Adam Roach
Ericsson Inc.

October 2001
Expires April 2002
<draft-ietf-sip-events-01.fm55>

SIP-Specific Event Notification

Status of this Memo

Abstract

This document describes an extension to the Session Initiation Protocol (SIP). The purpose of this extension is to provide an 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.

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.

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.

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 SIP-specific 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. Note, however, that event packages based on this framework may define arbitrarily complex rules which govern the subscription and notification for the events or classes of events they describe.

This draft does not describe an extension which may be used directly; it must be extended by other drafts (herein referred to as “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 5.

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:

```
Subscriber          Notifier
    |-----SUBSCRIBE---->|     Request state subscription
    |<-------200--------|     Acknowledge subscription
    |<------NOTIFY----- |     Return current state information
    |--------200------->|
    |<------NOTIFY----- |     Return current state information
    |--------200------->|
```

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. Syntax

This section describes the syntax extensions required for event notification in SIP. Semantics are described in section 4.

3.1. 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	o	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	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

3.1.1. 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.

3.1.2. 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.

3.2. New Headers

This table expands on tables 4 and 5 in RFC 2543 [1], as amended by the changes described in section 3.1.

Header field	where	proxy	ACK	BYE	CAN	INV	OPT	REG	SUB	NOT
Allow-Events	g	o	o	o	o	o	o	o	o	o
Allow-Events	489	-	-	-	-	-	-	m	m	
Event	R	-	-	-	-	-	-	m	m	
Subscription-Expires	R	-	-	-	-	-	-	-	o	

3.2.1. "Event" header

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

```
Event             =  ( “Event” | “o” ) “:” event-type
                     *(( “;” parameter-name
                     [“=” ( token | quoted-string ) ] )
event-type        =  event-package *( “.” event-subpackage )
event-package     =  token-nodot
event-subpackage  =  token-nodot
token-nodot       =  1*( alphanum | "-"  | "!" | "%" | "*"
                       | "_" | "+" | "`" | "'" | "~" )
```

Event is added to the definition of the element “request-header” in the SIP message grammar.

This document does not define values for event-types. These values will be defined by individual event packages, and MUST be registered with the IANA.

There must be exactly one event type listed per event header. Multiple events per message are disallowed.

For the curious, the “o” short form is chosen to represent “occurrence.”

3.2.2. “Allow-Events” Header

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

```
Allow-Events =  ( “Allow-Events” | “u” ) “:” 1#event-type
```

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

For the curious, the “u” short form is chosen to represent “understands.”

3.2.3. “Subscription-Expires” Header

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

```
Subscription-Expires =  “Subscription-Expires” “:”
                         ( SIP-date | delta-seconds )
                          *( “;” subexp-params )

subexp-params        =   “reason” “=” reason-code
                       | generic-param

reason-code          =   “migration”
                       | “maint”
                       | “admin”
                       | “timeout”
                       | reason-extension

reason-extension      =   token
```

Subscription-Expires is added to the definition of the element “request-header” in the SIP message grammar.

3.3. New Response Codes

3.3.1. “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].

3.3.2. “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.

4. Node Behavior

4.1. General

Unless noted otherwise, SUBSCRIBE and NOTIFY requests follow the same protocol rules governing the usage of tags, Route handling, Record-Route handling, Via handling, and Contact handling as INVITE; retransmission, reliability, CSeq handling and provisional responses are the same as those defined for BYE.

For the purposes of this document, a “leg” is defined as all messages sharing the tuple of “To” (including tag), “From” (including tag), and “Call-Id.” As in INVITE-initiated legs, requests containg no “To” tag are also considered to be part of the same leg as messages which contain a “To” tag but otherwise match.

4.1.1. Route Handling

Route and Record-Route handling for SUBSCRIBE and NOTIFY legs is generally the same as for INVITE and its subsequent responses. The exact method for echoing Record-Route headers in responses and using them to form Route headers in subsequent requests is described in RFC2543 [1]. For

the purposes of the following discussion, the “Contact” header is considered part of the “Record-Route” set.

From a subscriber perspective, the “Record-Route” headers received in a SUBSCRIBE response are stored locally and placed in the “Route” headers for SUBSCRIBE refreshes. To support forking of SUBSCRIBE requests, “Record-Route” headers received in NOTIFY requests MUST be echoed back in the NOTIFY responses; if no route for the leg has been established, these “Record-Route” headers MUST be stored locally and MUST be placed in the “Route” headers for SUBSCRIBE refreshes.

