          An Extension to Session Initiation Protocol (SIP) Events
                    for Conditional Event Notification

Abstract

   The Session Initiation Protocol (SIP) events framework enables
   receiving asynchronous notification of various events from other SIP
   user agents. This framework defines the procedures for creating,
   refreshing, and terminating subscriptions, as well as fetching and
   periodic polling of resource state. These procedures provide no
   tools to avoid replaying event notifications that have already been
   received by a user agent. This memo defines an extension to SIP
   events that allows the subscriber to condition the subscription
   request to whether the state has changed since the previous
   notification was received. When such a condition is true, either the
   body of a resulting event notification or the entire notification
   message is suppressed.

Table of Contents

1. Introduction

The Session Initiation Protocol (SIP) events framework provides an extensible facility for requesting notification of certain events from other SIP user agents. This framework includes procedures for creating, refreshing, and terminating subscriptions, as well as the possibility to fetch or periodically poll the event resource.

Several instantiations of this framework, called event packages have been defined, e.g., for presence [RFC3856], message waiting indications [RFC3842], and registrations [RFC3680].

By default, every SUBSCRIBE request generates a NOTIFY request containing the latest event state. Typically, a SUBSCRIBE request is issued by the subscriber whenever it needs a subscription to be installed, periodically refreshed, or terminated. Once the subscription has been installed, the majority of the NOTIFYs generated by the subscription refreshes are superfluous; the subscriber usually is in possession of the event state already, except in the unlikely case where a state change exactly coincides with the periodic subscription refresh. In most cases, the final event state generated upon terminating the subscription similarly contains resource state that the subscriber already has.

Fetching or polling of resource state behaves in a similarly suboptimal way in cases where the state has not changed since the previous poll occurred. In general, the problem lies with the inability to persist state across a SUBSCRIBE request.

This memo defines an extension to optimize the SIP events framework. This extension allows a notifier to tag notifications (called entity-tags hereafter) and the subscriber to condition its subsequent SUBSCRIBE requests for actual changes since a notification carrying that entity-tag was issued. The solution is similar to conditional requests defined in the Hypertext Transfer Protocol (HTTP) [RFC2616], and follows the mechanism already defined for the PUBLISH [RFC3903] method for issuing conditional event publications.

This memo is structured as follows. Section 2 explains the background, motivations, and requirements for the work; Section 3 gives a general overview of the mechanism; Section 4 explains the underlying model for resources and entities as they apply to conditional notification; Section 5 defines the subscriber behavior; Section 6 defines the notifier behavior; Section 7 includes the protocol element definitions; Section 8 includes the IANA considerations; and Section 9 includes the security considerations.

[1.1](#). Document Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT",
"SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this
document are to be interpreted as described in [BCP 14](#), [RFC 2119](#)
[[RFC2119](#)] and indicate requirement levels for compliant
implementations.

[1.2](#). Terminology

In addition to the terminology introduced in [[RFC3261](#)], [[RFC3265](#)],
and [[RFC3903](#)], this specification uses these additional terms to
describe the objects of conditional notification:

resource
   An object identified by a URI whose resource state can be accessed
   using the SIP Event Notification framework. There is a single
   authoritative notifier responsible for communicating the resource
   state.

entity
   The representation of resource state. An entity consists of the
   state data carried in the body of a NOTIFY message, as well as
   related meta-data in the message header. There may be many
   versions of an entity, one current and the others stale. Each
   version of an entity is identified by an entity-tag, which is
   guaranteed to be unique across all versions of all entities for a
   resource and event package.

[2](#). Motivations and Background

[2.1](#). Overview

A SUBSCRIBE request creates a subscription with a finite lifetime.
This lifetime is negotiated using the Expires header field, and
unless the subscription is refreshed by the subscriber before the
expiration is met, the subscription is terminated. The frequency of
these subscription refreshes depends on the event package, and
typically ranges from minutes to hours.

[2.2](#). Problem Description

The SIP events framework does not include different protocol methods
for initiating and terminating of subscriptions, subscription
refreshes, and fetches inside and outside of the SIP dialog. The
SUBSCRIBE method is overloaded to perform all of these functions.
The difference between a fetch that does not create a (lasting)
subscription and a SUBSCRIBE that creates one is in the Expires

header field value of the SUBSCRIBE; a zero-expiry SUBSCRIBE only
generates a single NOTIFY, after which the subscription immediately
terminates. Lasting subscriptions typically have relatively short
expiry periods, requiring periodic sending of new SUBSCRIBE requests
in order to refresh the subscription.

Each new SUBSCRIBE request generates a NOTIFY request containing the
latest resource state. Even if the state has not changed, it is sent
again in response to each poll or subscription refresh. This is very
similar to the HTTP [RFC2616] problem of repeated GET operations on a
resource. HTTP solves the problem using conditional requests. The
server versions each entity with an entity-tag that identifies a
specific instance of that entity. Clients making GET requests can
then include the entity-tag for the version of the entity that they
believe to be current in an "If-None-Match" header field. The server
can compare this entity-tag to the entity it believes to be current
and suppress resending the entity in the response if the server
believes the client's version matches. In other words, the server
doesn't resend information that the client has already received.

The SIP PUBLISH [RFC3903] method uses a similar mechanism, where a
refresh of a publication is done by reference to its assigned entity-
tag, instead of retransmitting the event state each time the
publication expiration is extended.

