Network Working Group   H. Khartabil
Request for Comments: 4660   Telio
Category: Standards Track   E. Leppanen
                                                             M. Lonnfors
                                                        J. Costa-Requena
                                                                   Nokia
                                                          September 2006

           Functional Description of Event Notification Filtering

Status of This Memo

   This document specifies an Internet standards track protocol for the
   Internet community, and requests discussion and suggestions for
   improvements. Please refer to the current edition of the "Internet
   Official Protocol Standards" (STD 1) for the standardization state
   and status of this protocol. Distribution of this memo is unlimited.

Abstract

   The SIP event notification framework describes the usage of the
   Session Initiation Protocol (SIP) for subscriptions and notifications
   of changes to the state of a resource. The document does not
   describe a mechanism whereby filtering of event notification
   information can be achieved.

   This document describes the operations a subscriber performs in order
   to put filtering rules associated with a subscription to event
   notification information in place. The handling, by the subscriber,
   of responses to subscriptions carrying filtering rules and the
   handling of notifications with filtering rules applied to them are
   also described. Furthermore, the document conveys how the notifier
   behaves when receiving such filtering rules and how a notification is
   constructed.

Table of Contents

1. Introduction

   SIP event notification is described in [3]. It defines a general
   framework for sending subscriptions and receiving notifications in
   SIP-based systems. It introduces the concept of event packages,
   which are concrete applications of the general event framework to a
   specific usage of events.

   Filtering is a mechanism for controlling the content of event
   notifications. Additionally, the subscriber may specify the rules
   for when a notification should be sent to it. The filtering
   mechanism is expected to be particularly valuable to users of mobile
   wireless access devices. The characteristics of the devices
   typically include high latency, low bandwidth, low data processing
   capabilities, small display, and limited battery power. Such devices
   can benefit from the ability to filter the amount of information
   generated at the source of the event notification. However,
   implementers need to be aware of the computational burden on the
   source of the event notification. This is discussed further in
   Section 8.

   It is stated in [3] that the notifier may send a NOTIFY at any time,
   but typically it is sent when the state of the resource changes. It
   also states that the notifications would contain the complete and
   current state of the resource authorized for a certain subscriber to
   see. The format of such resource state information is package
   specific. In this memo, we assume that the NOTIFY for any package
   contains an XML document.

   This document, together with [5], presents a mechanism for filtering
   whereby a subscriber describes its preference of when notifications
   are to be sent to it and what they are to contain. It also describes
   how the notifier functions when generating notifications by taking
   into account filters and default functionality of the package/
   service.

   The XML format for defining the filter is described in [5].

2. Conventions

   In this document, the key words 'MUST', 'MUST NOT', 'REQUIRED',
   'SHALL', 'SHALL NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED', 'MAY',
   and 'OPTIONAL' are to be interpreted as described in RFC 2119 [1] and
   indicate requirement levels for compliant implementations.

   "Content" refers to the XML document that appears in a notification
   reflecting the state of a resource.

3. Client Operation

3.1. Transport Mechanism

   Transportation of the filter to the server is achieved by inserting
   the XML document, as defined in [5], in the body of the SUBSCRIBE
   request. Alternatively, the XML document can be uploaded to the
   server using means outside the scope of this document.

3.2. SUBSCRIBE Bodies

   SIP entities compliant with this specification MUST support the
   content type 'application/simple-filter+xml'.

3.3. Subscriber Generating of SUBSCRIBE Requests

   This section presents additional functionality required from the
   subscriber when filters are used in the bodies of the SUBSCRIBE
   requests. Normal operations of services (e.g., as defined in [8],
   [10], and [4]) are otherwise followed.

   As defined in [3], the SUBSCRIBE message MAY contain a body. This
   body would carry filtering information. Honouring those filters is
   at the discretion of the notifier and might depend on local policies.

   No content in the body of a SUBSCRIBE indicates to the notifier that
   no filter is being requested, so the notifier is instructed to send
   all the NOTIFY requests using the notifier's own or service-specific
   policy. Note that, for example, in the list case [4], the filter
   might have been uploaded to the server beforehand (by means outside
   the scope of this document).

   If the body of the SUBSCRIBE includes the filter, the body MUST be of
   the MIME-Type 'application/simple-filter+xml'.

3.3.1. Defining the Filtering Rules

   Multiple filters MAY be included in one SUBSCRIBE. This is achieved
   by including multiple <filter> elements in the filter [5]. Each
   <filter> element may include a 'uri' attribute.

   A SUBSCRIBE request destined to a list URI [4] MAY include multiple
   filters specific to individual resources. This is achieved by
   including multiple <filter> elements with different URIs of resources
   in each of those elements. This resource specific resource-specific
   filter are processed first before any list specific list-specific
   filter, if any. The list specific list-specific filter may or may
   not include a URI.

Furthermore, regardless of whether the SUBSCRIBE is destined to a
list URI, there can only be one filter applicable to a single
resource or domain within a single SUBSCRIBE. That is, each filter
within a subscription MUST uniquely identify one resource or one
domain.

A filter can be enabled and disabled using the 'enabled' attribute in
the <filter> element, as described in [5].

3.3.2. Request-URI vs. Filter URI

The URI in the filter defines the target resource. For example, in
the Presence service case, it is the presentity's presence
information to which the filter is applied. The subscriber MAY
choose to leave the URI in the filter undefined. If the URI is not
defined within the filter, the filter applies to the resource
identified in the Request-URI. Similarly, the subscriber MAY define
a filter URI. If the Request-URI is a list URI [4], the filter URI
MUST be the list URI, a sub-list URI, or resource whose URI is one of
the URIs that result from a lookup, by a Resource List Server (RLS),
on the Request-URI. If it is not, the filter may be ignored or may
be rejected. URI matching is done according to the matching rules
defined for a particular scheme (SIP URI matching rules are defined
in RFC 3261 [2]).

A filter may also be addressed to a domain using the 'domain'
attribute instead of the 'uri' attribute. In this case, the filter
applies to resources in that domain. This can be used when a
subscription is for a resource that is an event list with many
resources from differing domains. If an individual resource-specific
filter is present along with the domain filter, this
resource-specific filter overrides any domain-specific filter, if
any.

