Network Working Group   A. B. Roach
Request for Comments: 3265   dynamicsoft
Updates: 2543   June 2002
Category: Standards Track

        Session Initiation Protocol (SIP)-Specific Event Notification

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.

Table of Contents

1. Introduction

   The ability to request asynchronous notification of events proves
   useful in many types of SIP 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 PSTN and Internet
   Internetworking (PINT) [2] status (based on call state events).

   The methods described in this document provide 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 elaborate rules which govern the
   subscription and notification for the events or classes of events
   they describe.

   This document does not describe an extension which may be used
   directly; it must be extended by other documents (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
instantiable class by further extensions. Guidelines for creating
these extensions are described in section 4.

1.1. 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------->|
```

Subscriptions are expired and must be refreshed by subsequent
SUBSCRIBE messages.

1.2. Documentation Conventions

There are several paragraphs throughout this document which provide
motivational or clarifying text. Such passages are non-normative,
and are provided only to assist with reader comprehension. These
passages are set off from the remainder of the text by being indented
thus:

   This is an example of non-normative explanatory text. It does not
   form part of the specification, and is used only for
   clarification.

Numbers in square brackets (e.g., [1]) denote a reference to one of
the entries in the reference sections; see sections 8 and 9.

The all-capital terms "MUST", "SHOULD", "MAY", "SHOULD NOT", "MUST
NOT", and "RECOMMENDED" are used as defined in RFC 2119 [5].

The use of quotation marks next to periods and commas follows the
convention used by the American Mathematical Society; although
contrary to traditional American English convention, this usage lends
clarity to certain passages.

2. Definitions

   Event Package: An event package is an additional specification which
      defines a set of state information to be reported by a notifier to
      a subscriber. Event packages also define further syntax and
      semantics based on the framework defined by this document required
      to convey such state information.

   Event Template-Package: An event template-package is a special kind
      of event package which defines a set of states which may be
      applied to all possible event packages, including itself.

   Notification: Notification is the act of a notifier sending a NOTIFY
      message to a subscriber to inform the subscriber of the state of a
      resource.

   Notifier: A notifier is a user agent which generates NOTIFY requests
      for the purpose of notifying subscribers of the state of a
      resource. Notifiers typically also accept SUBSCRIBE requests to
      create subscriptions.

   State Agent: A state agent is a notifier which publishes state
      information on behalf of a resource; in order to do so, it may
      need to gather such state information from multiple sources.
      State agents always have complete state information for the
      resource for which they are creating notifications.

   Subscriber: A subscriber is a user agent which receives NOTIFY
      requests from notifiers; these NOTIFY requests contain information
      about the state of a resource in which the subscriber is
      interested. Subscribers typically also generate SUBSCRIBE
      requests and send them to notifiers to create subscriptions.

   Subscription: A subscription is a set of application state associated
      with a dialog. This application state includes a pointer to the
      associated dialog, the event package name, and possibly an
      identification token. Event packages will define additional
      subscription state information. By definition, subscriptions
      exist in both a subscriber and a notifier.

   Subscription Migration: Subscription migration is the act of moving a
      subscription from one notifier to another notifier.

3. Node Behavior

3.1. Description of SUBSCRIBE Behavior

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

3.1.1. Subscription Duration

   SUBSCRIBE requests SHOULD contain an "Expires" header (defined in SIP
   [1]). This expires value indicates the duration of the subscription.
   In order to keep subscriptions effective beyond the duration
   communicated in the "Expires" header, subscribers need to refresh
   subscriptions on a periodic basis using a new SUBSCRIBE message on
   the same dialog as defined in SIP [1].

   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
   but MUST NOT be longer than specified in the request. The period of
   time in the response is the one which defines the duration of the
   subscription.

   An "expires" parameter on the "Contact" header has no semantics for
   SUBSCRIBE and is explicitly not equivalent to an "Expires" header in
   a SUBSCRIBE request or response.

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

      In addition to being a request to unsubscribe, a SUBSCRIBE message
      with "Expires" of 0 also causes a fetch of state; see section
      3.3.6.

   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 3.2.2.

3.1.2. 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 per
the request routing procedures outlined in SIP [1]. 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@dynamicsoft.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 MAY also contain an "id" parameter.
This "id" parameter, if present, contains an opaque token which
identifies the specific subscription within a dialog. An "id"
parameter is only valid within the scope of a single dialog.

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

Event packages may also define parameters for the Event header; if
they do so, they must define the semantics for such parameters.

3.1.3. Additional SUBSCRIBE Header Values

Because SUBSCRIBE requests create a dialog as defined in SIP [1],
they 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 SIP [1].

3.1.4. Subscriber SUBSCRIBE Behavior

3.1.4.1. Requesting a Subscription

SUBSCRIBE is a dialog-creating method, as described in SIP [1].

When a subscriber wishes to subscribe to a particular state for a
resource, it forms a SUBSCRIBE message. If the initial SUBSCRIBE