2.3. Requirements

As a summary, here is the required functionality to solve the
presented issues:

REQ1:   It must be possible to suppress the NOTIFY request (or at a
        minimum, the event body therein) if the subscriber is already
        in possession of (or has previously received and discarded)
        the latest event state of the resource.

REQ2:   This mechanism must apply to initial subscriptions in which
        the subscriber is attempting to resume an earlier
        subscription that has been paused.

REQ3:   This mechanism must apply to refreshing a subscription.

REQ4:   This mechanism must apply to terminating a subscription
        (i.e., an unsubscribe).

REQ5:   This mechanism must apply to fetching or polling of resource
        state.

3. Overview of Operation

   Whenever a subscriber initiates a subscription, it issues a SUBSCRIBE
   request. The SUBSCRIBE request is sent, routed, and processed by the
   notifier normally, i.e., according to the Session Initiation Protocol
   [RFC3261] and SIP-Specific Event Notification [RFC3265].

   If the notifier receiving the SUBSCRIBE request supports conditional
   subscriptions, it generates an entity-tag for the current entity, and
   includes it in a SIP-ETag header field of the NOTIFY request. The
   entity-tag is unique across all versions of all entities for a
   resource and event package. See Section 4 for more on this.

   Entity-tags are independent of subscriptions. This allows
   notifications generated to a fetch or a poll to have valid entity-
   tags even across subsequent fetches or polls.

   The subscriber will store the entity-tag received in the notification
   along with the resource state. It can then later use this entity-tag
   to make a SUBSCRIBE contain a condition in the form of a "Suppress-
   If-Match" header field. Unlike the "If-Match" condition in a PUBLISH
   [RFC3903] request, which applies to whether the PUBLISH succeeds or
   returns an error, this condition applies to the stream of
   notifications that are sent after the SUBSCRIBE request has been
   processed.

   The Suppress-If-Match header field contains the last entity-tag seen
   by the subscriber. This condition, if true, instructs the notifier
   to suppress either the body of a subsequent notification, or the
   entire notification.

   The condition is evaluated by matching the value of the header field
   against the entity-tag of the entity that would normally be sent in
   the associated NOTIFY message. There is also a wildcard entity-tag
   with a special value of "*" that always matches.

```
   Subscriber                            Notifier
   ----------                            --------

   (1) SUBSCRIBE        -------->
       Expires: 3600
                        <--------        (2) 200 (or 202)

                        <--------        (3) NOTIFY
                                             Subscription-State: active
                                             SIP-ETag: ffee2
   (4) 200              -------->

          ... time passes ...

   (5) SUBSCRIBE        -------->                 \ if "ffee2"
       Suppress-If-Match: ffee2                   |   matches
       Expires: 3600                              |   local
                                                  |   entity-tag
                                                  |
                        <--------        (6) 204  / then

       ... time passes and resource state (entity) changes...


                        <--------        (7) NOTIFY
                                             Subscription-State: active
                                             SIP-ETag: ca89a
   (8) 200              -------->


          ... time passes ...


   (9) SUBSCRIBE        -------->                 \ if "ca89"
       Suppress-If-Match: ca89a                   |   matches
       Expires: 0                                 |   local
                                                  |   entity-tag
                                                  |
                        <--------        (10) 204  / then
```