From a notifier perspective, SUBSCRIBE request “Record-Route” headers are echoed back in the SUBSCRIBE response and stored locally. The locally stored copy of the “Record-Route” headers is placed in the “Route” headers when generating NOTIFY requests.

The result of the forgoing rules is that proxies wishing to remain on the signalling path for subsequent requests in the leg MUST include themselves in a “Record-Route” for all requests, not just the initial SUBSCRIBE.

4.1.2. Detecting support for SUBSCRIBE and NOTIFY

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[1].

The presence of the “Allow-Events” header in a message is sufficient to indicate support for SUBSCRIBE and NOTIFY.

The “methods” parameter for Contact may also be used to specifically announce support for SUBSCRIBE and NOTIFY messages when registering. (See reference [9] for details on the “methods” parameter).

4.1.3. CANCEL requests

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.

4.1.4. State Agents and Notifier Migration

When state agents (see section 5.4.11.) are used, it is often useful to allow migration of subscriptions between state agents and the nodes for which they are providing state aggregation (or even among various state agents). Such migration may be effected by sending a "NOTIFY" with an "Session-Expires" header of "0," and a reason parameter of "migration." This NOTIFY request is otherwise normal, and is formed as described in section 4.3.3.

Upon receipt of this NOTIFY message, the subscriber SHOULD attempt to re-subscribe (as described in the following sections). The resulting SUBSCRIBE message can then be proxied or redirected to the node to which notification responsibility is passing.

4.2. Description of SUBSCRIBE Behavior

The SUBSCRIBE method is used to request current state and state updates from a remote node.

4.2.1. 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.

Initial SUBSCRIBE requests MUST contain a "tag" parameter (as defined in RFC 2543 [1]) in the "From" header, and MUST NOT contain a "tag" parameter in the "To" header. Responses to SUBSCRIBE requests MUST contain a "tag" parameter in the "To" header.

The "tag" in the "To" header allows the subscriber to differentiate between NOTIFY requests from different clients in the case that the SUBSCRIBE request was forked. SUBSCRIBE requests for re-subscription MUST contain "tag" parameters in both the "To" and "From" headers (matching those previously established for the leg).

The relationship between subscriptions and (INVITE-initiated) sessions sharing the same call leg identification information is undefined. Re-using call leg information for subscriptions is allowed, but sharing of such information does not change the semantics of the INVITE session or the SUBSCRIBE leg; in particular, they SHOULD NOT share a route set.

Similarly, the relationship between a subscription in one direction (e.g. from node A to node B) and a subscription in the opposite direction (from B to A) with the same call leg identification information is undefined. While re-using such information is allowed, the sharing of such information does not change the semantics of either SUBSCRIBE leg. In particular, they SHOULD NOT share a route set.

4.2.2. Subscription duration

SUBSCRIBE requests SHOULD 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.

If no “Expires” header is present in a SUBSCRIBE request, the implied default is defined by the event package being used.

200-class responses to SUBSCRIBE requests also MUST contain an “Expires” header. The period of time in the response MAY be shorter or longer than specified in the request. 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 4.3.3.

4.2.3. 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 token which indicates the type of state for which a subscription is being requested. This token will be registered with the IANA and will correspond to an event package 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 return “489 Bad Event” to any SUBSCRIBE messages without an EVENT header.

If the event package to which the event token corresponds defines behavior associated with the body of its SUBSCRIBE requests, those semantics apply.

4.2.4. Additional SUBSCRIBE Header Values

The “Contact:” header in a SUBSCRIBE message will contain information about where resulting NOTIFY requests are to be sent. Note that such requests are still subject to route handling, as described in section 4.1.1. 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. Event packages 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].

4.2.5. Subscriber SUBSCRIBE Behavior

4.2.5.1. Requesting a Subscription

When a subscriber wishes to subscribe to a particular state for a resource, it 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 indi-

cates 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@domain).

This SUBSCRIBE request will be confirmed with a final response. 200-class responses indicate that the subscription has been accepted, and that a NOTIFY will be sent immediately. A 200 response indicates that the subscription has been accepted and that the user is authorized to subscribe to the requested resource. A 202 response merely indicates that the subscription has been understood, and that authorization may or may not have been granted.

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].

4.2.5.2. 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 except as described below.