represents a request outside of a dialog (as it typically will), its
construction follows the procedures outlined in SIP [1] for UAC
request generation outside of a dialog.

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 no subscription or dialog
has 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 SIP [1].

A SUBSCRIBE request MAY include an "id" parameter in its "Event"
header to allow differentiation between multiple subscriptions in the
same dialog.

3.1.4.2. Refreshing of Subscriptions

At any time before a subscription expires, the subscriber may refresh
the timer on such a subscription by sending another SUBSCRIBE request
on the same dialog as the existing subscription, and with the same
"Event" header "id" parameter (if one was present in the initial
subscription). The handling for such a request is the same as for
the initial creation of a subscription except as described below.

    If the initial SUBSCRIBE message contained an "id" parameter on
    the "Event" header, then refreshes of the subscription must also
    contain an identical "id" parameter; they will otherwise be
    considered new subscriptions in an existing dialog.

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
3.1.4.1.)

   If a SUBSCRIBE request to refresh a subscription fails with a non-481
   response, 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 in the
   "Subscription-State" header "expires" parameter.

      Note that many such errors indicate that there may be a problem
      with the network or the notifier such that no further NOTIFY
      messages will be received.

3.1.4.3. 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 message.

3.1.4.4. Confirmation of Subscription Creation

   The subscriber can expect to receive a NOTIFY message from each node
   which has processed 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. Documents which define new event packages MUST define
   this "neutral state" in such a way that makes sense for their
   application (see section 4.4.7.).

   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 3.2.4.

3.1.5. Proxy SUBSCRIBE Behavior

   Proxies need no additional behavior beyond that described in SIP [1]
   to support SUBSCRIBE. If a proxy wishes to see all of the SUBSCRIBE
   and NOTIFY requests for a given dialog, it MUST record-route the
   initial SUBSCRIBE and any dialog-establishing NOTIFY requests. Such
   proxies SHOULD also record-route all other SUBSCRIBE and NOTIFY
   requests.

      Note that subscribers and notifiers may elect to use S/MIME
      encryption of SUBSCRIBE and NOTIFY requests; consequently, proxies
      cannot rely on being able to access any information that is not
      explicitly required to be proxy-readable by SIP [1].

3.1.6. Notifier SUBSCRIBE Behavior

3.1.6.1. Initial 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.

      This requirement is imposed primarily to prevent the non-INVITE
      transaction timeout timer F (see [1]) from firing during the
      SUBSCRIBE transaction, since interaction with a user would often
      exceed 64*T1 seconds.

   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 3.1.6.3.

   The notifier MAY also check that the duration in the "Expires" header
   is not too small. If and only if the expiration interval is greater
   than zero AND smaller than one hour AND less than a notifier-
   configured minimum, the notifier MAY return a "423 Interval too
   small" error which contains a "Min-Expires" header field. The "Min-
   Expires" header field is described in SIP [1].

   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 a dialog (if
   necessary), and returns a "200 OK" response (unless doing so would
   reveal authorization policy in an undesirable fashion; see section
   5.2.).

   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
   authorized yet.

   When a subscription is created in the notifier, it stores the event
   package name and the "Event" header "id" parameter (if present) as
   part of the subscription information.

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 lengthen it.

   If the duration specified in a SUBSCRIBE message is unacceptably
   short, the notifier may be able to send a 423 response, as
   described earlier in this section.

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 SIP [1] may be used in response
to SUBSCRIBE requests, as appropriate.

3.1.6.2. Confirmation of Subscription Creation/Refreshing

Upon successfully accepting or refreshing a subscription, notifiers
MUST send a NOTIFY message immediately to communicate the current
resource state to the subscriber. This NOTIFY message is sent on the
same dialog as created by the SUBSCRIBE response. 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 3.2.2. 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.

3.1.6.3. Authentication/Authorization of SUBSCRIBE requests

Privacy concerns may require that notifiers apply policy to determine
whether a particular subscriber is authorized to subscribe to a
certain set of events. Such policy may be defined by mechanisms such
as access control lists or real-time interaction with a user. In
general, authorization of subscribers prior to authentication is not
particularly useful.

SIP authentication mechanisms are discussed in SIP [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.

      Of course, when acting as a proxy, a node will perform normal
      proxy authentication (using 407). The foregoing explanation is a
      reminder that notifiers are always UAs, and as such perform UA
      authentication.

   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 5.2.

   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 containing a "Subscription-State" header
   with a value of "terminated" and a reason parameter of "rejected".

3.1.6.4. 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 subscription. As with the initial subscription, the
   server MAY shorten 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 the duration specified in a SUBSCRIBE
   message is unacceptably short, the notifier SHOULD respond with a
   "423 Subscription Too Brief" message.

   If no refresh for a notification address is received before its
   expiration time, the subscription is removed. When removing a
   subscription, the notifier SHOULD send a NOTIFY message with a
   "Subscription-State" value of "terminated" to inform it that the
   subscription is being removed. If such a message is sent, the
   "Subscription-State" header SHOULD contain a "reason=timeout"