3.3.3. Changing Filters within a Dialog

The subscriber MAY reset or change the filter by re-issuing a new
SUBSCRIBE request within the existing dialog. A SUBSCRIBE within the
exiting dialog that does not contain a filter is assumed to maintain
existing filters. This means that filters are persistent within a
dialog and are only explicitly removed.

A subscriber requiring removal of a filter may do so by using the
'remove="true"' attribute, as defined in [5].

In the case where the URI in the filter is that of a list, a
subscriber may override the existing filter with a filter for an
individual resource that is part of the list subscribed to earlier by

issuing a new SUBSCRIBE within the existing dialog and including a
filter, specific for that individual resource, using a new filter ID.
The new filter need not include the original filter since a filter is
only removed in the manner indicated above.

A filter is replaced by the subscriber re-issuing the filter using
the same filter ID and replacing the contents of the filter.
Replacing a filter by changing the filter ID and keeping the resource
URI is considered an error since this causes the server to assume
that two filters are placed for the same resource.

Again, a filter can be disabled and re-enabled using the 'enabled'
attribute in the <filter> element, as described in [5].

3.3.4. Subscriber Interpreting of SIP Responses

The SUBSCRIBE request will be confirmed with a final response. A
200-class response indicates 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 filter
is accepted. A "202" response merely indicates that the subscription
has been understood, that the content type has been accepted, and
that authorization may or may not have been granted. A "202"
response also indicates that the filter has not been accepted yet.
The acceptance of the filter MAY arrive in a subsequent NOTIFY.

A non-200 class final response indicates that no subscription or
dialog has been created, and no subsequent NOTIFY message will be
sent. All non-200 class final responses have the same meanings and
handling as described in [2] and [3].

Specifically, a "415" response indicates that the MIME type
'application/simple-filter+xml' is not understood by the notifier. A
"488" response indicates that the content type (filter) is understood
but some aspects of it were either not understood or not accepted.

3.4. Subscriber Processing of NOTIFY Requests

If the 2xx response was returned for the SUBSCRIBE, the NOTIFY that
follows MAY contain a body that describes the present state of the
resource after the filters have been applied.

If the NOTIFY indicates that a subscription has been terminated [3],
the subscription is assumed to be terminated. Behaviour in such
events is also described in [3].

If the subscription is indicated as active, NOTIFY requests are
handled as described in package-specific documents and in [3].

4. Resource List Server Behaviour

   The Resource List Server is defined in [4]. This section describes
   how such an entity behaves in the presence of a filter in a
   subscription to a list.

4.1. Request-URI vs. Filter URI

   If the URI is not defined within the filter, the filter applies to
   the resource list identified in the Request-URI of the SUBSCRIBE
   request. This results in the filters being applied to all the
   notifications that the RLS issues to this subscription. The same
   processing applies to a filter that defines a URI that matches the
   request-URI of the SUBSCRIBE request. That is, the filter applies to
   all notifications that the RLS issues to this subscription.

   If the URI indicated by the filter is for one resource whose URI is
   one of the URIs that result from a lookup by the RLS on the
   Request-URI, the filter for that particular resource is extracted and
   propagated in the SUBSCRIBE request sent to that resource. It is
   possible to have more than one filter in a SUBSCRIBE request body,
   and therefore a filter specific to a resource MUST be extracted and
   only that one is propagated. For example, if the Request-URI in a
   SUBSCRIBE has the value "sip:mybuddies@example.com", where
   "bob@example.com" is a resource belonging to that list, and the URI
   in a filter is "sip:bob@example.com", the filter specific for Bob is
   extracted and placed in the body of the SUBSCRIBE sent to
   "bob@example.com".

   If the URI indicated by the filter is for one resource whose URI is
   NOT under the RLS administrative control, the RLS propagates the
   filter to all the fanned out subscriptions. This is to accommodate
   the scenario where the subscriber knows that there are sub-lists in
   the event list that are under a different administrative domain from
   that where the original subscription was sent, and the subscriber
   wishes to set a filter for a resource in that sub-list.

   If the URI indicated by the filter is for one resource whose URI is
   under the RLS administrative control but is not part of the resource
   list that the subscription was addressed to, the filter is not
   propagated. In this case, it is the RLS's responsibility to make
   sure that this filter is applied to notifications issued, if
   information about that resource is present.

For example: If we have 2 lists, each located on its own RLS:

List1 (list1@example.com) on RLS1 has: bob@example.com

list2@biloxi.com

List2 on RLS2 has: alice@biloxi.com sarah@example.com
(Note: list2 is a resource in list1)

RLS1 receives the following SUBSCRIBE request (the SUBSCRIBE is
addressed to list1 and contains 2 filters: one for sarah@example.com
and the other for alice@biloxi.com):

```
SUBSCRIBE sip:List1@example.com SIP/2.0
...
<?xml version="1.0" encoding="UTF-8"?>
<filter-set xmlns="urn:ietf:params:xml:ns:simple-filter">
  <ns-bindings>
    <ns-binding prefix="pidf" urn="urn:ietf:params:xml:ns:pidf"/>
  </ns-bindings>
  <filter id="999" uri="sip:sarah@example.com">
    <what>
      <include type="namespace">
        urn:ietf:params:xml:ns:pidf</include>
      <exclude>
        //pidf:tuple/pidf:note</exclude>
    </what>
  </filter>
  <filter id="8439" uri="sip:alice@biloxi.com">
    <what>
      <include>
        //pidf:tuple/pidf:status/pidf:basic</include>
    </what>
  </filter>
</filter-set>
```

RLS1 fans out subscriptions to resources on list1. The text above
suggests that if a filter is destined to a resource that is not part
of the list and is outside the administrative domain of an RLS, then
that filter is propagated. The rest are consumed. In our example,
only the filter to alice@biloxi.com is propagated since biloxi.com is
not under the administrative domain of RLS1. The filter to
sarah@example.com is consumed, and RLS1 needs to apply that filter to
notifications it receives.

URI matching is done according to the matching rules defined for a
particular scheme (SIP URI matching rules are defined in RFC 3261
[2]).