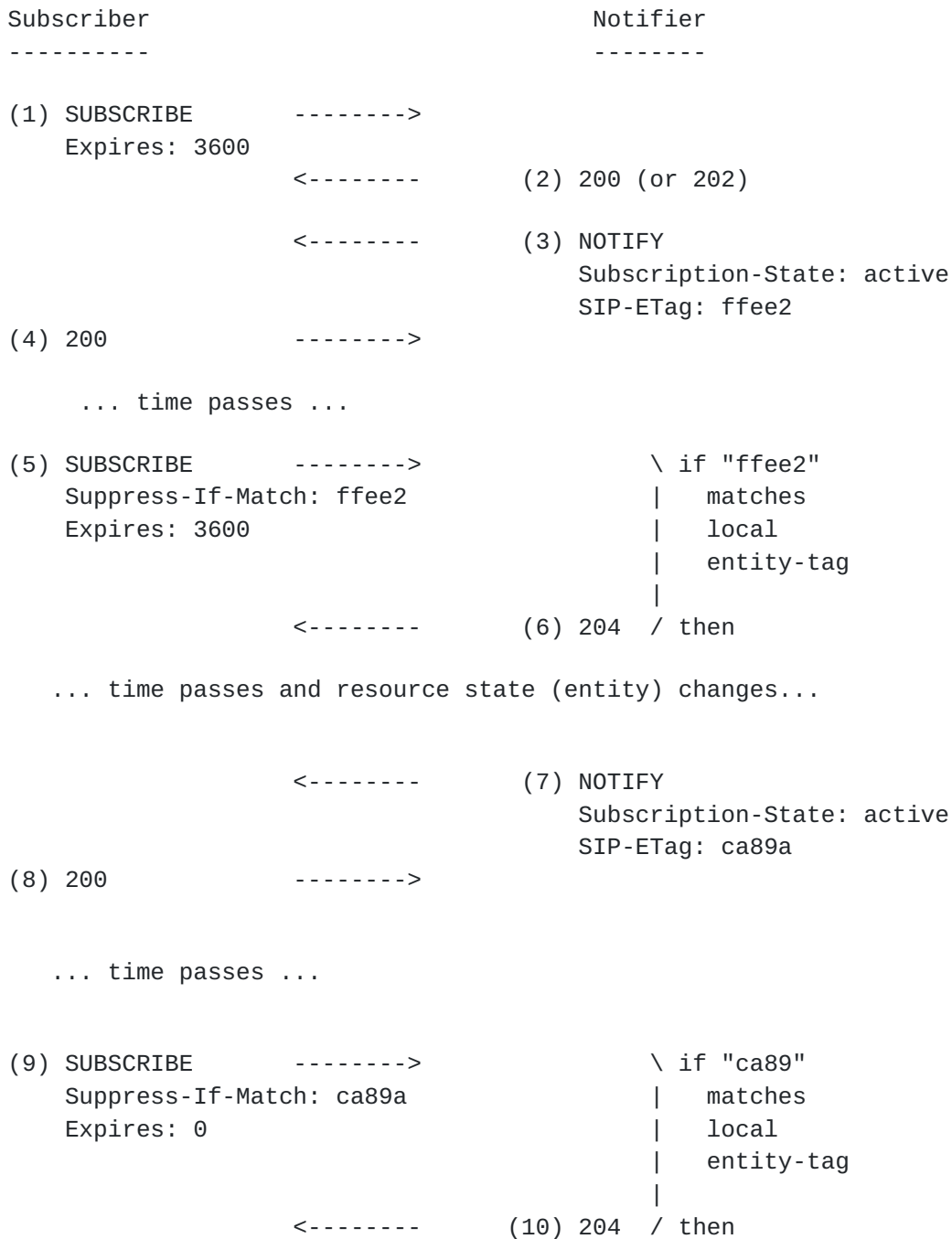   Figure 1: Example Message Flow

   Figure 1 describes a typical message flow for conditional
   notification:

   (1)   The subscriber initiates a subscription by sending a SUBSCRIBE
         request for a resource.

   (2)   After proper authentication and authorization, the notifier
         accepts the subscription.

   (3)   The notifier then immediately sends the initial event
         notification, including a unique entity-tag in a SIP-ETag
         header field.

   (4)   The subscriber accepts the notification and stores the entity-
         tag value along with the resource state.

   (5)   Later, the subscriber refreshes the subscription, and includes
         an entity-tag in a Suppress-If-Match header field.

   (6)   The notifier evaluates the condition by matching its local
         entity-tag value for the resource against the value of the
         Suppress-If-Match header field. If the condition evaluates to
         true, the notifier informs the subscriber that the notification
         will not be sent.

   (7)   At some point, the state of the resource changes, e.g., the
         presence status of a user changes from online to busy. This
         triggers an event notification with a new value in the SIP-ETag
         header field.

   (8)   The subscriber accepts the notification and stores the new
         entity-tag along with the resource state.

   (9)   After a while, the subscriber decides to terminate the
         subscription. It adds a condition for Suppress-If-Match, and
         includes the entity-tag it received in the previous NOTIFY.

   (10) The notifier evaluates the condition by matching its entity-tag
         for the resource against the value of the Suppress-If-Match
         header field. If the condition evaluates to true, the notifier
         informs the subscriber that no notification will be sent. This
         concludes the subscription.

   The benefit of using conditional notification in this example is in
   the reduction of the number of NOTIFY requests the subscriber can
   expect to receive. Each event notification that the subscriber has
   already seen is suppressed by the notifier. This example illustrates
   only one use case for the mechanism; the same principles can be used
   to optimize the flow of messages related to other event notification
   use cases.