   parameter.

      The sending of a NOTIFY when a subscription expires allows the
      corresponding dialog to be terminated, if appropriate.

3.2. 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 3.2.4. In other words, knowledge 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 terminate its corresponding subscription; in other
   words, a single SUBSCRIBE request may trigger several NOTIFY
   requests.

3.2.1. 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
   3.1.2.).

   As in SUBSCRIBE requests, NOTIFY "Event" headers will contain a
   single event package name for which a notification is being
   generated. The package name in the "Event" header MUST match the
   "Event" header in the corresponding SUBSCRIBE message. If an "id"
   parameter was present in the SUBSCRIBE message, that "id" parameter
   MUST also be present in the corresponding NOTIFY messages.

   Event packages may define semantics associated with the body of their
   NOTIFY requests; if they do so, those semantics apply. NOTIFY bodies
   are 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. This body will contain either the
   state of the subscribed resource or a pointer to such state in the
   form of a URI (see section 4.4.13).

3.2.2. Notifier NOTIFY Behavior

   When a SUBSCRIBE request is answered with a 200-class response, the
   notifier MUST immediately construct and send a NOTIFY request to the
   subscriber. When a change in the subscribed state occurs, the
   notifier SHOULD immediately construct and send a NOTIFY request,
   subject to authorization, local policy, and throttling
   considerations.

   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) due to a timeout
   condition, and the subscription was installed using a soft-state
   mechanism (such as SUBSCRIBE), the notifier SHOULD remove the
   subscription.

      This behavior prevents unnecessary transmission of state
      information for subscribers who have crashed or disappeared from
      the network. Because such transmissions will be sent multiple
      times, per the retransmission algorithm defined in SIP [1]
      (instead of the typical single transmission for functioning
      clients), continuing to service them when no client is available
      to acknowledge them could place undue strain on a network. Upon
      client restart or reestablishment of a network connection, it is
      expected that clients will send SUBSCRIBE messages to refresh
      potentially stale state information; such messages will re-install
      subscriptions in all relevant nodes.

   If the NOTIFY request fails (as defined above) due to an error
   response, and the subscription was installed using a soft-state
   mechanism, the notifier MUST remove the corresponding subscription.

      A notify error response would generally indicate that something
      has gone wrong with the subscriber or with some proxy on the way
      to the subscriber. If the subscriber is in error, it makes the
      most sense to allow the subscriber to rectify the situation (by
      re-subscribing) once the error condition has been handled. If a
      proxy is in error, the periodic SUBSCRIBE refreshes will re-
      install subscription state once the network problem has been
      resolved.

   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).

If the above behavior were not required, subscribers receiving a
notify for an unknown subscription would need to send an error
status code in response to the NOTIFY and also send a SUBSCRIBE
request to remove the subscription. Since this behavior would
make subscribers available for use as amplifiers in denial of
service attacks, we have instead elected to give the 481 response
special meaning: it is used to indicate that a subscription must
be cancelled under all circumstances.

NOTIFY requests MUST contain a "Subscription-State" header with a
value of "active", "pending", or "terminated". The "active" value
indicates that the subscription has been accepted and has been
authorized (in most cases; see section 5.2.). The "pending" value
indicates that the subscription has been received, but that policy
information is insufficient to accept or deny the subscription at
this time. The "terminated" value indicates that the subscription is
not active.

If the value of the "Subscription-State" header is "active" or
"pending", the notifier SHOULD also include in the "Subscription-
State" header an "expires" parameter which indicates the time
remaining on the subscription. The notifier MAY use this mechanism
to shorten a subscription; however, this mechanism MUST NOT be used
to lengthen a subscription.

Including expiration information for active and pending
subscriptions is useful in case the SUBSCRIBE request forks, since
the response to a forked SUBSCRIBE may not be received by the
subscriber. Note well that this "expires" value is a parameter on
the "Subscription-State" header, NOT an "Expires" header.

If the value of the "Subscription-State" header is "terminated", the
notifier SHOULD also include a "reason" parameter. The notifier MAY
also include a "retry-after" parameter, where appropriate. For
details on the value and semantics of the "reason" and "retry-after"
parameters, see section 3.2.4.

3.2.3. Proxy NOTIFY Behavior

Proxies need no additional behavior beyond that described in SIP [1]
to support NOTIFY. If a proxy wishes to see all of the SUBSCRIBE and
NOTIFY requests for a given dialog, it MUST record-route the initial
SUBSCRIBE and any dialog-establishing NOTIFY requests. Such proxies
SHOULD also record-route all other SUBSCRIBE and NOTIFY requests.

Note that subscribers and notifiers may elect to use S/MIME
encryption of SUBSCRIBE and NOTIFY requests; consequently, proxies
cannot rely on being able to access any information that is not
explicitly required to be proxy-readable by SIP [1].

3.2.4. 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. The rules
for matching NOTIFY requests with subscriptions that create a new
dialog are described in section 3.3.4. Notifications for
subscriptions which were created inside an existing dialog match if
they are in the same dialog and the "Event" headers match (as
described in section 7.2.1.)

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 SHOULD be
authenticated, using any defined SIP authentication mechanism.

NOTIFY requests MUST contain "Subscription-State" headers which
indicate the status of the subscription.

If the "Subscription-State" header value is "active", it means that