Subscription renewals will contain a "To" field matching the "From" field in the first NOTIFY request for the leg containing the subscription to be refreshed. They will contain the same "From" and "Call-ID" fields as the original SUBSCRIBE request, and an incremented "CSeq" number from the original SUBSCRIBE request. Route handling is as discussed in section 4.1.1.

If a SUBSCRIBE request to refresh a subscription receives a "481" response, this indicates that the subscription has been terminated and that the subscriber did not receive notification of this fact. In this case, the subscriber should consider the subscription invalid. If the subscriber wishes to re-subscribe to the state, he does so by composing an unrelated initial SUBSCRIBE request with a freshly-generated Call-ID and a new, unique "From" tag (see section 4.2.5.1.)

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 com-

municated by NOTIFY in “Subscription-Expires,” except as described above.

4.2.5.3. 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”.

4.2.5.4. 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 arrives, the subscriber should consider the state of the subscribed resource to be in a neutral state. Event packages which define new event packages MUST define this “neutral state” in such a way that makes sense for their application (see section 5.4.7.).

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 4.3.5.

4.2.6. Proxy SUBSCRIBE Behavior

Proxies need no additional behavior beyond that described in RFC 2543 [1] to support SUBSCRIBE. If a proxy wishes to see all of the SUBSCRIBE and NOTIFY requests for a given leg, it MUST record-route all SUBSCRIBE and NOTIFY requests.

4.2.7. Notifier SUBSCRIBE Behavior

4.2.7.1. 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 4.2.7.3.

If the SUBSCRIBE request contains a tag parameter in the “To” field, but the notifier has no record of the indicated leg, the notifier has two options. If the notifier is able and willing to reconstruct subscription state, he may accept the subscription as an initial subscription. If the notifier cannot or is not willing to reconstitute such state, it should respond with a “481 Subscription does not exist.”

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 subscription, it creates the subscription and returns a “200 OK” response, unless doing so would reveal authorization policy in an undesirable fashion (see section 6.2.).

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), or wishes to mask authorization policy, it will return a “202 Accepted” response. This response indicates that the request has been received and understood, but does not necessarily imply that the subscription has been created yet.

The “Expires” values present in SUBSCRIBE 200-class responses behave in the same way as they do in REGISTER responses: the server MAY shorten or lengthen the interval.

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.

4.2.7.2. Confirmation of Subscription Creation/Refreshing

Upon successfully accepting or refreshing of a subscription, notifiers MUST send a NOTIFY message immediately 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 or neutral body. See section 4.3.3. for further details on NOTIFY message generation.

Note that a NOTIFY message is always sent immediately after any 200-class response to a SUBSCRIBE request, regardless of whether the subscription has already been authorized.

4.2.7.3. Authentication/Authorization of SUBSCRIBE requests

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, unless doing so might reveal information that should stay private; see section 6.2.

If the notifier owner is interactively queried to determine whether a subscription is allowed, a “202 Accept” response is returned immediately. Note that a NOTIFY message is still formed and sent under these circumstances, as described in the previous section.

If subscription authorization was delayed and the notifier wishes to convey that such authorization has been declined, it may do so by sending a NOTIFY message containting a “Subscription-Expires” header with a value of “0” and a reason parameter of “admin.”

4.2.7.4. Refreshing of Subscriptions

When a notifier receives a subscription refresh, assuming that the subscriber is still authorized, the notifier updates the expiration time for subscription. As with the initial subscription, the server MAY shorten or increase the amount of time until expiration. 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, the subscription is removed. When removing a subscription, the notifier MAY send a NOTIFY message with a “Subscription-Expires” value of “0” to inform it that the subscription is being removed. If such a message is sent, the “Subscription-Expires” header SHOULD contain a “reason=timeout” parameter.

4.3. Description of NOTIFY Behavior

NOTIFY messages are sent to inform subscribers of changes in state to which the subscriber has a subscription. Subscriptions are typically put in place using the SUBSCRIBE method; however, it is possible that other means have been used.

If any non-SUBSCRIBE mechanisms are defined to create subscriptions, it is the responsibility of the parties defining those mechanisms to ensure that correlation of a NOTIFY message to the corresponding subscription is possible. Designers of such mechanisms are also warned to make a distinction between sending a NOTIFY message to a subscriber who is aware of the subscription, and sending a NOTIFY message to an unsuspecting node. The latter behavior is invalid, and MUST receive a "481 Subscription does not exist" response (unless some other 400- or 500-class error code is more applicable), as described in section 4.3.5. In other words, knowlege of a subscription must exist in both the subscriber and the notifier to be valid, even if installed via a non-SUBSCRIBE mechanism.

