	R	o	o
Proxy-Authenticate	407	o	o
Proxy-Authorization	R	o	o
Proxy-Require	R	o	o
Require	R	o	o
Retry-After	R	-	-
Retry-After	404,480,486	o	o
Retry-After	503	o	o
Retry-After	600,603	o	o
Response-Key	R	o	o
Record-Route	R	o	o
Record-Route	2xx	o	o
Route	R	o	o
Server	r	o	o
Subject	R	o	o
Timestamp	g	o	o
To	gc(1)	m	m
Unsupported	420	o	o
User-Agent	g	o	o
Via	gc(2)	m	m
Warning	r	o	o
WWW-Authenticate	401	o	o

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.

Header field	where	proxy	ACK	BYE	CAN	INV	OPT	REG	SUB	NOT
Allow-Events	g		o	o	o	o	o	o	o	o
Event	R		-	-	-	-	-	-	m	m
Event	r		-	-	-	-	-	-	-	-

4.2.1. "Event" header

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

```
Event       =  "Event" ":" event-type
               *(( ";" parameter-name
               ["=" ( token | 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

(e.g. "Event: com.ericsson.foo").

Further note that there must be exactly one event type listed per event header. Multiple events per message are disallowed.

4.2.2. "Allow-Events" Header

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

```
Allow-Events =  "Allow-Events" ":" 1#event-type
```

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

4.3. New Response Codes

4.3.1. "202 Accepted" Response Code

The 202 response is added to the "Success" header field definition:

```
Success  = "200"  ;  OK
         |  "202"  ;  Accepted
```

"202 Accepted" has the same meaning as that defined in HTTP/1.1 [6] .

4.3.2. "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 package specified in a "Event" header field.

5. Node Behavior

Unless noted otherwise, 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, provisional responses, and message formatting as those defined in RFC 2543 [1] for BYE.

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. Note also that the presence of the "Allow-Events" header in a message is sufficient to indicate support for SUBSCRIBE and NOTIFY.

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. In general, since neither SUBSCRIBE nor NOTIFY are allowed to have protracted transactions, attempts to cancel them are expected to fail.

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, From, and Call-ID fields 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.

The relation between subscriptions and (INVITE-initiated) sessions sharing the same call leg identification information is undefined. Re-using session leg information for subscriptions is discouraged.

5.1.2. Subscription duration

SUBSCRIBE requests MUST contain an "Expires" header. This expires value indicates 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.

200-class responses to SUBSCRIBE requests 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.

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

Notifiers may also wish to cancel subscriptions to events; this is useful, for example, when the resource to which a subscription refers is no longer available. Further details on this mechanism are discussed in section 5.2.3.

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).

Subscribers MUST include exactly one "Event" header in SUBSCRIBE requests, indicating to which event or class of events they are subscribing. 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 (see [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 practical.

5.1.4. Additional SUBSCRIBE Header Values

The "Contact:" header in a SUBSCRIBE message will contain information about where resulting NOTIFY requests are to be sent. Each SUBSCRIBE request must have exactly one "Contact:" header.

SUBSCRIBE requests MAY contain an "Accept" header. This header, if present, indicates the body formats allowed in 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.

Header values not described in this document are to be interpreted as described in RFC 2543 [1] .

5.1.5. Subscriber SUBSCRIBE Behavior

5.1.5.1. Requesting a Subscription

When a subscriber wishes to subscribe to (or refresh a subscription to) an event class, he forms a SUBSCRIBE message.

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).

This SUBSCRIBE request will be confirmed with a final response. 200-class responses indicate that the subscriber will be receiving a confirmation of subscription in the form of a NOTIFY message. A 200 response can be interpreted to mean that the requested subscription has succeeded and that a NOTIFY is to be expected immediately. A 202 response indicates that there may be a sizable delay before a notification is received, pending the actual creation of the subscription. For most implementations, there will be no difference in handling these two response codes.

The "Expires" header in a 200-class response to SUBSCRIBE

indicates the actual duration for which the subscription will

remain active (unless refreshed).

Non-200 class final responses indicate that the subscription has not been created, and no subsequent NOTIFY message will be sent. All non-200 class responses (with the exception of "489," described herein) have the same meanings and handling as described in RFC 2543 [1] .