the subscription has been accepted and (in general) has been
authorized. If the header also contains an "expires" parameter, the
subscriber SHOULD take it as the authoritative subscription duration
and adjust accordingly. The "retry-after" and "reason" parameters
have no semantics for "active".

If the "Subscription-State" value is "pending", the subscription has
been received by the notifier, but there is insufficient policy
information to grant or deny the subscription yet. If the header
also contains an "expires" parameter, the subscriber SHOULD take it
as the authoritative subscription duration and adjust accordingly.
No further action is necessary on the part of the subscriber. The
"retry-after" and "reason" parameters have no semantics for
"pending".

If the "Subscription-State" value is "terminated", the subscriber
should consider the subscription terminated. The "expires" parameter
has no semantics for "terminated". If a reason code is present, the
client should behave as described below. If no reason code or an
unknown reason code is present, the client MAY attempt to re-

subscribe at any time (unless a "retry-after" parameter is present,
in which case the client SHOULD NOT attempt re-subscription until
after the number of seconds specified by the "retry-after"
parameter). The defined reason codes are:

deactivated: The subscription has been terminated, but the subscriber
   SHOULD retry immediately with a new subscription. One primary use
   of such a status code is to allow migration of subscriptions
   between nodes. The "retry-after" parameter has no semantics for
   "deactivated".

probation: The subscription has been terminated, but the client
   SHOULD retry at some later time. If a "retry-after" parameter is
   also present, the client SHOULD wait at least the number of
   seconds specified by that parameter before attempting to re-
   subscribe.

rejected: The subscription has been terminated due to change in
   authorization policy. Clients SHOULD NOT attempt to re-subscribe.
   The "retry-after" parameter has no semantics for "rejected".

timeout: The subscription has been terminated because it was not
   refreshed before it expired. Clients MAY re-subscribe
   immediately. The "retry-after" parameter has no semantics for
   "timeout".

giveup: The subscription has been terminated because the notifier
   could not obtain authorization in a timely fashion. If a "retry-
   after" parameter is also present, the client SHOULD wait at least
   the number of seconds specified by that parameter before
   attempting to re-subscribe; otherwise, the client MAY retry
   immediately, but will likely get put back into pending state.

noresource: The subscription has been terminated because the resource
   state which was being monitored no longer exists. Clients SHOULD
   NOT attempt to re-subscribe. The "retry-after" parameter has no
   semantics for "noresource".

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 SIP [1] may also be returned, as
appropriate. In no case should a NOTIFY transaction extend for any
longer than the time necessary for automated processing. In
particular, subscribers MUST NOT wait for a user response before
returning a final response to a NOTIFY request.

3.3. General

3.3.1. 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 SIP [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 [8] for details on the "methods"
      parameter).

3.3.2. CANCEL requests

   No semantics are associated with cancelling SUBSCRIBE or NOTIFY.

3.3.3. Forking

   In accordance with the rules for proxying non-INVITE requests as
   defined in SIP [1], 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 different dialogs in
   response to a single SUBSCRIBE message, each dialog represents a
   different destination to which the SUBSCRIBE request was forked. For
   information on subscriber handling in such situations, see section
   4.4.9.

3.3.4. Dialog creation and termination

   If an initial SUBSCRIBE request is not sent on a pre-existing dialog,
   the subscriber will wait for a response to the SUBSCRIBE request or a
   matching NOTIFY.

   Responses are matched to such SUBSCRIBE requests if they contain the
   same the same "Call-ID", the same "From" header "tag", and the same
   "CSeq". Rules for the comparison of these headers are described in
   SIP [1]. If a 200-class response matches such a SUBSCRIBE request,
   it creates a new subscription and a new dialog (unless they have
   already been created by a matching NOTIFY request; see below).

NOTIFY requests are matched to such SUBSCRIBE requests if they
contain the same "Call-ID", a "To" header "tag" parameter which
matches the "From" header "tag" parameter of the SUBSCRIBE, and the
same "Event" header field. Rules for comparisons of the "Event"
headers are described in section 7.2.1. If a matching NOTIFY request
contains a "Subscription-State" of "active" or "pending", it creates
a new subscription and a new dialog (unless they have already been
created by a matching response, as described above).

If an initial SUBSCRIBE is sent on a pre-existing dialog, a matching
200-class response or successful NOTIFY request merely creates a new
subscription associated with that dialog.

Multiple subscriptions can be associated with a single dialog.
Subscriptions may also exist in dialogs associated with INVITE-
created application state and other application state created by
mechanisms defined in other specifications. These sets of
application state do not interact beyond the behavior described for a
dialog (e.g., route set handling).

A subscription is destroyed when a notifier sends a NOTIFY request
with a "Subscription-State" of "terminated".

   A subscriber may send a SUBSCRIBE request with an "Expires" header
   of 0 in order to trigger the sending of such a NOTIFY request;
   however, for the purposes of subscription and dialog lifetime, the
   subscription is not considered terminated until the NOTIFY with a
   "Subscription-State" of "terminated" is sent.

If a subscription's destruction leaves no other application state
associated with the dialog, the dialog terminates. The destruction
of other application state (such as that created by an INVITE) will
not terminate the dialog if a subscription is still associated with
that dialog.

   Note that the above behavior means that a dialog created with an
   INVITE does not necessarily terminate upon receipt of a BYE.
   Similarly, in the case that several subscriptions are associated
   with a single dialog, the dialog does not terminate until all the