   A filter may also be addressed to a domain using the 'domain'
   attribute instead of the 'uri' attribute. In this case, the filter
   applies to resources in that domain, and the RLS MUST NOT apply
   filters to any notifications it sends. Instead, it MUST forward the
   filter with all fanned-out subscriptions to the notifiers.

   As indicated in Section 3.3.1, multiple filters can be present in a
   SUBSCRIBE request. Filters can also be added or modified as
   indicated in Section 3.3.3. In such circumstances, an RLS MUST check
   that there are no filters addressed to the same resource or domain,
   and if there are, it MUST reject the SUBSCRIBE request with a "488"
   error response.

4.2. Changing Filters within a Dialog

   If an RLS receives a subscription refresh request with no filters
   specified (empty payload), the RLS assumes that the client does not
   wish to update the filters. If an RLS receives a subscription
   refresh with a filter containing the 'remove="true"' attribute, as
   defined in [5], the RLS assumes that the client is removing that
   filter identified by the filter ID.

   If an RLS receives a subscription refresh request with a filter that
   already exists (i.e., having the same filter ID), the RLS interprets
   it as a replacement of the existing filter. Replacing a filter by
   changing the filter ID and keeping the resource URI is considered an
   error since this causes the RLS to assume that two filters are in
   place for the same resource.

   A filter can be disabled and re-enabled using the 'enabled' attribute
   in the <filter> element, as described in [5].

5. Server Operation

5.1. NOTIFY Bodies

   SIP entities compliant with this specification MUST support
   content-type 'application/simple-filter+xml'.

5.2. Notifier Processing of SUBSCRIBE Requests

   This section presents additional functionality required from the
   notifier when filters are used in the bodies of the SUBSCRIBE
   requests. Normal package-specific functionality is otherwise
   followed.

The notifier will examine the Content-Type header field and will
return a 415 response if it does not understand the content type
'application/simple-filter+xml'.

A 200-class response indicates that the subscription has been
accepted, and the NOTIFY will be sent immediately. A "200" response
indicates that the subscription has been accepted, the user is
authorized, and the filter is accepted. A "202" response merely
indicates that the subscription has been understood, but that the
authorization may or may not have been granted. A "202" response
also indicates that the filters have not been accepted yet. The
acceptance of the filters MAY arrive in a subsequent NOTIFY.

Procedures described in Section 5.4 are followed if an error is
encountered.

As indicated in Section 3.3.1, multiple filters can be present in a
SUBSCRIBE request. Filters can also be added or modified as
indicated in Section 3.3.3. In such circumstances, a server MUST
check that there are no filters addressed to the same resource or
domain, and if they are, it MUST reject the SUBSCRIBE request with a
"488" error response.

5.2.1. Request-URI vs. Filter URI

The subscriber may have chosen to leave the URI in the filter
undefined. If the URI is not defined within the filter, the filter
applies to the resource identified in the Request-URI.

Similarly, the subscriber may have chosen to include a URI in the
filter. In this case, the filter applies to all notifications sent
with content associated with the resource with that URI for this
subscription. If the Request-URI and the URI in the filter do not
match, the filter may be ignored or rejected. URI matching is done
according to the matching rules defined for a particular scheme (SIP
URI matching rules are defined in RFC 3261 [2]).

A filter may also be addressed to a domain using the 'domain'
attribute instead of the 'uri' attribute. In this case, the filter
applies to resources in that domain. A notifier MUST ignore any
filter using a 'domain' attribute containing a domain for which this
notifier is not responsible. The notifier MUST NOT apply such a
filter to any notification it sends. Notifiers belonging to the
domain MUST apply the filter to all notifications it sends for that
subscription, unless policy dictates otherwise.

5.2.2. Changing Filters within a Dialog

   If a server receives a subscription refresh request with no filters
   specified (empty payload), it assumes that the client does not wish
   to update the filters. If it receives a subscription refresh with a
   filter containing the 'remove="true"' attribute, as defined in [5],
   the server assumes that the client is removing the filter identified
   by the filter ID.

   If the server receives a subscription refresh request with a filter
   that already exists (i.e., having the same filter ID), it interprets
   it as a replacement of the existing filter. Replacing a filter by
   changing the filter ID and keeping the resource URI is considered an
   error since this causes the server to assume that two filters are
   placed for the same resource.

5.3. Notifier Generating of NOTIFY Requests

   Upon receiving the SUBSCRIBE with the filter, the notifier SHOULD
   retain the filter as long as the subscription persists. The filter
   MAY be incorporated within an existing subscription (in an active
   dialog) by sending a re-SUBSCRIBE that includes the filter in the
   body.

   If the response sent to the SUBSCRIBE was a "202" and the "202" was
   chosen because the filter could not be accepted that time, the NOTIFY
   MAY be used to terminate the subscription if the filter is found
   unacceptable.

   As described in [3], the NOTIFY message MAY contain a body that
   describes the state of the resource. This body is in one of the
   formats listed in the Accept header field of the SUBSCRIBE, or in the
   package-specific default if the Accept header field is omitted.

   Based on the contents of a filter, the following processing occurs:

   o A filter with only a <what> element will result in sending the
      requested resource state information in that <what> element
      whenever there is a change in the resource state.

   o A filter with only a <trigger> element will result in sending all
      resource state information whenever there is a change in the
      resource state that matches the triggers.

   o A filter with <what> and <trigger> elements will result in sending
      the requested resource state information in that <what> element
      whenever there is a change in the resource state that matches the
      triggers.

When a filter is disabled (by setting the 'enabled' attribute to
"false"), it means the same thing as the absence of that filter.
That is, all state and state changes are reported by issuing a
notification to the subscriber (assuming there are no other filters).

When a filter is re-enabled (by setting the 'enabled' attribute to
"true" or by omitting the 'enabled' attribute), the notifier behaves
as if the filter has just been placed by the SUBSCRIBE request
enabling it. Immediate NOTIFY rules, as stated in Section 5.3.1,
apply.