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

4.3.1. Correlation

NOTIFY requests MUST contain the same Call-ID as the SUBSCRIBE request which ordered them; the "To" field MUST match the "From" field in the SUBSCRIBE that ordered them, and the "From" field MUST match the "To" field that was sent in the 200-class response to the SUBSCRIBE. This is the same set of criteria that define a call leg.

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

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.

If multiple NOTIFY messages are received in response to a single SUBSCRIBE message, they represent different destinations to which the SUBSCRIBE request was forked. Unless the event package specifies otherwise, the subscriber may either accept all such notifications as representing different legs (which are then refreshed separately), or send a 481 response to any NOTIFYs on legs that it does not want to keep alive.

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.

4.3.2. 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 4.2.3.).

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 token which identifies the event or class of events for which a notification is being generated.

If the event package 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.

4.3.3. Notifier NOTIFY Behavior

When a SUBSCRIBE request is successfully processed or a relevant change in the subscribed state occurs, the notifier will immediately construct and send a NOTIFY request to the subscriber(s), as specified in the "Contact" field of the SUBSCRIBE request. This message is subject to standard Route/Record-Route handling, as described in section 4.1.1.

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]).

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 NOTIFY request fails (as defined above), and the subscription was installed using a soft-state mechanism (such as SIP signalling), the notifier MUST remove the corresponding subscription.

If a NOTIFY request receives a 481 response, the notifier MUST remove the corresponding subscription even if such subscription was installed by non-SUBSCRIBE means (such as an administrative interface).

NOTIFY requests SHOULD contain an “Subscription-Expires” header which indicates the remaining duration of the subscription (such a header is useful in case the SUBSCRIBE request forks, since the response to a forked subscribe -- which contains the “Expire” header that specifies the agreed-upon expiration time -- may not be received by the subscriber). The notifier SHOULD 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 “Subscription-Expires” value of “0.”

If the duration of a subscription has been shortened or terminated by the “Subscription-Expires” header as compared to the most recent 200-class SUBSCRIBE response sent, that header SHOULD include a “reason” parameter indicating the reason that such action was taken. Currently, three such values are defined: “migration” indicates that the node acting as notifier is transferring responsibility for maintaing such state information to another node; this only makes sense when subscriptions are terminated, not when they are shortened. “Maint” indicates that the subscription is being truncated or terminated due to server maintainance, and “admin” indicates that the subscription has been removed or shortened administratively (e.g. by a change in ACL policy). Finally, if the notifier elects to send a NOTIFY upon timeout of the subscription, they SHOULD include a “Subscription-Expires” header with a value of “0” and a reason parameter of “timeout.”

4.3.4. Proxy NOTIFY Behavior

Proxies need no additional behavior beyond that described in RFC 2543 [1] to support NOTIFY. If a proxy wishes to see all of the SUBSCRIBE and NOTIFY requests for a given leg, it MUST record-route all SUBSCRIBE and NOTIFY requests.

4.3.5. 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 MUST return a "481 Subscription does not exist" response unless another 400- or 500-class response is more appropriate.

If, for some reason, the event package designated in the "Event" header of the NOTIFY request is not supported, the subscriber will 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 SHOULD contain "Subscription-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.

When the subscription is terminated or shortened using the "Subscription-Expires" mechanism, there SHOULD be a reason parameter present. If it is present and the subscriber is still interested in receiving updates to the state information, the subscriber SHOULD attempt re-subscribe upon expiration if it is set to "migration," "timeout," is not present, or is set to an unknown value. If it is set to either "maint" or "admin," the user SHOULD NOT attempt re-subscription.

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.

4.4. 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. Note that normal Route and Record-Route handle still applies; see section 4.1.1.

4.5. Allow-Events header usage

The "Allow-Events" header, if present, includes a list of tokens which indicates the event packages supported by the client (if sent in a request) or server (if sent in a response). In other words, a node sending an "Allow-Events" header is advertising that it can process SUBSCRIBE requests and generate NOTIFY requests for all of the event packages listed in that header.

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.

Note that "Allow-Events" headers MUST NOT be inserted by proxies.

5. Event Packages

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

5.1. Appropriateness of Usage

When designing an event package 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 event packages 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.