   subscriptions in it are destroyed.

3.3.5. State Agents and Notifier Migration

   When state agents (see section 4.4.11.) 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 message with a "Subscription-State" header of "terminated",
and a reason parameter of "deactivated". This NOTIFY request is
otherwise normal, and is formed as described in section 3.2.2.

Upon receipt of this NOTIFY message, the subscriber SHOULD attempt to
re-subscribe (as described in the preceding sections). Note that
this subscription is established on a new dialog, and does not re-use
the route set from the previous subscription dialog.

The actual migration is effected by making a change to the policy
(such as routing decisions) of one or more servers to which the
SUBSCRIBE request will be sent in such a way that a different node
ends up responding to the SUBSCRIBE request. This may be as simple
as a change in the local policy in the notifier from which the
subscription is migrating so that it serves as a proxy or redirect
server instead of a notifier.

Whether, when, and why to perform notifier migrations may be
described in individual event packages; otherwise, such decisions are
a matter of local notifier policy, and are left up to individual
implementations.

3.3.6. 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 a SUBSCRIBE with an "Expires" of 0.

Of course, an immediate fetch while a subscription is active may be
effected by sending a SUBSCRIBE with an "Expires" equal to the number
of seconds remaining in the subscription.

Upon receipt of this SUBSCRIBE request, the notifier (or notifiers,
if the SUBSCRIBE request was forked) will send a NOTIFY request
containing resource state in the same dialog.

Note that the NOTIFY messages triggered by SUBSCRIBE messages with
"Expires" headers of 0 will contain a "Subscription-State" value of
"terminated", and a "reason" parameter of "timeout".

Polling of event state can cause significant increases in load on the
network and notifiers; as such, it should be used only sparingly. In
particular, polling SHOULD NOT be used in circumstances in which it
will typically result in more network messages than long-running
subscriptions.

When polling is used, subscribers SHOULD attempt to cache
authentication credentials between polls so as to reduce the number
of messages sent.

3.3.7. 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
all methods which initiate dialogs and their responses (such as
INVITE) and OPTIONS responses.

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.

3.3.8. PINT Compatibility

The "Event" header is considered mandatory for the purposes of this
document. However, to maintain compatibility with PINT (see [2]),
servers MAY interpret a SUBSCRIBE request with no "Event" header as
requesting a subscription to PINT events. If a server does not
support PINT, it SHOULD return "489 Bad Event" to any SUBSCRIBE
messages without an "Event" header.

4. Event Packages

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

4.1. Appropriateness of Usage

When designing an event package using the methods described in this
document 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 follow the development of "Guidelines for
   Authors of SIP Extensions" [7] 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 hundredth of a second could easily overload such a network.

4.2. Event Template-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.

   Event template-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. Event template-packages may even be
   applied to other event template-packages.

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

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

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

4.3. 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.

4.3.1. Complete State Information

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 some circumstances, conveying the current state alone may be
insufficient for a particular class of events. In these cases, the
event packages should include complete state information along with
the event that occurred. For example, conveying "no customer service
representatives available" may not be as useful as conveying "no
customer service representatives available; representative
sip:46@cs.xyz.int just logged off".

4.3.2. State Deltas

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.

Any event package that supports delta changes to states MUST include
a version number that increases by exactly one for each NOTIFY
transaction in a subscription. Note that the state version number
appears in the body of the message, not in a SIP header.

If a NOTIFY arrives that has a version number that is incremented by
more than one, the subscriber knows that a state delta has been
missed; it ignores the NOTIFY message containing the state delta

(except for the version number, which it retains to detect message
loss), and re-sends a SUBSCRIBE to force a NOTIFY containing a
complete state snapshot.

4.4. Event Package Responsibilities

Event packages are not required to reiterate 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 weakened 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 in standards-track
RFCs and SIP extension documents, authors of event packages need
to address each of the issues detailed in the following
subsections, whenever applicable.

4.4.1. Event Package Name

This section, which MUST be present, 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 6.

4.4.2. 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.

4.4.3. 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.

4.4.4. 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.

4.4.5. NOTIFY Bodies

The NOTIFY body is used to report state on the resource being
monitored. Each package MUST define 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 MUST define which MIME type is to be assumed if
none are specified in the "Accept" header of the SUBSCRIBE request.

4.4.6. 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 5.2.).

4.4.7. 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 4.3. Such a section is
required.

This section may optionally describe the behavior used to process the
subsequent response.

4.4.8. 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).

4.4.9. Handling of forked requests

   Each event package MUST specify whether forked SUBSCRIBE requests are
   allowed to install multiple subscriptions.

   If such behavior is not allowed, the first potential dialog-