5.3.1. Generation of NOTIFY Contents

If the NOTIFY being sent is the one sent immediately after a 2xx
response to the original SUBSCRIBE, its contents MUST be populated
according to the filter <what> element, unless the processing of the
filters will take too long or the NOTIFY request is following a "202"
response to the SUBSCRIBE request and is terminating the
subscription. In the case that the filter is taking too long to
process, the NOTIFY request being sent may be empty or may be
populated with a pre-configured value as authorised to that
subscriber. If applying the filter results in no content to be
delivered, the NOTIFY MUST be sent with empty contents. If the
filter contains <trigger> elements, the notifier ignores the trigger
values when generating the first NOTIFY request.

The input to the content filter is a package-specific XML document
(e.g., [7] and [9]) derived according to the package-specific
specifications, (e.g., [8] and [10]).

The content is filtered according to the expressions in the <what>
element of the filter. The expression indicates the delivered XML
elements and/or attributes. Prefixes of the namespaces of the items
of the XML document to be filtered must be expanded before applying
the filter to the items.

The expression directly states the XML elements and attributes to be
delivered in the NOTIFY, along with their values. In addition to the
selected contents, the namespaces of all the selected items are also
included in the NOTIFY. The XML elements and/or attributes indicated
by the expression in the <what> element must be items that the
subscriber is authorised to see. If they are not, the notifier
policy dictates the behaviour of the notifier (which can ignore the
filter, parts of the filter, or reject the filter completely).
Implementers need to carefully consider such an implementation
decision; the subscriber may not be aware of the authorised contents
and therefore most likely will include a filter requesting
unauthorised contents. It is therefore RECOMMENDED that notifiers

just ignore the parts of the filter that are requesting unauthorised
info (i.e., the filter in the <filter> element where the unauthorised
contents are requested is ignored). If polite blocking is used by
the notifier, the notifier may choose to deliver notifications
containing bogus information in the unauthorised elements or
attributes and applying the filter afterwards.

The resultant XML document MUST be well formed and valid according to
the XML schema. This means that all mandatory elements and
attributes, along with their values, MUST be included in the XML
document regardless of the expression. In other words, if the result
of applying a filter on an XML document is a non-valid XML document,
the notifier MUST add elements and attributes, along with their
values, from the original XML document into the newly formulated one
in order for it to be valid.

5.3.2. Handling of Notification Triggering Rules

There can be several <trigger> elements inside one <filter> element.
If the criteria for any of the <trigger> elements are satisfied, a
NOTIFY SHOULD be generated.

The items (XML elements and/or attributes) indicated by the
expression in the <changed> element, <added> element, or <removed>
element must be items that the subscriber is authorised to access.
If they are not, the notifier policy dictates the behaviour of the
notifier (which can ignore the filter, parts of the filter, or reject
the filter completely).

5.4. Handling Abnormal Cases

In case of an invalid filter definition where the XML document of the
filter is not aligned with the XML schema of the filter format [5],
the notifier rejects the SUBSCRIBE request with a "488" response. A
Warning header field in the response may give a better indication as
to why the filters were not accepted. If the subscription was
accepted with a "202" response but the invalid filter was discovered
after that, a NOTIFY with a subscription-state of value 'terminated'
is sent. An event-reason-value "badfilter", introduced here, of
subexp-params [3] MAY be included.

In case of an erroneous expression in the filter definition, the
notifier either ignores the filter definition or terminates the
subscription.

If a <what> or <trigger> element is empty, the notifier proceeds as
if the element did not exist.

6. XML Document Validation

   The subscriber of the filter MUST ensure that the XML document
   inserted as the SUBSCRIBE request body is well formed and valid. The
   subscriber MUST NOT insert any extension elements or attributes into
   the XML document unless it has access to the extension schema and can
   validate the XML document. The XML document notifier MAY validate
   the XML document according to the schemas, including extension
   schemas, to which it has access that are applicable to this XML
   document.

7. Examples

   The following sections include filtering examples for Presence and
   Watcher Information. The format of filter is according to [5].

7.1. Presence Specific Examples

   This section describes three use cases where the presence information
   filtering solution is utilised [8]. In the first use case, the
   watcher is interested in getting messaging-specific information of a
   certain presentity. In the second use case, the watcher is
   interested in getting information about the communication means and
   contact addresses on which the presentity is currently available for
   communication. The third case shows how a presentity can request
   triggers to receive notifications.

   Below is the presentity's presence information in PIDF [7]. It
   includes two tuples: one for the instant messaging and another for
   the voice-related information.

```
<?xml version="1.0" encoding="UTF-8"?>
    <presence xmlns="urn:ietf:params:xml:ns:pidf"
                    xmlns:rpid="urn:ietf:params:xml:ns:pidf:rpid"
                    entity="sip:presentity@example.com">
       <tuple id="432sd">
          <status>
             <basic>closed</basic>
          </status>
          <rpid:class>IM</rpid:class>
          <contact>im:presentity@example.com</contact>
       </tuple>
       <tuple id="thr76jk">
          <status>
             <basic>open</basic>
          </status>
          <rpid:class>voice</rpid:class>
          <contact>tel:2224055555@example.com</contact>
```

```
            </tuple>
          </presence>
```

7.1.1. Subscriber Requests Messaging-Related Information

   The subscriber initiates a subscription to the presentity's messaging
   (MMS, IM, and SMS) related presence information. The subscription
   includes the content limiting filter.

   The filtered content is indicated with an expression. This
   expression selects the <basic> element and all the parent elements
   (i.e., the <status>, the <tuple>, and its root element), the <class>
   element, and the <contact> element. The filter matches if the
   <class> element contains "MMS", "SMS", or "IM".

   In this case, the notification includes the contents of the tuple
   that has the value "IM" in its <class> element.

   SUBSCRIBE request from the subscriber including filter:

   SUBSCRIBE sip:presentity@example.com
   Via: SIP/2.0/TCP client.example.com:5060;branch=z9hG4bKxjfdsjfk
   To: <sip:presentity@example.com>
   From: <sip:watcher@example.com>;tag:12341111
   Call-ID: 32432udfidfjmk342
   Cseq: 1 SUBSCRIBE
   Expires: 3600
   Event: Presence
   Contact: <sip:watcher@client.example.com>
   Content-Type: application/simple-filter+xml
   Content-Length: ...