5.2. Sub-packages

Normal event packages define a set of state applied to a specific type of resource, such as user presence, call state, and messaging mailbox state.

Sub-packages are a special type of package which define a set of state applied to other packages, such as statistics, access policy, and subscriber lists. Sub-packages may even be applied to other sub-packages.

To extend the object-oriented analogy made earlier, sub-packages can be thought of as templatized C++ packages which must be applied to other packages to be useful.

The name of a sub-package as applied to a package is formed by appending a period followed by the sub-package name to the end of the package. For example, if a subpackage called "watcherinfo" were being applied to a package called "presence," the event token used in "Event" and "Allow-Events" would be "presence.watcherinfo".

Sub-packages must be defined so that they can be applied to any arbitrary package. In other words, sub-packages cannot be specifically tied to one or a few "parent" packages in such a way that they will not work with other packages.

5.3. Amount of State to be Conveyed

When designing event packages, 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.

There are two possible solutions to this problem that event packages may choose to implement.

For packages which typically convey state information that is reasonably small (on the order of 1 kb or so), it is suggested that event packages are designed so as to send complete state information when an event occurs. In the circumstances that state may not be sufficient for a particular class of events, the event packages 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”.)

In the case that the state information to be conveyed is large, the event package may choose to detail a scheme by which NOTIFY messages contain state deltas instead of complete state. Such a scheme would work as follows: any NOTIFY sent in immediate response to a SUBSCRIBE contains full state information. NOTIFY messages sent because of a state change will contain only the state information that has changed; the subscriber will then merge this information into its current knowledge about the state of the resource. If a NOTIFY arrives that has a CSeq number that is incremented by more than one, the subscriber knows that a state delta may have been missed; it ignores the NOTIFY message containing a state delta and re-sends a SUBSCRIBE to force a NOTIFY containing complete state snapshot.

5.4. Event Package Responsibilities

Event packages are not required to re-iterate any of the behavior described in this document, although they may choose to do so for clarity or emphasis. In general, though, such packages are expected to describe only the behavior that extends or modifies the behavior described in this document.

Note that any behavior designated with “SHOULD” or “MUST” in this document is not allowed to be changed by extension documents; however, such documents may elect to strengthen “SHOULD” requirements to “MUST” strength if required by their application.

In addition to the normal sections expected by “Instructions to RFC Authors”[7] and “Guidelines for Authors of SIP Extensions”[3], authors of event packages MUST address each of the issues detailed in the following subsections, whenever applicable.

5.4.1. Event Package Name

This mandatory section of an event package defines the token name to be used to designate the event package. It MUST include the information which appears in the IANA registration of the token. For information on registering such types, see section 7.

5.4.2. Event Package Parameters

If parameters are to be used on the "Event" header to modify the behavior of the event package, the syntax and semantics of such headers must be clearly defined.

5.4.3. SUBSCRIBE Bodies

It is expected that most, but not all, event packages 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 event packages are strongly encouraged to re-use existing MIME types for message bodies where practical.

This mandatory section of an event package defines what type or types of event bodies are expected in SUBSCRIBE requests (or specify that no event bodies are expected). It should point to detailed definitions of syntax and semantics for all referenced body types.

5.4.4. Subscription Duration

It is recommended that event packages give a suggested range of times considered reasonable for the duration of a subscription. Such packages MUST also define a default "Expires" value to be used if none is specified.

5.4.5. NOTIFY Bodies

The NOTIFY body is used to report state on the resource being monitored. Each package must define a what type or types of event bodies are expected in NOTIFY requests. Such packages must specify or cite detailed specifications for the syntax and semantics associated with such event body.

Event packages also need to define which MIME type is to be assumed if none are specified in the "Accept" header of the SUBSCRIBE request.

5.4.6. Notifier processing of SUBSCRIBE requests

This section describes the processing to be performed by the notifier upon receipt of a SUBSCRIBE request. Such a section is required.

Information in this section includes details of how to authenticate subscribers and authorization issues for the package. Such authorization issues may include, for example, whether all SUBSCRIBE requests for this package are answered with 202 responses (see section 6.2.).

5.4.7. Notifier generation of NOTIFY requests

This section of an event package describes the process by which the notifier generates and sends a NOTIFY request. This includes detailed information about what events cause a NOTIFY to be sent, how to compute the state information in the NOTIFY, how to generate “neutral” or “fake” state information to hide authorization delays and decisions from users, and whether state information is complete or deltas for notifications (see section 5.3.)