4. Resource Model for Entity-Tags

   The key to understanding how conditional notification works is
   understanding the underlying resource model of event notification.
   In general, this model is similar to the resource model of HTTP with
   some key differences. This section explains in detail the model as
   it applies to SIP events. Figure 2 illustrates the model.

```
                        +-----+
    ............        |     |
    .          .        | URI |
    . Represen .        |     |
    . tation   .        +-----+
    .          .           |*
    ............           |
         .                 |
         .                 V
         .            +----------+                 +---------+
     composition |            |*            |  Event  |
         +------<>|  Resource |----------->| Package |<----.
         |        |           |            |         |     |
         |        +----------+            +----.----+     |
         |                                   /_\           |
         |*                                   | classification
      +--------+                              |           |
      |        |      .----------------.------'           |
      | Entity |      |                |                  |
      |        |      |                |                  |*
      +--------+  +----------+    +------------+    +----------+
          ^       |          |    |            |    |          |
          |       | Presence |    | Conference |    | Template |
          |       |          |    |            |    |          |
         |1..*    +----------+    +------------+    +----.-----+
      +---------+                                        /_\
      |         |                                         |
      | Version |                                         |
      |         |                                +---------+
      +---------+                                | Watcher |
          |1                                     |  Info   |
          |                                      |         |
          V                                      +---------+
      +---------+
      | Entity- |
      |   Tag   |
      |         |
      +---------+
```
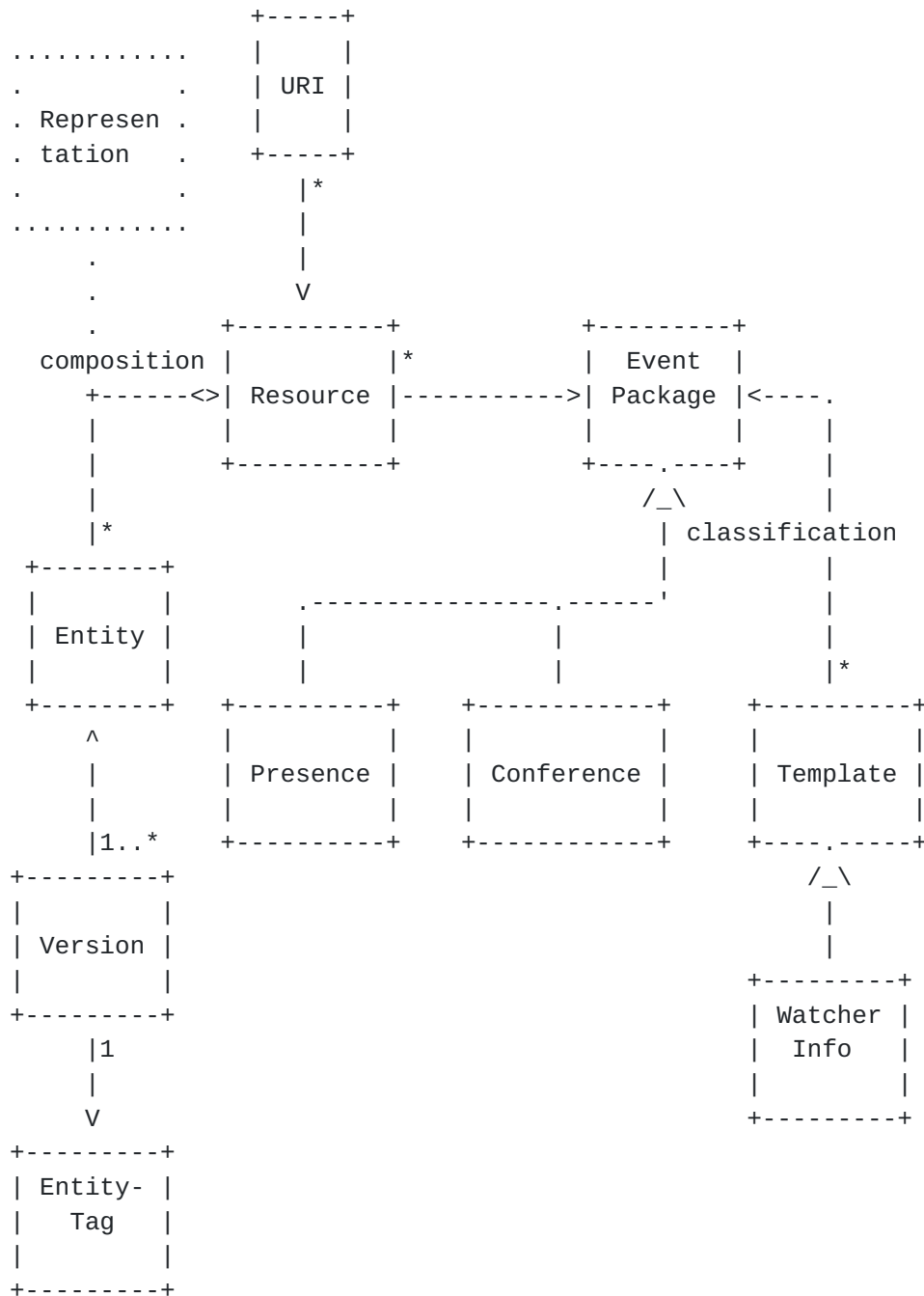
               Figure 2: Resource Model Diagram

For a given event package, there is a single authoritative agent
responsible for zero or more resources. That is, even for a
distributed agent, the resource state is uniform across all
instances. The resource itself can be a list of resources [RFC4662].
Conditional notification for list subscriptions is addressed in
Section 6.5.

A resource is identified by zero or more URIs, which can be SIP URIs,
pres URIs [RFC3859], or similar. Subscribers use this URI to
subscribe to the resource for certain types of events, identified by
the event package.

With a successful subscription, a subscriber receives event
notifications that communicate the resource state and the changes
thereto. Each event notification carries a representation of the
current resource state. This representation is influenced by many
factors, e.g., authorization and filtering rules, and the event
composition rules of the notifier.

This representation is realized in an "entity". Each resource may be
associated with zero or more entities. For example, there may be
multiple subscribers to the presence information of a single user (a
resource), and each subscriber may have a different filtered view of
that resource, producing one entity per subscriber. However, each
entity is associated with one and only one resource; there is no
"compositing" of resources at the entity level. Resources may
themselves be made up of information from other resources (be
"composite resources"), but this does not change the one-resource-
per-entity rule.

An entity consists of the data carried in the body of a NOTIFY
message and related meta-data in the message header. Whenever the
data in the body or any of the meta-data changes, the notifier MUST
produce a new entity-tag. This meta-data MUST include, but is not
limited to the following SIP header fields defined in the Session
Initiation Protocol [RFC3261] and SIP Specific Event Notification
[RFC3265]:

     1.   Content-Disposition

     2.   Content-Encoding

     3.   Content-Language

     4.   Content-Length

     5.   Content-Type

      6. Event

   Note that the Subscription-State is explicitly not part of the
   entity. In the future, event packages may define additional fields
   that implementations need to consider as part of the entity.