   <?xml version="1.0" encoding="UTF-8"?>
   <filter-set xmlns="urn:ietf:params:xml:ns:simple-filter">
     <ns-bindings>
       <ns-binding prefix="pidf" urn="urn:ietf:params:xml:ns:pidf"/>
       <ns-binding prefix="rpid"
                         urn="urn:ietf:params:xml:ns:pidf:rpid"/>
     </ns-bindings>
     <filter id="123" uri="sip:presentity@example.com">
       <what>
         <include type="xpath">
           //pidf:tuple[rpid:class="IM" or rpid:class="SMS"
           or rpid:class="MMS"]/pidf:status/pidf:basic
         </include>
         <include type="xpath">
           //pidf:tuple[rpid:class="IM" or rpid:class="SMS"
           or rpid:class="MMS"]/rpid:class
```

```
      </include>
      <include type="xpath">
        //pidf:tuple[rpid:class="IM" or rpid:class="SMS"
        or rpid:class="MMS"]/pidf:contact
      </include>
      </what>
    </filter>
  </filter-set>
```

Notification to the subscriber:

```
NOTIFY sip:watcher@client.example.com SIP/2.0
Via: SIP/2.0/TCP presence.example.com:5060;branch=z9hG4bKxjfder
To: <sip:watcher@example.com>;tag:12341111
From: <sip:presentity@example.com>;tag:232321
Call-ID: 32432udfidfjmk342
Cseq: 1 NOTIFY
Event: Presence
Subscription-State: active; expires=3599
Contact: sip:presentity@server.example.com
Content-Type: application/pidf+xml
Content-Length: ...

<?xml version="1.0" encoding="UTF-8"?>
    <presence xmlns="urn:ietf:params:xml:ns:pidf"
    xmlns:rpid="urn:ietf:params:xml:ns:pidf:rpid"
    entity="sip:presentity@example.com">
        <tuple id="432sd">
           <status>
              <basic>closed</basic>
           </status>
           <rpid:class>IM</rpid:class>
           <contact>im:presentity@example.com</contact>
        </tuple>
    </presence>
```

7.1.2. Subscriber Fetches Information about "Open" Communication Means

   The subscriber makes a subscription to the presentity's available
   communication means. The subscription includes the content-limiting
   filter.

   The filtered content is indicated with an expression. This
   expression selects the <basic> element and all the parent elements
   (i.e., the <status>, the <tuple>, and its root element), the <class>
   element, and the <contact> element. The filter matches if the
   <basic> element's value is "open".

   In this case, the notification returns the contents of the tuple that
   has the value "open" inside the <status> element.

   SUBSCRIBE request from the subscriber including filter:

   SUBSCRIBE sip:presentity@example.com SIP/2.0
   Via: SIP/2.0/TCP client.example.com:5060;branch=z9hG4bKxjfdsjfk
   To: <sip:presentity@example.com>
   From: <sip:watcher@example.com>;tag:12341111
   Call-ID: 32432udfidfjmk342
   Cseq: 1 SUBSCRIBE
   Expires: 3600
   Event: Presence
   Contact: <sip:watcher@client.example.com>
   Content-Type: application/simple-filter+xml
   Content-Length: ...

   <?xml version="1.0" encoding="UTF-8"?>
   <filter-set xmlns="urn:ietf:params:xml:ns:simple-filter">
     <ns-bindings>
       <ns-binding prefix="pidf" urn="urn:ietf:params:xml:ns:pidf"/>
       <ns-binding prefix="rpid"
                          urn="urn:ietf:params:xml:ns:pidf:rpid"/>
     </ns-bindings>
     <filter id="123" uri="sip:presentity@example.com">
       <what>
         <include type="xpath">
           //pidf:tuple/pidf:status[pidf:basic="open"]/pidf:basic
         </include>
         <include type="xpath">
           //pidf:tuple[pidf:status/pidf:basic="open"]/rpid:class
         </include>
         <include type="xpath">
           //pidf:tuple[pidf:status/pidf:basic="open"]/pidf:contact
         </include>
       </what>
     </filter>
   </filter-set>

   Notification to the subscriber:

   NOTIFY sip:watcher@client.example.com SIP/2.0
   Via: SIP/2.0/TCP presence.example.com:5060;branch=z9hG4bKxjfder
   To: <sip:watcher@example.com>;tag:12341111
   From: <sip:presentity@example.com>;tag:232321
   Call-ID: 32432udfidfjmk342
   Cseq: 1 NOTIFY
   Event: Presence

```
   Subscription-State: active; expires=3599
   Contact: sip:presentity@server.example.com
   Content-Type: application/pidf+xml
   Content-Length: ...

   <?xml version="1.0" encoding="UTF-8"?>
      <presence xmlns="urn:ietf:params:xml:ns:pidf"
      xmlns:rpid="urn:ietf:params:xml:ns:pidf:rpid"
      entity="sip:presentity@example.com">
         <tuple id="thr76jk">
            <status>
               <basic>open</basic>
            </status>
               <rpid:class>voice</rpid:class>
               <contact>tel:2224055555@example.com</contact>
         </tuple>
      </presence>
```

7.1.3. Subscriber Requests Notifications When Presentity's Status
        Changes

   The subscriber subscribes to the presentity, specifying in the filter
   that it wants notifications only when the <basic> element has changed
   to value "open".

   SUBSCRIBE request from the subscriber including filter:

```
   SUBSCRIBE sip:presentity@example.com
   Via: SIP/2.0/TCP client.example.com:5060;branch=z9hG4bKxjfdsjfk
   To: <sip:presentity@example.com>
   From: <sip:watcher@example.com>;tag:12341111
   Call-ID: 32432udfidfjmk342
   Cseq: 1 SUBSCRIBE
   Expires: 3600
   Event: Presence
   Contact: <sip:watcher@client.example.com>
   Content-Type: application/simple-filter+xml
   Content-Length: ...