5.1.5.2. Refreshing of Subscriptions

At any time before a subscription expires, the subscriber may refresh the timer on such a subscription by re-sending a SUBSCRIBE request. The handling for such a request is the same as for the initial creation of a subscription, with the exception that these renewals will contain a the same "To," "From," and "Call-ID" as the original SUBSCRIBE request, and an incremented "CSeq" number.

If a SUBSCRIBE request to refresh a subscription fails, the original subscription is still considered valid for the duration of the most recently known "Expires" value as negotiated by SUBSCRIBE and its response, or as communicated by NOTIFY.

5.1.5.3. Unsubscribing

Unsubscribing is handled in the same way as refreshing of a subscription, with the "Expires" header set to "0." Note that a successful unsubscription will also trigger a final "NOTIFY".

5.1.5.4. Confirmation of Subscription Creation

The subscriber can expect to receive a NOTIFY message from each node which has registered a successful subscription or subscription refresh. Until the first NOTIFY message(s) arrive, the subscriber should consider the state of the subscribed resource to be in an undefined state. Extension drafts which define new event packages MUST define this "undefined state" in such a way that makes sense for their application.

Due to the potential for both out-of-order messages and forking, the subscriber MUST be prepared to receive NOTIFY messages before the SUBSCRIBE transaction has completed.

Except as noted above, processing of this NOTIFY is the same as in section 5.2.5.

5.1.6. 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 5.1.5. and 5.1.7.

5.1.7. Notifier SUBSCRIBE Behavior

5.1.7.1. SUBSCRIBE Transaction Processing

In no case should a SUBSCRIBE transaction extend for any longer than the time necessary for automated processing. In particular, notifiers MUST NOT wait for a user response before returning a final response to a SUBSCRIBE request.

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 and authorization per its local policy. See section 5.1.7.3.

If the notifier is able to immediately determine that it understands the event package, that the authenticated subscriber is authorized to subscribe, and that there are no other barriers to creating the subscriptions, it creates the subscription and returns a "200 OK" response.

If the notifier cannot immediately create the subscription (e.g. it needs to wait for user input for authorization, or is acting for another node which is not currently reachable), it will return a "202 Accepted" response. This response indicates that the request has been received and understood, but that no action has yet taken place.

The "Expires" values present in SUBSCRIBE 200-class responses behave in the same way as they do in REGISTER responses: the server MAY shorten the interval, but MUST not increase it.

200-class responses to SUBSCRIBE requests will not generally contain any useful information beyond subscription duration; their primary purpose is to serve as a reliability mechanism. State information will be communicated via a subsequent NOTIFY request from the notifier.

The other response codes defined in RFC 2543 may be used in response to SUBSCRIBE requests, as appropriate.

5.1.7.2. Confirmation of Subscription Creation/Refreshing

Upon successful creation or refreshing of a subscription,

notifiers MUST send a NOTIFY message as soon as practical to

communicate the current resource state to the subscriber. If the resource has no meaningful state at the time that the SUBSCRIBE message is processed, this NOTIFY message MAY contain an empty body. See section 5.2.3. for further details on NOTIFY message generation.

If the response to the SUBSCRIBE message was 202, this initial NOTIFY will serve as indication that the subscription has finally been processed. In the case that the subscription has not been created (e.g. the notifier was waiting for authorization and such authorization failed), the notifier SHOULD indicate to the subscriber that the subscription does has not been created by setting the "Expires" header to "0" in this initial NOTIFY response.

5.1.7.3. Authentication/Authorization of SUBSCRIBE requests

Note that privacy concerns may require that notifiers either use access lists or ask the notifier owner, on a per-subscription basis, whether a particular remote node is authorized to subscribe to a certain set of events. In general, authorization of users prior to authentication is not particularly useful.

SIP authentication mechanisms are discussed in RFC2543 [1] . 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 fails based on an access list or some other automated mechanism (i.e. it can be automatically authoritatively determined that the subscriber is not authorized to subscribe), the notifier SHOULD reply to the request with a "403 Forbidden" or "603 Decline" response, as appropriate. Depending on the situation, such a response may have security implications; see section 6.

If the notifier owner is interactively queried to determine whether a subscription is allowed, a "202 Accept" response is returned immediately, and the subsequent NOTIFY request is suppressed until the notifier owner responds.