It may optionally describe the behavior used to processes the subsequent response. Such a section is required.

5.4.8. Subscriber processing of NOTIFY requests

This section of an event package describes the process followed by the subscriber upon receipt of a NOTIFY request, including any logic required to form a coherent resource state (if applicable).

5.4.9. Handling of forked requests

Each event package should specify whether forked SUBSCRIBE requests are allowed to install multiple subscriptions. If such behavior is not allowed, any NOTIFY messages not matching the 200-class response to the initial SUBSCRIBE message are responded to with a 481.

In the case that multiple subscriptions are allowed, the event package must specify whether merging of the notifications to form a single state is required, and how such merging is to be performed. Note that it is possible that some event packages may be defined in such a way that each leg is tied to a mutually exclusive state which is unaffected by the other legs; this must be clearly stated if it is the case.

5.4.10. Rate of notifications

Each event package is expected to define a requirement (RECOMMENDED, SHOULD or MUST strength) which defines an absolute maximum on the rate at which notifications are allowed to be generated by a single notifier.

Such packages may further define a throttle mechanism which allows subscribers to further limit the rate of notification.

5.4.11. State Agents

Designers of event packages should consider whether their package can benefit from network aggregation points (“State Agents”) and/or nodes which act on behalf of other nodes. (For example, nodes which provide state information about a resource when such a resource is unable or unwilling to pro-

vide such state information itself). An example of such an application is a node which tracks the presence and availability of a user in the network.

If state agents are to be used by the package, the package must specify how such state agents aggregate information and how they provide authentication and authorization.

5.4.12. Examples

Event packages should include several demonstrative message flow diagrams paired with several typical, syntactically correct and complete messages.

It is recommended that documents describing event packages clearly indicate that such examples are informative and not normative, with instructions that implementors refer to the main text of the draft for exact protocol details.

6. Security Considerations

6.1. Access Control

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 notifier should have the ability to selectively reject subscriptions based on the calling party (based on access control lists), using standard SIP authentication mechanisms. The methods for creation and distribution of such access control lists is outside the scope of this draft.

6.2. Release of Sensitive Policy Information

The mere act of returning a 200 or certain 4xx and 6xx responses to SUBSCRIBE requests may, under certain circumstances, create privacy concerns by revealing sensitive policy information. In these cases, the notifier should always return a 202 response. While the subsequent NOTIFY message may not convey true state, it MUST appear to contain a potentially correct piece of data from the point of view of the subscriber, indistinguishable from a valid response. Information about whether a user is authorized to subscribe to the requested state is never conveyed back to the original user under these circumstances.

6.3. Denial-of-Service attacks

The current model (one SUBSCRIBE request triggers a SUBSCRIBE response and one or more NOTIFY requests) is a classic setup for an amplifier node to be used in a smurf attack.

Also, the creation of state upon receipt of a SUBSCRIBE request can be used by attackers to consume resources on a victim's machine, rendering it unusable.

To reduce the chances of such an attack, implementations of notifiers SHOULD require authentication. Authentication issues are discussed in RFC2543 [1].

7. IANA Considerations

(This section is not applicable until this document is published as an RFC.)

This document defines an event-type namespace which requires a central coordinating body. The body chosen for this coordination is the Internet Assigned Numbers Authority (IANA).

There are two different types of event-types: normal event packages, and event sub-packages; see section 5.2. To avoid confusion, subpackage names and package names share the same namespace; in other words, a sub-package MUST NOT share a name with a package.

Following the policies outlined in "Guidelines for Writing an IANA Considerations Section in RFCs"[8], normal event package identification tokens are allocated as First Come First Served, and event sub-package identification tokens are allocated on a IETF Consensus basis.

Registrations with the IANA MUST include the token being registered and whether the token is a package or a subpackage. Further, packages MUST include contact information for the party responsible for the registration and/or a published document which describes the event package. Sub-package token registrations MUST include a pointer to the published RFC which defines the sub-package.

Registered tokens to designate packages and sub-packages MUST NOT contain the character ".", which is used to separate sub-packages from packages.

7.1. Registration Template

As this document specifies no package or sub-package names, the initial IANA registration for event types will be empty. The remainder of the text in this section gives an example of the type of information to be maintained by the IANA; it also demonstrates all five possible permutations of package type, contact, and reference.

The table below lists the event packages and sub-packages defined in “SIP-Specific Event Notification” [RFC xxxx]. Each name is designated as a package or a subpackage under “Type.”