   <?xml version="1.0" encoding="UTF-8"?>
   <filter-set xmlns="urn:ietf:params:xml:ns:simple-filter">
     <ns-bindings>
       <ns-binding prefix="pidf" urn="urn:ietf:params:xml:ns:pidf"/>
     </ns-bindings>
     <filter id="123" uri="sip:presentity@example.com">
       <trigger>
         <changed from="closed" to="open">
           /pidf:presence/pidf:tuple/pidf:status/pidf:basic
```

```
            </changed>
          </trigger>
        </filter>
      </filter-set>
```

At some point during the subscription, a second PIDF document is
created with both tuples having a status of "closed":

```
<?xml version="1.0" encoding="UTF-8"?>
    <presence xmlns="urn:ietf:params:xml:ns:pidf"
    xmlns:rpid="urn:ietf:params:xml:ns:pidf:rpid"
    entity="sip:presentity@example.com">
        <tuple id="432sd">
           <status>
             <basic>closed</basic>
           </status>
             <rpid:class>IM</rpid:class>
             <contact>im:presentity@example.com</contact>
        </tuple>
        <tuple id="thr76jk">
           <status>
              <basic>closed</basic>
           </status>
           <rpid:class>voice</rpid:class>
           <contact>tel:2224055555@example.com</contact>
        </tuple>
    </presence>
```

A NOTIFY is not sent to the subscriber in this case.

Now, a third PIDF document is created when the IM status changes to
"open":

```
<?xml version="1.0" encoding="UTF-8"?>
    <presence xmlns="urn:ietf:params:xml:ns:pidf"
    xmlns:rpid="urn:ietf:params:xml:ns:pidf:rpid"
    entity="sip:presentity@example.com">
        <tuple id="432sd">
           <status>
              <basic>open</basic>
           </status>
           <rpid:class>IM</rpid:class>
           <contact>im:presentity@example.com</contact>
        </tuple>
        <tuple id="thr76jk">
           <status>
              <basic>closed</basic>
           </status>
```

```
            <rpid:class>voice</rpid:class>
            <contact>tel:2224055555@example.com</contact>
         </tuple>
      </presence>
```

   Notification containing both tuples is sent to the subscriber in this
   case:

```
   NOTIFY sip:watcher@client.example.com SIP/2.0
   Via: SIP/2.0/TCP presence.example.com:5060;branch=z9hG4bKxjfder
   To: <sip:watcher@example.com>;tag:12341111
   From: <sip:presentity@example.com>;tag:232321
   Call-ID: 32432udfidfjmk342
   Cseq: 1 NOTIFY
   Event: Presence
   Subscription-State: active; expires=3599
   Contact: sip:presentity@server.example.com
   Content-Type: application/pidf+xml
   Content-Length: ...

   <?xml version="1.0" encoding="UTF-8"?>
      <presence xmlns="urn:ietf:params:xml:ns:pidf"
      xmlns:rpid="urn:ietf:params:xml:ns:pidf:rpid"
      entity="sip:presentity@example.com">
         <tuple id="432sd">
            <status>
               <basic>closed</basic>
            </status>
            <rpid:class>IM</rpid:class>
            <contact>im:presentity@example.com</contact>
         </tuple>
         <tuple id="thr76jk">
            <status>
               <basic>open</basic>
            </status>
            <rpid:class>voice</rpid:class>
            <contact>tel:2224055555@example.com</contact>
         </tuple>
      </presence>
```

7.2. Watcher Information Specific Examples

   The examples in this section use the winfo template-package with the
   presence event package [10].

   Watcher information to a Presentity:

```
      <?xml version="1.0"?>
        <watcherinfo xmlns="urn:ietf:params:xml:ns:watcherinfo"
      version="0" state="full">
        <watcher-list resource="sip:presentity@example.com"
                    package="presence">
           <watcher status="active"
              id="sr8fdsj"
              duration-subscribed="509"
              expiration="20"
              event="approved">sip:watcherA@example.com"</watcher>
           <watcher status="pending"
              id="sr8fdsj"
              duration-subscribed="501"
              expiration="100"
              event="subscribe">sip:watcherB@example.com"</watcher>
           <watcher status="terminated"
              id="sr8fdsj"
              duration-subscribed="500"
              expiration="0"
              event="rejected">sip:watcherC@example.com"</watcher>
           <watcher status="active"
              id="sr8fdsj"
              duration-subscribed="20"
              expiration="30"
              event="approved">sip:watcherD@example.com"</watcher>
        </watcher-list>
        </watcherinfo>
```

7.2.1. Watcher Subscriber Makes Subscription to Get All the Information
        about Active Watchers

   SUBSCRIBE request from the presentity including the filter:

   SUBSCRIBE sip:presentity@example.com
   Via: SIP/2.0/TCP client.example.com:5060;branch=z9hG4bKxjfdsjfk
   To: <sip:presentity@example.com>
   From: <sip:presentity@example.com>;tag:12341111
   Call-ID: 32432udfidfjmk342
   Cseq: 1 SUBSCRIBE
   Expires: 3600
   Event: Presence.winfo
   Contact: sip:presentity@client.example.com
   Content-Type: application/simple-filter+xml
   Content-Length: ...

   <?xml version="1.0" encoding="UTF-8"?>
   <filter-set xmlns="urn:ietf:params:xml:ns:simple-filter">
     <ns-bindings>
       <ns-binding prefix="wi"
                         urn="urn:ietf:params:xml:ns:watcherinfo"/>
     </ns-bindings>
     <filter id="123" uri="sip:presentity@example.com">
       <what>
         <include>
           /wi:watcherinfo/wi:watcher-list[@package="presence"]/
           wi:watcher[@status="active"]
         </include>
   </what>
   </filter>
   </filter-set>

   Notification to the subscriber:

   NOTIFY sip:presentity@client.example.com SIP/2.0
   Via: SIP/2.0/TCP presence.example.com:5060;branch=z9hG4bKxjfder
   To: sip:presentity@example.com;tag:12341111
   From: sip:presentity@example.com;tag:232321
   Call-ID: 32432udfidfjmk342
   Cseq: 1 NOTIFY
   Contact: sip:presentity@server.example.com
   Event: Presence.winfo

   Content-Type: application/watcherinfo+xml
   Content-Length: ...