   establishing message will create a dialog. All subsequent NOTIFY
   messages which correspond to the SUBSCRIBE message (i.e., match "To",
   "From", "From" header "tag" parameter, "Call-ID", "CSeq", "Event",
   and "Event" header "id" parameter) but which do not match the dialog
   would be rejected with a 481 response. Note that the 200-class
   response to the SUBSCRIBE can arrive after a matching NOTIFY has been
   received; such responses might not correlate to the same dialog
   established by the NOTIFY. Except as required to complete the
   SUBSCRIBE transaction, such non-matching 200-class responses are
   ignored.

   If installing of multiple subscriptions by way of a single forked
   SUBSCRIBE is allowed, the subscriber establishes a new dialog towards
   each notifier by returning a 200-class response to each NOTIFY. Each
   dialog is then handled as its own entity, and is refreshed
   independent of the other dialogs.

   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 dialog is tied to a mutually exclusive state
   which is unaffected by the other dialogs; this MUST be clearly stated
   if it is the case.

4.4.10. Rate of notifications

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

   Each package MAY further define a throttle mechanism which allows
   subscribers to further limit the rate of notification.

4.4.11. 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 provide 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.

   Event packages MAY also outline specific scenarios under which
   notifier migrations take place.

4.4.12. 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 document
   for exact protocol details.

4.4.13. Use of URIs to Retrieve State

   Some types of event packages may define state information which is
   potentially too large to reasonably send in a SIP message. To
   alleviate this problem, event packages may include the ability to
   convey a URI instead of state information; this URI will then be used
   to retrieve the actual state information.

   The precise mechanisms for conveying such URIs are out of the scope
   of this document.

5. Security Considerations

5.1. Access Control

   The ability to accept subscriptions should be under the direct
   control of the notifier's 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 subscriber identity (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 document.

5.2. Notifier Privacy Mechanism

   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.

   Individual packages and their related documents for which such a mode
   of operation makes sense can further describe how and why to generate
   such potentially correct data. For example, such a mode of operation
   is mandated by RFC 2779 [6] for user presence information.

5.3. 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 SIP [1].

5.4. Replay Attacks

   Replaying of either SUBSCRIBE or NOTIFY can have detrimental effects.

   In the case of SUBSCRIBE messages, attackers may be able to install
   any arbitrary subscription which it witnessed being installed at some
   point in the past. Replaying of NOTIFY messages may be used to spoof
   old state information (although a good versioning mechanism in the
   body of the NOTIFY messages may help mitigate such an attack). Note
   that the prohibition on sending NOTIFY messages to nodes which have
   not subscribed to an event also aids in mitigating the effects of
   such an attack.

   To prevent such attacks, implementations SHOULD require
   authentication with anti-replay protection. Authentication issues
   are discussed in SIP [1].

5.5. Man-in-the middle attacks

   Even with authentication, man-in-the-middle attacks using SUBSCRIBE
   may be used to install arbitrary subscriptions, hijack existing
   subscriptions, terminate outstanding subscriptions, or modify the
   resource to which a subscription is being made. To prevent such
   attacks, implementations SHOULD provide integrity protection across
   "Contact", "Route", "Expires", "Event", and "To" headers of SUBSCRIBE
   messages, at a minimum. If SUBSCRIBE bodies are used to define
   further information about the state of the call, they SHOULD be
   included in the integrity protection scheme.

   Man-in-the-middle attacks may also attempt to use NOTIFY messages to
   spoof arbitrary state information and/or terminate outstanding
   subscriptions. To prevent such attacks, implementations SHOULD
   provide integrity protection across the "Call-ID", "CSeq", and
   "Subscription-State" headers and the bodies of NOTIFY messages.

   Integrity protection of message headers and bodies is discussed in
   SIP [1].

5.6. Confidentiality

   The state information contained in a NOTIFY message has the potential
   to contain sensitive information. Implementations MAY encrypt such
   information to ensure confidentiality.

   While less likely, it is also possible that the information contained
   in a SUBSCRIBE message contains information that users might not want
   to have revealed. Implementations MAY encrypt such information to
   ensure confidentiality.

To allow the remote party to hide information it considers sensitive,
all implementations SHOULD be able to handle encrypted SUBSCRIBE and
NOTIFY messages.

The mechanisms for providing confidentiality are detailed in SIP [1].

6. IANA Considerations

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 template-packages; see section 4.2. To avoid confusion,
template-package names and package names share the same namespace; in
other words, an event template-package MUST NOT share a name with a
package.

Following the policies outlined in "Guidelines for Writing an IANA
Considerations Section in RFCs" [4], normal event package
identification tokens are allocated as First Come First Served, and
event template-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 template-package. Further,
packages MUST include contact information for the party responsible
for the registration and/or a published document which describes the
event package. Event template-package token registrations MUST
include a pointer to the published RFC which defines the event
template-package.

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

6.1. Registration Information

As this document specifies no package or template-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 template-packages
defined in "SIP-Specific Event Notification" [RFC3265]. Each name is
designated as a package or a template-package under "Type".