5.1.7.4. Refreshing of Subscriptions

When a notifier receives a subscription refresh, assuming that the subscriber is still authorized, the notifier updates the expiration time for the "Contact:" address present in the SUBSCRIBE. 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.

If no refresh for a notification address is received before its expiration time, that address is removed from the list of addresses. When removing a contact, the notifier MAY send a NOTIFY message to that contact with an "Expires" value of "0" to inform it that the subscription is being removed. If all notification addresses are removed, the entire subscription is deleted.

5.2. Description of NOTIFY Behavior

Note that a NOTIFY does not cancel its corresponding subscription; in other words, a single SUBSCRIBE request may trigger several NOTIFY requests.

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.

Note that successful SUBSCRIBE requests will receive only one 200-class response; however, due to forking, the subscription may have been accepted by multiple nodes. The subscriber MUST therefore be prepared to receive NOTIFY requests with "From:" tags which differ from the "To:" tag received in the SUBSCRIBE 200-class response.

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.

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

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 present are expected to be defined by the document which describes the relevant event package.

Note that 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 SUBSCRIBE request is successfully processed or a relevant change in the subscribed state occurs, the notifier will construct and send a NOTIFY request to the subscriber(s), as specified in the "Contact" field of the SUBSCRIBE request. 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" header, 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 MUST contain a "Contact" header. This contact

header is used by the subscriber in building "Route" headers for

subsequent subscriptions (i.e. refreshes).

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 MUST remove the contact from the appropriate subscription. If removal of the contact leaves no remaining contacts, the entire subscription is removed.

NOTIFY requests MAY contain an "Expires" header which indicates the remaining duration of the subscription. The notifier MAY use this header to adjust the time remaining on the subscription; however, this mechanism MUST not be used to lengthen a subscription, only to shorten it. The notifier may inform a subscriber that a subscription has been removed by sending a NOTIFY message with an "Expires" value of "0."

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.

A notable exception to the above behavior will occur when clients are designed to receive NOTIFY messages for subscriptions set up via any means other than a SUBSCRIBE message (e.g. HTTP requests, static provisioning). Such clients will need to, under certain circumstances, process unmatched NOTIFY requests as if they had previous knowledge of the subscription.

If, for some reason, the event package designated in the "Event" header of the NOTIFY request is not supported, the subscriber should respond with a "489 Bad Event" response.

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

NOTIFY requests may contain "Expires" headers which indicate the time remaining on the subscription. If this header is present, the subscriber SHOULD take it as the authoritative duration and

adjust accordingly. If an expires value of "0" is present, the

subscriber should consider the subscription terminated.

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.

Extension drafts should describe appropriate handling for the situation in which NOTIFY requests are received from multiple notifiers. In general, such handling will involve a simple merging of the received notifications into a single, overall state.

5.3. Polling Resource State

A natural consequence of the behavior described in the preceding sections 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 active may be effected by sending an appropriate SUBSCRIBE with an "Expires" greater than 0.

Upon receipt of this SUBSCRIBE request, the notifier (or notifiers, if the SUBSCRIBE request was forked) will send a NOTIFY request containing resource state to the address in the SUBSCRIBE "Contact" field.

5.4. 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. Security Considerations

The ability to accept 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" or "603 Decline" 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 or 202 response and send a NOTIFY message with (possibly erroneous) state. Note that this behavior is a rare exception, and should not be exhibited without justification.

7. Open Issues

7.1. 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?

The only comments I've received on this issue so far favor deciding that this premise is not generally applicable, removing it as an open issue.

7.2. Event Throttling

Is the concept of throttling events (e.g. "never inform me of events more frequently than once every n seconds, no matter what") useful enough across all event types that we should define a top-level mechanism for this, or do we let extension drafts that might benefit from this sort of scheme define their own throttles?

The comments I've received on this topic are split between suggesting that event packages should define their own throttles, if appropriate, and suggesting that a general-purpose throttle mechanism would save event-package creators unnecessary re-invention of the same concepts. Supporting arguments for both

viewpoints should be taken to the sip-events mailing list, please

(see section 11.)

7.3. Resource identification for out-of-band subscriptions

In this draft, we explicitly allow subscriptions to be put into place via a mechanism other than a SUBSCRIBE request. Many people believe that sanctioning of such behavior in the base draft is important. It raises an interesting issue, however, that is probably not completely appropriate for this draft to solve. For documentation purposes, the problem is this: 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.