```
<?xml version="1.0"?>
  <watcherinfo xmlns="urn:ietf:params:xml:ns:watcherinfo"
version="0" state="full">
  <watcher-list resource="sip:presentity@example.com"
                package="presence">
      <watcher status="active"
         id="sr8fdsj"
         duration-subscribed="509"
         expiration="20"
         event="approved">sip:watcherA@example.com"</watcher>
      <watcher status="active"
         id="sr8fdsj"
         duration-subscribed="20"
         expiration="30"
         event="approved">sip:watcherD@example.com"</watcher>
   </watcher-list>
   </watcherinfo>
```

7.2.2. Watcher Subscriber Requests Information of Watchers with
        Specific Subscription Duration Conditions

   SUBSCRIBE request from the presentity including the filter:

```
   SUBSCRIBE sip:presentity@example.com
   Via: SIP/2.0/TCP client.example.com:5060;branch=z9hG4bKxjfdsjfk
   To: <sip:presentity@example.com>;tag:12341111
   From: <sip:presentity@example.com>
   Call-ID: 32432udfidfjmk342
   Cseq: 1 SUBSCRIBE
   Expires: 0
   Event: Presence.winfo
   Contact: <sip:presentity@client.example.com>
   Content-Type: application/simple-filter+xml
   Content-Length: ...

   <?xml version="1.0" encoding="UTF-8"?>
   <filter-set xmlns="urn:ietf:params:xml:ns:simple-filter">
     <ns-bindings>
       <ns-binding prefix="wi"
                        urn="urn:ietf:params:xml:ns:watcherinfo"/>
     </ns-bindings>
     <filter id="123" uri="sip:presentity@example.com">
       <what>
         <include>
           /wi:watcherinfo/wi:watcher-list[@package="presence"]/
           wi:watcher[@duration-subscribed>500]
         </include>
       </what>
```

```
      </filter>
   </filter-set>

   Notification to the subscriber:

   NOTIFY sip:presentity@client.example.com SIP/2.0
   Via: SIP/2.0/TCP presence.example.com:5060;branch=z9hG4bKxjfder
   To: sip:presentity@example.com;tag:12341111
   From: sip:presentity@example.com;tag:232321
   Call-ID: 32432udfidfjmk342
   Cseq: 1 NOTIFY
   Contact: sip:presentity@server.example.com
   Event: Presence.winfo

   Content-Type: application/watcherinfo+xml
   Content-Length: ...

   <?xml version="1.0"?>
     <watcherinfo xmlns="urn:ietf:params:xml:ns:watcherinfo"
   version="0" state="full">
     <watcher-list resource="sip:presentity@example.com"
                   package="presence">
         <watcher status="active"
            id="sr8fdsj"
            duration-subscribed="509"
            expiration="20"
            event="approved">sip:watcherA@example.com"</watcher>
         <watcher status="pending"
            id="sr8fdsj"
            duration-subscribed="501"
            expiration="100"
            event="subscribe">sip:watcherB@example.com"</watcher>
     </watcher-list>
     </watcherinfo>
```

7.2.3. Watcher Subscriber Requests Specific Watcher Info on Specific
        Triggers

   This filter selects watcher information notifications [9] to be sent
   when the pending subscription status has changed from "pending" to
   "terminated". In the notification, only the watchers that have a
   status of "terminated" and an event of "rejected" are included.

   SUBSCRIBE request from the Watcher Subscriber including the filter:

```
   SUBSCRIBE sip:presentity@example.com
   Via: SIP/2.0/TCP client.example.com:5060;branch=z9hG4bKxjfdsjfk
   To: <sip:presentity@example.com>;tag:12341111
```

```
   From: <sip:presentity@example.com>
   Call-ID: 32432udfidfjmk342
   Cseq: 1 SUBSCRIBE
   Expires: 0
   Event: Presence.winfo
   Contact: <sip:presentity@client.example.com>
   Content-Type: application/simple-filter+xml
   Content-Length: ...

   <?xml version="1.0" encoding="UTF-8"?>
   <filter-set xmlns="urn:ietf:params:xml:ns:simple-winfo-filter">
     <ns-bindings>
       <ns-binding prefix="wi"
                            urn="urn:ietf:params:xml:ns:watcherinfo"/>
     </ns-bindings>
     <filter id="123" uri="sip:presentity@example.com">
       <what>
         <include>
           /wi:watcherinfo/wi:watcher-list[@package="presence"]/
           wi:watcher[@status="terminated" and @event="rejected"]
         </include>
       </what>
       <trigger>
         <changed from="pending"
                                            to="terminated">
           //@status
         </changed>
       </trigger>
     </filter>
   </filter-set>
```

   At some point during the subscription, a second Winfo document is
   created due to some change:

```
    <?xml version="1.0"?>
       <watcherinfo xmlns="urn:ietf:params:xml:ns:watcherinfo"
      version="0" state="full">
       <watcher-list resource="sip:presentity@example.com"
                     package="presence">
           <watcher status="active"
              id="sr8fdsj"
              duration-subscribed="509"
              expiration="20"
              event="approved">sip:watcherA@example.com</watcher>
           <watcher status="terminated"
              id="sr8fdsj"
              duration-subscribed="501"
              expiration="100"
```

```
                event="rejected">sip:watcherB@example.com"</watcher>
            <watcher status="terminated"
               id="sr8fdsj"
               duration-subscribed="500"
               expiration="0"
               event="rejected">sip:watcherC@example.com"</watcher>
            <watcher status="active"
               id="sr8fdsj"
               duration-subscribed="20"
               expiration="30"
               event="approved">sip:watcherD@example.com"</watcher>
       </watcher-list>
        </watcherinfo>
```

   Notification to the subscriber is created, taking into account the
   <trigger> and <what> elements:

```
   NOTIFY sip:presentity@client.example.com SIP/2.0
   Via: SIP/2.0/TCP presence.example.com:5060;branch=z9hG4bKxjfder
   To: sip:presentity@example.com;tag:12341111
   From: sip:presentity@example.com;tag:232321
   Call-ID: 32432udfidfjmk342
   Cseq: 1 NOTIFY
   Contact: sip:presentity@server.example.com
   Event: Presence.winfo

   Content-Type: application/watcherinfo+xml
   Content-Length: ...