```
   Package Name        Type          Contact       Reference
   ------------        ----          -------       ---------
   example1            package       [Roach]
   example2            package       [Roach]       [RFC3265]
   example3            package                     [RFC3265]
   example4            template      [Roach]       [RFC3265]
   example5            template                    [RFC3265]

   PEOPLE
   ------
   [Roach] Adam Roach <adam@dynamicsoft.com>

   REFERENCES
   ----------
   [RFC3265] Roach, A., "SIP-Specific Event Notification", RFC 3265,
             June 2002.
```

6.2. Registration Template

   To: ietf-sip-events@iana.org
   Subject: Registration of new SIP event package

   Package Name:

      (Package names must conform to the syntax described in
      section 7.2.1.)

   Is this registration for a Template Package:

      (indicate yes or no)

   Published Specification(s):

      (Template packages require a published RFC. Other packages
      may reference a specification when appropriate).

   Person & email address to contact for further information:

6.3. Header Field Names

   This document registers three new header field names, described
   elsewhere in this document. These headers are defined by the
   following information, which is to be added to the header sub-
   registry under http://www.iana.org/assignments/sip-parameters.

   Header Name: Allow-Events
   Compact Form: u

      Header Name: Subscription-State
      Compact Form: (none)

      Header Name: Event
      Compact Form: o

6.4. Response Codes

   This document registers two new response codes. These response codes
   are defined by the following information, which is to be added to the
   method and response-code sub-registry under
   http://www.iana.org/assignments/sip-parameters.

   Response Code Number: 202
   Default Reason Phrase: Accepted

   Response Code Number: 489
   Default Reason Phrase: Bad Event

7. Syntax

   This section describes the syntax extensions required for event
   notification in SIP. Semantics are described in section 3. Note
   that the formal syntax definitions described in this document are
   expressed in the ABNF format used in SIP [1], and contain references
   to elements defined therein.

7.1. New Methods

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

   This table expands on tables 2 and 3 in SIP [1].

      Header   Where   SUB NOT
      ------   -----   --- ---
      Accept   R   o   o
      Accept   2xx   -   -
      Accept   415   o   o
      Accept-Encoding   R   o   o
      Accept-Encoding   2xx   -   -
      Accept-Encoding   415   o   o
      Accept-Language   R   o   o
      Accept-Language   2xx   -   -
      Accept-Language   415   o   o
      Alert-Info   R   -   -
      Alert-Info   180   -   -
      Allow   R   o   o

Allow	2xx	o	o
Allow	r	o	o
Allow	405	m	m
Authentication-Info	2xx	o	o
Authorization	R	o	o
Call-ID	c	m	m
Contact	R	m	m
Contact	1xx	o	o
Contact	2xx	m	o
Contact	3xx	m	m
Contact	485	o	o
Content-Disposition		o	o
Content-Encoding		o	o
Content-Language		o	o
Content-Length		t	t
Content-Type		*	*
CSeq	c	m	m
Date		o	o
Error-Info	300-699	o	o
Expires		o	-
Expires	2xx	m	-
From	c	m	m
In-Reply-To	R	-	-
Max-Forwards	R	m	m
Min-Expires	423	m	-
MIME-Version		o	o
Organization		o	-
Priority	R	o	-
Proxy-Authenticate	407	m	m
Proxy-Authorization	R	o	o
Proxy-Require	R	o	o
RAck	R	-	-
Record-Route	R	o	o
Record-Route	2xx,401,484	o	o
Reply-To		-	-
Require		o	o
Retry-After	404,413,480,486	o	o
Retry-After	500,503	o	o
Retry-After	600,603	o	o
Route	R	c	c
RSeq	1xx	o	o
Server	r	o	o
Subject	R	-	-
Supported	R	o	o
Supported	2xx	o	o
Timestamp		o	o
To	c(1)	m	m
Unsupported	420	o	o

```
   User-Agent                            o    o
   Via                             c     m    m
   Warning                         R     -    o
   Warning                         r     o    o
   WWW-Authenticate               401    m    m
```

7.1.1. 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.

7.1.2. 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.

7.2. New Headers

   This table expands on tables 2 and 3 in SIP [1], as amended by the
   changes described in section 7.1.