I get the general feeling that this problem isn't well understood by the community, and that it deserves a great deal more thought than it's receiving.

8. Changes

8.1. Changes from -02

- Clarification under "Notifier SUBSCRIBE behavior" which indicates that the first NOTIFY message (sent immediately in response to a SUBSCRIBE) may contain an empty body, if resource state doesn't make sense at that point in time.

- Text on message flow in overview section corrected

- Removed suggestion that clients attempt to unsubscribe whenever they receive a NOTIFY for an unknown event. Such behavior opens up DOS attacks, and will lead to message loops unless additional precautions are taken. The 481 response to the NOTIFY should serve the same purpose.

- Changed processing of non-200 responses to NOTIFY from "SHOULD remove contact" to "MUST remove contact" to support the above change.

- Re-added discussion of out-of-band subscription mechanisms (including open issue of resource identification).

- Added text specifying that SUBSCRIBE transactions are not to be prolonged. This is based on the consensus that non-INVITE transactions should never be prolonged; such consensus within the SIP working group was reached at the 49th IETF.

- Added "202 Accepted" response code to support the above change. The behavior of this 202 response code is a generalization of that described in the presence draft [5] .

- Updated to specify that the response to an unauthorized SUBSCRIBE request is 603 or 403.

- Level-4 subheadings added to particularly long sections to break them up into logical units. This helps make the behavior description seem somewhat less rambling. This also caused some re-ordering of these paragraphs (hopefully in a way that makes them more readable).

- Some final mopping up of old text describing "call related" and "third party" subscriptions (deprecated concepts).

- Duplicate explanation of subscription duration removed from subscriber SUBSCRIBE behavior section.

- Other text generally applicable to SUBSCRIBE (instead of just subscriber handling of SUBSCRIBE) moved to parent section.

- Updated header table to reflect mandatory usage of "Expires" header in SUBSCRIBE requests and responses

- Removed "Event" header usage in responses

- Added sentence suggesting that notifiers may notify subscribers when a subscription has timed out.

- Clarified that a failed attempt to refresh a subscription does not imply that the original subscription has been cancelled.

- Clarified that 489 is a valid response to "NOTIFY" requests.

- Minor editorial changes to clean up awkward and/or unclear grammar in several places

8.2. Changes from -01

- Multiple contacts per SUBSCRIBE message disallowed.

- Contact header now required in NOTIFY messages.

- Distinction between third party/call member events removed.

- Distinction between call-related/resource-related events removed.

- Clarified that subscribers must expect NOTIFY messages before the SUBSCRIBE transaction completes

- Added immediate NOTIFY message after successful SUBSCRIBE; this solves a myriad of issues, most having to do with forking.

- Added discussion of "undefined state" (before a NOTIFY arrives).

- Added mechanism for notifiers to shorten/cancel outstanding subscriptions.

- Removed open issue about appropriateness of new "489" response.

- Removed all discussion of out-of-band subscriptions.

- Added brief discussion of event state polling.

9. References

[1] M. Handley/H. Schulzrinne/E. Schooler/J. Rosenberg, "SIP: Session Initiation Protocol", RFC 2543, IETF; March 1999.

[2] Adam Roach, "Automatic Call Back Service in SIP", Internet Draft <draft-roach-sip-acb-00.txt>, IETF; March 2000. Work in progress.

[3] J. Rosenberg, H. Schulzrinne, "Guidelines for Authors of SIP Extensions", <draft-ietf-sip-guidelines-01.txt>, IETF; July 2000. Work in progress.

[4] S. Petrack, L. Conroy, "The PINT Service Protocol", RFC 2848, IETF; June 2000.

[5] J. Rosenberg et. al., "SIP Extensions for Presence", <draft-rosenberg-impp-presence-00.txt>, IETF; June 2000. Work in progress.

[6] R. Fielding et. al., "Hypertext Transfer Protocol -- HTTP/1.1", RFC2068, IETF, January 1997.

10. 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.

11. Feedback and Discussion

Comments regarding this draft are welcomed at the address listed below.

General-purpose discussion of asynchronous event topics, including this draft, should be taken on the sip-events mailing list (and NOT the general-purpose SIP mailing list). To subscribe, go to http://groups.yahoo.com/group/sip-events

12. 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