   <?xml version="1.0"?>
     <watcherinfo xmlns="urn:ietf:params:xml:ns:watcherinfo"
   version="0" state="full">
     <watcher-list resource="sip:presentity@example.com"
                   package="presence">
         <watcher status="terminated"
            id="sr8fdsj"
            duration-subscribed="501"
            expiration="100"
            event="rejected">sip:watcherB@example.com"</watcher>
         <watcher status="terminated"
            id="sr8fdsj"
            duration-subscribed="500"
            expiration="0"
            event="rejected">sip:watcherC@example.com"</watcher>
     </watcher-list>
     </watcherinfo>
```

8. Security Considerations

   The presence of filters in the body of a SIP message has a
   significant effect on the ways in which the request is handled at a
   server. As a result, it is especially important that messages
   containing this extension be authenticated and authorized.
   Authentication can be achieved using the Digest Authentication
   mechanism described in [2]. The authorisation decision is based on
   the permissions that the resource (notifier) has given to the
   watcher. An example of such auhorisation policy can be found in
   [11].

   Processing of requests and looking up filters requires set operations
   and searches, which can require some amount of computation. This
   enables a DoS attack whereby a user can send requests with
   substantial numbers of messages with large contents, in the hopes of
   overloading the server. To counter this, the server can establish a
   limit on the number of occurrences of the <what>, <changed>, <added>,
   and <removed> elements that are allowed in the filters. A default
   limit of 40 is RECOMMENDED; however, servers may raise or lower the
   limit depending upon their specific engineered capacity.

   Requests can reveal sensitive information about a User Agent's (UA's)
   capabilities. If this information is sensitive, it SHOULD be
   encrypted using SIP S/MIME capabilities [6]. All package-specific
   security measures MUST be followed.

   Propagating filters in SUBSCRIBE requests to foreign domains reveals
   sensitive information about a user's resource lists. It is therefore
   required that an RLS does not forward a filter if that filter is
   addressed to a resource that is under the administrative domain of
   the RLS, but that is not on the resource list. Section 4.1 shows an
   example where such a scenario can occur.

   Note that a filtered document located at a subscriber may project
   false reality. For example, if a subscriber asked to be notified
   when a resource has changed his presence state from "closed" to
   "open" but not from "open" to "closed", then the subscriber may
   afterwards be under the false impression that the resource's presence
   state is "open", even long after the resource has changed it to
   "closed". Therefore, subscribers need to be sure what they put in a
   filter, understand what they asked for, and be prepared to be out of
   sync with the real state of a resource.

9. IANA Considerations

   A new event-reason-value "badfilter" is defined to represent the
   event where the filter is not well formed and/or not accepted. No
   IANA registration is required for this value.

10. Acknowledgements

   The authors would like to thank George Foti, Tim Moran, Sreenivas
   Addagatla, Juha Kalliokulju, Jari Urpalainen, and Mary Barnes for
   their valuable input.

11. References

11.1. Normative References

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

   [2] 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.

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

   [4] Roach, A.B., Campbell, B., and J. Rosenberg, "A Session
        Initiation Protocol (SIP) Event Notification Extension for
        Resource Lists", RFC 4663, September 2006.

   [5] Khartabil, H., Leppanen, E., Lonnfors, M., and J. Costa-Requena,
        "An Extensible Markup Language (XML)-Based Format for Event
        Notification Filtering", RFC 4661, September 2006.

   [6] Ramsdell, B., "Secure/Multipurpose Internet Mail Extensions
        (S/MIME) Version 3.1 Message Specification", RFC 3851, July
        2004.

11.2. Informative References

   [7] Sugano, H., Fujimoto, S., Klyne, G., Bateman, A., Carr, W., and
        J. Peterson, "Presence Information Data Format (PIDF)", RFC
        3863, August 2004.

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

   [9] Rosenberg, J., "An Extensible Markup Language (XML) Based Format
        for Watcher Information", RFC 3858, August 2004.

   [10] Rosenberg, J., "A Watcher Information Event Template-Package for
        the Session Initiation Protocol (SIP)", RFC 3857, August 2004.

   [11] Rosenberg, J., "Presence Authorization Rules", Work in Progress,
        June 2006.

Authors' Addresses

    Hisham Khartabil
    Telio
    P.O. Box 1203 Vika
    Oslo
    Norway

    Phone: +47 2167 3544
    EMail: hisham.khartabil@telio.no

    Eva Leppanen
    Nokia
    P.O BOX 785
    Tampere
    Finland

    Phone: +358 7180 77066
    EMail: eva-maria.leppanen@nokia.com

    Mikko Lonnfors
    Nokia
    P.O BOX 321
    Helsinki
    Finland

    Phone: + 358 71800 8000
    EMail: mikko.lonnfors@nokia.com

    Jose Costa-Requena
    Nokia
    P.O. Box 321
    FIN-00045 NOKIA GROUP
    FINLAND

    Phone: +358 71800 8000
    EMail: jose.costa-requena@nokia.com

Full Copyright Statement

Intellectual Property

   The IETF takes no position regarding the validity or scope of any
   Intellectual Property Rights or other rights that might be claimed to
   pertain to the implementation or use of the technology described in
   this document or the extent to which any license under such rights
   might or might not be available; nor does it represent that it has
   made any independent effort to identify any such rights. Information
   on the procedures with respect to rights in RFC documents can be
   found in BCP 78 and BCP 79.

   Copies of IPR disclosures made to the IETF Secretariat and any
   assurances of licenses to be made available, or the result of an
   attempt made to obtain a general license or permission for the use of
   such proprietary rights by implementers or users of this
   specification can be obtained from the IETF on-line IPR repository at
   http://www.ietf.org/ipr.

   The IETF invites any interested party to bring to its attention any
   copyrights, patents or patent applications, or other proprietary
   rights that may cover technology that may be required to implement
   this standard. Please address the information to the IETF at
   ietf-ipr@ietf.org.

Acknowledgement