   An entity has one or more versions of which only one is current and
   all others stale. Each version has an entity-tag, which uniquely
   identifies it across all versions of all entities pertaining to a
   single resource and event package.

   Note that two entity-tags for different resources being equal does
   not indicate identical entities. In other words, if an entity-tag
   received for a subscription to a first resource matches an entity-tag
   received for a subscription to a second resource, the subscriber
   cannot assume that the two entity values are equal.

   With partial event notification, the NOTIFY message only carries the
   delta state, or the set of changes to the previous version of the
   entity. In that case, implementations MUST consider the full event
   state as the version of the entity to which the entity-tag in the
   NOTIFY message applies.

   The conditional notification mechanism is independent of the way in
   which subscriptions are installed. In other words, the mechanism
   supports implicit subscriptions, such as those associated with the
   REFER method [RFC3515].

   It is possible that the same resource is in some shape or form
   accessible through another mechanism in addition to SIP Event
   Notification, e.g., HTTP or the SIP PUBLISH method. In general,
   implementations MUST NOT expect the entity-tags to be shared between
   the mechanisms, unless event packages or specific applications of SIP
   events explicitly define such dependencies.

5. Subscriber Behavior

   This section augments the subscriber behavior defined in RFC 3265
   [RFC3265]. It first discusses general issues related to indicating
   support for the mechanism (Section 5.1) and creating conditions in
   SUBSCRIBE requests (Section 5.2). Next, it describes subscriber
   behavior for receiving NOTIFY requests (Section 5.3), and specific
   client workflows for polling resource state (Section 5.4), resuming a
   subscription (Section 5.5), refreshing a subscription (Section 5.6),
   and terminating a subscription (Section 5.7). Finally, handling of
   transient errors is discussed (Section 5.8).

5.1. Detecting Support for Conditional Notification

   The mechanism defined in this memo is backwards compatible with SIP
   events [RFC3265] in that a notifier supporting this mechanism will
   insert a SIP entity-tag in its NOTIFY requests, and a subscriber that
   understands this mechanism will know how to use it in creating a
   conditional request.

   Unaware subscribers will simply ignore the entity-tag, make requests
   without conditions, and receive the default treatment from the
   notifier. Unaware notifiers will simply ignore the conditional
   header fields and continue normal operation.

5.2. Generating SUBSCRIBE Requests

   When creating a conditional SUBSCRIBE request, the subscriber MUST
   include a single conditional header field including an entity-tag in
   the request. The condition is evaluated by comparing the entity-tag
   of the subscribed resource with the entity-tag carried in the
   conditional header field. If they match, the condition evaluates to
   true.

   Unlike the condition introduced for the SIP PUBLISH [RFC3903] method,
   these conditions do not apply to the SUBSCRIBE request itself, but to
   the resulting NOTIFY requests. When true, the condition drives the
   notifier to change its behavior with regard to sending the
   notifications after the SUBSCRIBE.

   This specification defines a new header field called Suppress-If-
   Match. This header field introduces a condition to the SUBSCRIBE
   request. If true, it instructs the notifier either to omit the body
   of the resulting NOTIFY message (if the SUBSCRIBE is not sent within
   an existing dialog) or to suppress (i.e., block) the NOTIFY request
   that would otherwise be triggered by the SUBSCRIBE (for an
   established dialog). In the latter case, the SUBSCRIBE message will
   be answered with a 204 (No Notification) response. As long as the
   condition remains true, it also instructs the notifier either to
   suppress any subsequent NOTIFY request or, if there are reportable
   changes in the NOTIFY header, e.g., the Subscription-State has
   changed, to suppress the body of any subsequent NOTIFY request.

   If the condition is false, the notifier follows its default behavior.

   If the subscriber receives a 204 (No Notification) response to an in-
   dialog SUBSCRIBE, the subscriber MUST consider the event state and
   the subscription state unchanged.

The value of the Suppress-If-Match header field is an entity-tag,
which is an opaque token that the subscriber simply copies (byte-
wise) from a previously received NOTIFY request. Inclusion of an
entity-tag in a Suppress-If-Match header field of a SUBSCRIBE request
indicates that the client has a copy of, or is capable of recreating
a copy of, the entity associated with that entity-tag.

Example:

    Suppress-If-Match: b4cf7

The header field can also be wildcarded using the special "*" entity-
tag value. Such a condition always evaluates to true regardless of
the value of the current entity-tag for the resource.

Example:

    Suppress-If-Match: *

Such a wildcard condition effectively quenches a subscription; the
only notifications received are those reporting changes to the
subscription state and those in response to a SUBSCRIBE message sent
outside of an existing dialog. In both cases, the notifications will
not contain a body.

    A subscription with a wildcard Suppress-If-Match condition is
    useful in scenarios where the subscriber wants to temporarily put
    a subscription in dormant mode. For example, a host may want to
    conserve bandwidth and power when it detects from screen or input
    device inactivity that the user isn't actively monitoring the
    presence statuses of contacts.

5.3. Receiving NOTIFY Requests

When a subscriber receives a NOTIFY request that contains a SIP-ETag
header field, it MUST store the entity-tag if it wishes to make use
of the conditional notification mechanism. The subscriber MUST be
prepared to receive a NOTIFY with any entity-tag value, including a
value that matches any previous value that the subscriber might have
seen.