```
   Header field       where proxy ACK BYE CAN INV OPT REG PRA SUB NOT
   ------------------------------------------------------------------
   Allow-Events         R          o   o   -   o   o   o   o   o   o
   Allow-Events        2xx         -   o   -   o   o   o   o   o   o
   Allow-Events        489         -   -   -   -   -   -   -   m   m
   Event                R          -   -   -   -   -   -   -   m   m
   Subscription-State  R          -   -   -   -   -   -   -   -   m
```

7.2.1. "Event" header

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

   For the purposes of matching responses and NOTIFY messages with
   SUBSCRIBE messages, the event-type portion of the "Event" header is
   compared byte-by-byte, and the "id" parameter token (if present) is
   compared byte-by-byte. An "Event" header containing an "id"
   parameter never matches an "Event" header without an "id" parameter.
   No other parameters are considered when performing a comparison.

Note that the forgoing text means that "Event: foo; id=1234" would
match "Event: foo; param=abcd; id=1234", but not "Event: foo" (id
does not match) or "Event: Foo; id=1234" (event portion does not
match).

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.

7.2.2. "Allow-Events" Header

Allow-Events is added to the definition of the element "general-
header" in the SIP message grammar. Its usage is described in
section 3.3.7.

7.2.3. "Subscription-State" Header

Subscription-State is added to the definition of the element
"request-header" in the SIP message grammar. Its usage is described
in section 3.2.4.

7.3. New Response Codes

7.3.1. "202 Accepted" Response Code

The 202 response is added to the "Success" header field definition.
"202 Accepted" has the same meaning as that defined in HTTP/1.1 [3].

7.3.2. "489 Bad Event" Response Code

The 489 event response is added to the "Client-Error" header field
definition. "489 Bad Event" is used to indicate that the server did
not understand the event package specified in a "Event" header field.

7.4. Augmented BNF Definitions

The Augmented BNF definitions for the various new and modified syntax
elements follows. The notation is as used in SIP [1], and any
elements not defined in this section are as defined in SIP and the
documents to which it refers.

SUBSCRIBEm   = %x53.55.42.53.43.52.49.42.45 ; SUBSCRIBE in caps
NOTIFYm   = %x4E.4F.54.49.46.59 ; NOTIFY in caps
extension-method = SUBSCRIBEm / NOTIFYm / token

```
    Event               = ( "Event" / "o" ) HCOLON event-type
                            *( SEMI event-param )
    event-type          = event-package *( "." event-template )
    event-package       = token-nodot
    event-template      = token-nodot
    token-nodot         = 1*( alphanum / "-"  / "!" / "%" / "*"
                               / "_" / "+" / "`" / "'" / "~" )
    event-param         = generic-param / ( "id" EQUAL token )

    Allow-Events = ( "Allow-Events" / "u" ) HCOLON event-type
                    *(COMMA event-type)

    Subscription-State   = "Subscription-State" HCOLON substate-value
                            *( SEMI subexp-params )
    substate-value       = "active" / "pending" / "terminated"
                            / extension-substate
    extension-substate   = token
    subexp-params        =   ("reason" EQUAL event-reason-value)
                            / ("expires" EQUAL delta-seconds)
                            / ("retry-after" EQUAL delta-seconds)
                            / generic-param
    event-reason-value   =   "deactivated"
                            / "probation"
                            / "rejected"
                            / "timeout"
                            / "giveup"
                            / "noresource"
                            / event-reason-extension
    event-reason-extension = token
```

8. Normative References

   [1]   Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A.,
         Peterson, J., Sparks, R., Handley, M. and E. Schooler, "SIP:
         Session Initiation Protocol", RFC 3261, June 2002.

   [2]   Petrack, S. and L. Conroy, "The PINT Service Protocol", RFC
         2848, June 2000.

   [3]   Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L.,
         Leach, P. and T. Berners-Lee, "Hypertext Transfer Protocol --
         HTTP/1.1", RFC 2616, June 1999.

   [4]   Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA
         Considerations Section in RFCs", BCP 26, RFC 2434, October
         1998.

   [5]   Bradner, S., "Key Words for Use in RFCs to Indicate Requirement
          Levels", BCP 14, RFC 2119, March 1997.

   [6]   Day, M., Aggarwal, S., Mohr, G. and J. Vincent, "Instant
          Messaging/Presence Protocol Requirements", RFC 2779, February
          2000.

9. Informative References

   [7]   Rosenberg, J. and H. Schulzrinne, "Guidelines for Authors of
          SIP Extensions", Work in Progress.

   [8]   Schulzrinne, H. and J. Rosenberg, "SIP Caller Preferences and
          Callee Capabilities", 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 for
   miscellaneous guidance, Jonathan Rosenberg for a thorough scrubbing
   of the -00 draft, and the authors of the "SIP Extensions for
   Presence" document for their input to SUBSCRIBE and NOTIFY request
   semantics.

11. Notice Regarding Intellectual Property Rights

   The IETF has been notified of intellectual property rights claimed in
   regard to some or all of the specification contained in this
   document. For more information, consult the online list of claimed
   rights at http://www.ietf.org/ipr.html

12. Author's Address

   Adam Roach
   dynamicsoft
   5100 Tennyson Parkway
   Suite 1200
   Plano, TX 75024
   USA

   EMail: adam@dynamicsoft.com
   Voice: sip:adam@dynamicsoft.com

13. Full Copyright Statement

Acknowledgement