Package Name	Type	Contact	Reference
example1	package	[Roach]	
example2	package	[Roach]	[RFC xxxx]
example3	package		[RFC xxxx]
example4	sub-package	[Roach]	[RFC xxxx]
example5	sub-package		[RFC xxxx]

PEOPLE

[Roach] Adam Roach <adam.roach@ericsson.com>

REFERENCES

[RFC xxxx] A. Roach “SIP-Specific Event Notification”, RFC XXXX, August 2002.

8. Changes

8.1. Changes from draft-ietf-...-00

- Removed “Open Issues” section. We have no known open issues remaining.

- Fixed confusing typo in section describing correlation of SUBSCRIBE requests

- Added explanitory text to clarify tag handling when generating re-subscriptions

- Expanded general handling section to include specific discussion of Route/Record-Route handling.

- Included use of “methods” parameter on Contact as a means for detecting support for SUBSCRIBE and NOTIFY.

- Clarified definition of term “leg”

- Added syntax for “Subscription-Expires” header.

- Changed NOTIFY messages to refer to “Subscription-Expires” everywhere (instead of “Expires.”)

- Added information about generation and handling of 481 responses to SUBSCRIBE requests

- Changed having Expires header in SUBSCRIBE from MUST to SHOULD; this aligns more closely with REGISTER behavior

- Removed experimental/private event package names, per list consensus

- Cleaned up some legacy text left over from very early drafts that allowed multiple contacts per subscription

- Strengthened language requiring the removal of subscriptions if a NOTIFY request fails with a 481. Clarified that such removal is required for all subscriptions, including administrative ones.

- Removed description of delaying NOTIFY requests until authorization is granted. Such behavior was inconsistent with other parts of this document.

- Moved description of event packages to later in document, to reduce the number of forward references.

- Minor editorial and nits changes

8.2. Changes from draft-roach-...-03

- Added DOS attacks section to open issues.

- Added discussion of forking to open issues

- Changed response to PINT request for notifiers who don’t support PINT from 400 to 489.

- Added sentence to security section to call attention to potential privacy issues of delayed NOTIFY responses.

- Added clarification: access control list handling is out of scope.

- (Hopefully) Final resolution on out-of-band subscriptions: mentioned in section 4.3. Removed from open issues.

- Made “Contact” header optional for SUBSCRIBE 1xx responses.

- Added description clarifying tag handling (section 4.2.1.)

- Removed event throttling from open issues.

- Editorial cleanup to remove term “extension draft” and similar; “event package” is now (hopefully) used consistently throughout the document.

- Remove discussion of event agents from open issues. This is covered in the event packages section now.

- Added discussion of forking to open issues.

- Added discussion of sub-packages

- Added clarification that, upon receiving a “NOTIFY” with an expires of “0”, the subscriber can re-subscribe.

This allows trivial migration of subscriptions between nodes.

- Added preliminary IANA Considerations section

- Changed syntax for experimental event tokens to avoid possibly ambiguity between experimental tokens and sub-packages.

- Slight adjustment to “Event” syntax to accommodate sub-packages.

- Added section describing the information which is to be included in documents describing event packages.

- Made 481 responses mandatory for unexpected notifications (allowing notifiers to remove subscriptions in error cases)

- Several minor non-semantic editorial changes.

8.3. Changes from draft-roach-...-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.

- 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

8.4. Changes from draft-roach-...-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.

[7] J. Postel, J. Reynolds, “Instructions to RFC Authors”, RFC2223, IETF, October 1997.

[8] T. Narten, H. Alvestrand, “Guidelines for Writing an IANA Considerations Section in RFCs”, BCP 26, IETF, October 1998.

[9] H. Schulzrinne, J. Rosenberg “SIP Caller Preferences and Callee Capabilities”, <draft-ietf-sip-callerprefs-04.txt>, IETF; June 14, 2001. Work in progress.

10. Acknowledgements

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, Jonathan Rosenberg for a thorough scrubbing of the -00 draft, and the authors of the “SIP Extensions for Presence” draft for their input to SUBSCRIBE and NOTIFY request semantics.

11. Author’s Address

Adam Roach
Ericsson Inc.
Mailstop L-04
740 E. Campbell Rd.

Richardson, TX 75081
USA
Phone: +1 972 583 7594
Fax: +1 972 669 0154
E-Mail: adam.roach@ericsson.com