The subscriber MUST NOT infer any meaning from the value of an
entity-tag; specifically, the subscriber MUST NOT assume identical
entities (i.e., event state) for NOTIFYs with identical entity-tag
values when those NOTIFYs result from subscription to different
resources.

   Note that there are valid cases for which identical entity-tag
   values on different resources may occur. For example, it is
   possible to generate entity-tag values using a one-way hash
   function, resulting in the possibility that two different
   resources having the same entity-value will also have the same
   entity-tag. Clients however MUST NOT assume that this is the
   case, as the algorithm for the generation of entity-tags is
   notifier-dependent and not negotiated with the subscriber.
   Consequently, the subscriber cannot differentiate between two
   entity-tags that have the same value because they are similar
   hashes of identical entities, or because two notifiers happen to
   have used the same sequential number as an entity-tag. Entity
   tags are only required to be unique for a given resource, not
   globally unique.

5.4. Polling or Fetching Resource State

   Polling with conditional notification allows a user agent to
   efficiently poll resource state. This is accomplished using the
   Suppress-If-Match condition:

```
  Subscriber                              Notifier
  ----------                              --------

 (1) SUBSCRIBE        -------->
     Expires: 0
                      <--------       (2) 202

                      <--------       (3) NOTIFY
                                          Subscription-State: terminated
                                          SIP-ETag: f2e45
                                          Content-Length: 17539

 (4) 200              -------->


... poll interval elapses ...


 (5) SUBSCRIBE        -------->
     Suppress-If-Match: f2e45
     Expires: 0
                      <--------       (6) 202

                      <--------       (7) NOTIFY
                                          Subscription-State: terminated
                                          SIP-ETag: f2e45
                                          Content-Length: 0

 (8) 200              -------->
```

                     Figure 3: Polling Resource State

   (1) The subscriber polls for resource state by sending a SUBSCRIBE
        with zero expiry (expires immediately).

   (2) The notifier accepts the SUBSCRIBE with a 202 (Accepted)
        response.

   (3) The notifier then immediately sends a first (and last) NOTIFY
        request with the current resource state and the current entity-
        tag in the SIP-ETag header field.

   (4) The subscriber accepts the notification with a 200 (OK)
        response.

   (5) After some arbitrary poll interval, the subscriber sends another
        SUBSCRIBE with a Suppress-If-Match header field that includes
        the entity-tag received in the previous NOTIFY.

(6) The notifier accepts the SUBSCRIBE with a 202 (Accepted)
     response. (202 would be used to indicate that the subscription
     request was understood without also indicating that it was
     authorized, as per Section 3.1.6.1 of SIP-Specific Event
     Notification [RFC3265].)

(7) Since the resource state has not changed since the previous poll
     occurred, the notifier sends a NOTIFY message with no body. It
     also mirrors the current entity-tag of the resource in the SIP-
     ETag header field.

(8) The subscriber accepts the notification with a 200 (OK)
     response.

5.5. Resuming a Subscription

Resuming a subscription means the ability to continue an earlier
subscription that either closed abruptly or was explicitly
terminated. When resuming, the subscription is established without
transmitting the resource state. This is accomplished with
conditional notification and the Suppress-If-Match header field:

```
        Subscriber                            Notifier
        ----------                            --------

   (1) SUBSCRIBE        -------->
       Suppress-If-Match: ega23
       Expires: 3600
                        <--------      (2) 202

                        <--------      (3) NOTIFY
                                           Subscription-State: active
                                           SIP-ETag: ega23
                                           Content-Length: 0
   (4) 200              -------->
```

                 Figure 4: Resuming a Subscription

(1) The subscriber attempts to resume an earlier subscription by
     including a Suppress-If-Match header field with the entity-tag
     it last received.

(2) The notifier accepts the subscription after proper
     authentication and authorization, by sending a 202 (Accepted)
     response.

(3) Since the condition is true, the notifier then immediately sends
      an initial NOTIFY request that has no body. It also mirrors the
      current entity-tag of the resource in the SIP-ETag header field.

(4) The subscriber accepts the NOTIFY and sends a 200 (OK) response.

Had the entity-tag not been valid any longer, the condition would
have evaluated to false, and the NOTIFY would have had a body
containing the latest resource state.

[5.6](#). Refreshing a Subscription

To refresh a subscription using conditional notification, the
subscriber creates a subscription refresh before the subscription
expires, and uses the Suppress-If-Match header field:

```
     Subscriber                              Notifier
     ----------                              --------

     (1) SUBSCRIBE       -------->
         Suppress-If-Match: aba91
         Expires: 3600

                         <--------       (2) 204
                                             Expires: 3600
```

Figure 5: Refreshing a Subscription

(1) Before the subscription expires, the subscriber sends a
      SUBSCRIBE request that includes the Suppress-If-Match header
      field with the latest entity-tag it has seen.

(2) If the condition evaluates to true, the notifier sends a 204 (No
      Notification) response and sends no NOTIFY request. The Expires
      header field of the 204 (No Notification) response indicates the
      new expiry time.

[5.7](#). Terminating a Subscription

To terminate a subscription using conditional notification, the
subscriber creates a SUBSCRIBE request with a Suppress-If-Match
condition:

```
       Subscriber                                   Notifier
       ----------                                   --------

       (1) SUBSCRIBE        -------->
           Suppress-If-Match: ega23
           Expires: 0


                            <--------        (2) 204
```

                    Figure 6: Terminating a Subscription

   (1) The subscriber decides to terminate the subscription and sends a
        SUBSCRIBE request with the Suppress-If-Match condition with the
        entity-tag it has last seen.

   (2) If the condition evaluates to true, the notifier sends a 204 (No
        Notification) response, which concludes the subscription, and
        the subscriber can clear all state related to the subscription.

5.8. Handling Transient Errors

   This section is non-normative.

   In some deployments, there may be Back-to-Back User Agent (B2BUA)
   devices that track SIP dialogs such as subscription dialogs. These
   devices may be unaware of the conditional notification mechanism.

   It is possible that some B2BUA devices may treat a NOTIFY with
   suppressed body as an error, or may expect all SUBSCRIBE messages to
   have an associated NOTIFY message.

   In general, there is very little that an endpoint can do to recover
   from such transient errors. The most that can be done is to try to
   detect such errors, and define a fallback behavior.

   If subscribers encounter transient errors in conditional
   notification, they should disable the feature and fall back to normal
   subscription behavior.

6. Notifier Behavior

   This section augments the notifier behavior as specified in RFC 3265
   [RFC3265].

6.1. Generating Entity-tags

   An entity-tag is a token carried in the SIP-ETag header field, and it
   is opaque to the client. The notifier is free to decide on any means
   for generating the entity-tag. It can have any value, except for
   "*". For example, one possible method is to implement the entity-tag
   as a simple counter, incrementing it by one for each generated
   notification per resource.

   A notifier MUST generate entity-tags for event notifications of all
   resources for which it is responsible. The entity-tag MUST be unique
   across all versions of all entities for each state of a resource as
   reported by a given event package. Otherwise said, for any
   subscription or sequence of subscriptions to a specific resource
   using a singular event package, each entity-tag produced MUST map to
   one and only one presentation of resource state (entity). Two
   identical entities for a specific resource might or might not have
   identical entity-tags; this decision is left to the notifier.

   An entity-tag is considered valid for as long as the entity exists.
   An entity becomes stale when its version is no longer the current
   one. The notifier MUST remember (or be able to recalculate) the
   entity-tag of an entity as long as the version of the entity is
   current. The notifier MAY remember the entity-tag longer than this,
   e.g., for implementing journaled state differentials (Section 6.4).

   The entity-tag values used in publications are not necessarily shared
   with the entity-tag values used in subscriptions. This is because
   there may not always be a one-to-one mapping between a publication
   and a notification of state change; there may be several sources to
   the event composition process, and a publication into a resource may
   not affect the resulting entity.

6.2. Suppressing NOTIFY Bodies

   When a condition in a SUBSCRIBE request for suppressing notifications
   is true (i.e., the local entity-tag for the resource state and the
   entity-tag in a Suppress-If-Match header field are byte-wise
   identical) but there are reportable changes in the NOTIFY header
   (e.g., the Subscription-State has changed), the notifier MUST
   suppress the body of the NOTIFY request. That is, the resulting
   NOTIFY contains no Content-Type header field, the Content-Length is
   set to zero, and no payload is attached to the message.

Additionally, when a condition in a SUBSCRIBE request for suppressing
notifications is true and the SUBSCRIBE message is not sent within an
established dialog, the notifier MUST send a NOTIFY request with a
suppressed entity body.

Suppressing the entity body of a NOTIFY does not change the current
entity-tag of the resource. Hence, the NOTIFY MUST contain a SIP-
ETag header field that contains the unchanged entity-tag of the
resource state.

A Suppress-If-Match header field that includes an entity-tag with the
value of "*" MUST always evaluate to true.

6.3. Suppressing NOTIFY Requests

When a condition in a SUBSCRIBE request to suppress notifications is
true (i.e., the local entity-tag of the resource and the entity-tag
in a Suppress-If-Match header field match), and the SUBSCRIBE is sent
within an established dialog, then the notifier MUST suppress the
resulting NOTIFY request, and generate a 204 (No Notification)
response. As long as the condition remains true, and there are no
reportable changes in the NOTIFY header, all subsequent NOTIFY
requests MUST also be suppressed.

Notifiers MUST NOT suppress a NOTIFY unless the corresponding
SUBSCRIBE message was sent in an established dialog.

A successful conditional SUBSCRIBE request MUST extend the
subscription expiry time.

Suppressing the entire NOTIFY has no effect on the entity-tag of the
resource. In other words, it remains unchanged.

A Suppress-If-Match header field that includes an entity-tag with the
value of "*" MUST always evaluate to true.

6.4. State Differentials

Some event packages support a scheme where notifications contain
state differentials, or state deltas [RFC3265], instead of complete
resource state.

Further extensions could define means for notifiers to keep track of
the state changes of a resource, e.g., storing the changes in a
journal. If a condition fails, the notifier would then send a state
differential in the NOTIFY rather than the full state of the event
resource. This is only possible if the event package and the
subscriber both support a payload format that has this capability.

When state differentials are sent, the SIP-ETag header field MUST
contain an entity-tag that corresponds to the full resource state.

6.5. List Subscriptions

The Event Notification Extension for Resource Lists [RFC4662] defines
a mechanism for subscribing to a homogeneous list of resources using
the SIP events framework.

A list subscription delivers event notifications that contain both
Resource List Meta-Information (RLMI) documents as well as the
resource state of the individual resources on the list.

Implementations MUST consider the full resource state of a resource
list including RLMI and the entity-header as the entity to which the
entity-tag applies.

7. Protocol Element Definitions

This section describes the protocol extensions required for
conditional notification.

7.1. 204 (No Notification) Response Code

The 204 (No Notification) response code indicates that the request
was successful, but the notification associated with the request will
not be sent. It is valid only in response to a SUBSCRIBE message
sent within an established dialog.

The response code is added to the "Success" production rule in the
SIP [RFC3261] message grammar.

7.2. Suppress-If-Match Header Field

The Suppress-If-Match header field is added to the definition of the
"message-header" rule in the SIP [RFC3261] grammar. Its use is
described in Sections 5, 6.3, and 6.2.

This header field is allowed to appear in any request, but its
behavior is only defined for the SUBSCRIBE request.

7.3. Grammar

This section defines the formal syntax for extensions described in
this memo in Augmented BNF (ABNF) [RFC5234]. The rules defined here
augment and reference the syntax defined in RFC 3261 [RFC3261] and
RFC 3903 [RFC3903].

```
Success              =/ "204"  ;  No Notification

      ; Success is defined in RFC 3261.

message-header       =/ Suppress-If-Match

      ; message-header is defined in RFC 3261.

Suppress-If-Match  =  "Suppress-If-Match" HCOLON ( entity-tag / "*" )

      ; entity-tag is defined in RFC 3903.
```

8. IANA Considerations

This document registers a new response code and a new header field name.

8.1. 204 (No Notification) Response Code

This document registers a new response code. This response code is defined by the following information, which has been added to the methods and response-codes sub-registry available from http://www.iana.org.

This information has been added under "Successful 2xx" category.

```
+---------------------+-----------+
| Response Code       | Reference |
+---------------------+-----------+
| 204 No Notification | [RFC5839] |
+---------------------+-----------+
```

8.2. Suppress-If-Match Header Field

This document registers a new SIP header field called Suppress-If-Match. This header field is defined by the following information, which has been added to the header fields sub-registry available from http://www.iana.org.

```
+-------------------+---------+-----------+
| Header Name       | Compact | Reference |
+-------------------+---------+-----------+
| Suppress-If-Match |         | [RFC5839] |
+-------------------+---------+-----------+
```

9. Security Considerations

The security considerations for SIP event notification are
extensively discussed in RFC 3265 [RFC3265]. This specification
introduces an optimization to SIP event notification, which in itself
does not alter the security properties of the protocol.

10. Acknowledgments

The following people have contributed corrections and suggestions to
this document: Adam Roach, Sean Olson, Johnny Vrancken, Pekka Pessi,
Eva Leppanen, Krisztian Kiss, Peili Xu, Avshalom Houri, David
Viamonte, Jonathan Rosenberg, Qian Sun, Dale Worley, Tolga Asveren,
Brian Stucker, Eric Rescorla, Arun Arunachalam, and the SIP and
SIMPLE working groups.

11. References

11.1. Normative References

[RFC2119]   Bradner, S., "Key words for use in RFCs to Indicate
            Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC3261]   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.

[RFC3265]   Roach, A., "Session Initiation Protocol (SIP)-Specific
            Event Notification", RFC 3265, June 2002.

[RFC3903]   Niemi, A., "Session Initiation Protocol (SIP) Extension
            for Event State Publication", RFC 3903, October 2004.

[RFC5234]   Crocker, D. and P. Overell, "Augmented BNF for Syntax
            Specifications: ABNF", STD 68, RFC 5234, January 2008.

11.2. Informative References

[RFC2616]   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.

[RFC3515]   Sparks, R., "The Session Initiation Protocol (SIP) Refer
            Method", RFC 3515, April 2003.

[RFC3680]   Rosenberg, J., "A Session Initiation Protocol (SIP) Event
            Package for Registrations", RFC 3680, March 2004.

   [RFC3842]   Mahy, R., "A Message Summary and Message Waiting
               Indication Event Package for the Session Initiation
               Protocol (SIP)", [RFC 3842](#), August 2004.

   [RFC3856]   Rosenberg, J., "A Presence Event Package for the Session
               Initiation Protocol (SIP)", [RFC 3856](#), August 2004.

   [RFC3859]   Peterson, J., "Common Profile for Presence (CPP)",
               [RFC 3859](#), August 2004.

   [RFC4662]   Roach, A., Campbell, B., and J. Rosenberg, "A Session
               Initiation Protocol (SIP) Event Notification Extension for
               Resource Lists", [RFC 4662](#), August 2006.

Authors' Addresses

   Aki Niemi
   Nokia
   P.O. Box 407
   NOKIA GROUP, FIN 00045
   Finland

   Phone: +358 50 389 1644
   EMail: aki.niemi@nokia.com

   Dean Willis (editor)
   Softarmor Systems
   3100 Independence Pkwy #311-164
   Plano, TX 75075
   USA

   Phone: +1 214 504 1987
   EMail: dean.willis@softarmor.com