A SIP Event Sub-Package for Watcher Information

STATUS OF THIS MEMO

This document is an Internet-Draft and is in full conformance with all provisions of Section 10 of RFC2026.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress".

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt

To view the list Internet-Draft Shadow Directories, see http://www.ietf.org/shadow.html.

Abstract

document defines the watcher information sub-package for the SIP event infrastructure. Watcher information refers to the set of users subscribed to a particular resource within a particular event package. This set changes dynamically as users subscribe, unsubscribe, are approved, or are rejected. A subscriber can subscribe to this information, and therefore learn about changes to it. This event package is a sub-package because it can be applied to any event package, including itself.

1 Introduction

The SIP event infrastructure is described in [1]. It defines a generic framework for subscription to, and notification of, events related to SIP systems. The framework defines the methods SUBSCRIBE

Rosenberg
and NOTIFY, and introduces the notion of a package. A package is a concrete application of the event framework to a particular class of events. Packages have been defined for user presence [2] and for message waiting indications [3], for example.

This draft defines a "sub-package" within the SIP events framework. A sub-package has all the properties of a regular SIP event package. However, it is always associated with some other event package, and can always be applied to any event package, including the sub-package itself.

The sub-package defined here is for watcher information, and is denoted with the token "winfo". For any event package, such as presence, there is a set of users who are subscribed to a particular user within that package. This set of users changes over time as new subscriptions come, old subscriptions expire, and subscriptions are approved or rejected. The set of users subscribed to a particular event package, and the state of their subscriptions, is referred to as watcher information. Since this state is itself dynamic, it is reasonable to subscribe to it in order to learn about changes to it. The watcher info event sub-package is meant to facilitate exactly that - tracking the state of subscriptions to a user in another package.

To denote this sub-package, the name is constructed by appending ".winfo" to the name of whatever package is being tracked. For example, the set of people subscribed to presence is defined by the "presence.winfo" package.

2 Usage Scenarios

There are many useful applications for the watcher information sub-package.

2.1 Presence Authorization

The motivating application for this package is presence authorization. When user A subscribes to the presence of user B, the subscription needs to be authorized. Frequently, that authorization needs to occur through direct user intervention. For that to happen, B’s software needs to become aware that a presence subscription has been requested. This is supported through watcher information. B’s client software would SUBSCRIBE to the watcher information for the presence of B:

```plaintext
SUBSCRIBE sip:B@foo.com SIP/2.0
Via: SIP/2.0/UDP pc34.foo.com
```
From: sip:B@foo.com;tag=123
To: sip:B@foo.com
Call-ID: 9987@1.2.3.4
CSeq: 9887 SUBSCRIBE
Contact: sip:B@pc34.foo.com
Event: presence.winfo

The policy of the server is such that it allows B to subscribe to its own watcher information. So, when A subscribes to B’s presence, B gets a notification of the change in watcher information state:

NOTIFY sip:B@pc34.foo.com SIP/2.0
Via: SIP/2.0/UDP pc34.foo.com
From: sip:B@foo.com;tag=xyz
To: sip:B@foo.com;tag=123
Call-ID: 9987@1.2.3.4
CSeq: 1288 NOTIFY
Contact: sip:B@server.foo.com
Event: presence.winfo
Content-Type: text/xml+winfo
Content-Length: ...

<watcherinfo>
  <presentity uri="sip:B@foo.com">
    <watcher uri="sip:A@foo.com" status="pending"/>
  </presentity>
</watcherinfo>

This indicates to B that A has subscribed, and that the server is awaiting approval. B’s software can alert B that this subscription is awaiting authorization. B can then go to a web page and set policy for that subscriber.

2.2 Blacklist Alerts

Much benefit is derived by allowing applications to subscribe to watcher information, in order to provide value-added features. An example application is blacklist alerts. In this scenario, an application maintains a list of known "bad guys". A user, Joe, signs up for service with the application, presumably by going to a web page and entering in there presence address. When someone attempts to SUBSCRIBE to Joe’s user presence, the application learns of this subscription. It checks the watcher’s URL against the database of
known bad guys. If there is a match, it sends email to Joe letting him know about this.

For this application to work, the application itself subscribes to the presence.winfo for Joe. This allows it to be notified when someone subscribes to Joe’s presence, so that it can run the name against its database.

For this application to work, Joe needs to make sure that the application is allowed to subscribe to his presence.winfo.

3 Package Definition

This section fills in the template that is needed in order to fully specify a SIP event package.

3.1 Package Name

As a sub-package, the watcher information sub-package has many names. Watcher information for any package foo is denoted by the name "foo.winfo". Recursive sub-packaging is explicitly allowed (and useful), so that "foo.winfo.winfo" is a valid package name.

3.2 SUBSCRIBE Bodies

A SUBSCRIBE for a watcher information package MAY contain a body. This body would serve the purpose of filtering the subscription. For example, in the case of presence, the body might indicate that notifications should contain full state every time something changes, and that the time the subscription was first made should not be included in the notifications.

A SUBSCRIBE for a watcher information package MAY be sent without a body. This implies the default subscription filtering policy. The default policy is:

- Notifications are generated every time there is any change in the state of the watcher information.
- Notifications do not normally contain full state; rather, they only indicate the watcher whose state has changed. Full state (the list of all watchers) is sent in the NOTIFY triggered from a SUBSCRIBE.

3.3 Expiration

Watcher information changes as users subscribe to a particular package, or their subscriptions time out. As a result, the state of
watcher information can change very dynamically, depending on the number of subscribers for a particular user in a given package. The rate at which subscriptions time out depends on how long a user maintains its subscription.

As a result of these factors, it is difficult to define a broadly useful default value for the lifetime of a subscription to watcher information. We arbitrarily choose one hour. However, clients SHOULD include in Expires header specifying their preferred duration.

### 3.4 NOTIFY Bodies

The body of the notification contains a watcher information document. This document describes some or all of the set of watchers for a given package, and the state of their subscriptions. All subscribers MUST support the watcherinfo+xml format described in [4], and MUST list its MIME type, application/watcherinfo+xml, in an Accept header present in the SUBSCRIBE request.

Other watcher information formats might be defined in the future. In that case, the subscriptions MAY indicate support for other presence formats. However, they MUST always support and list application/watcherinfo+xml as an allowed format.

Of course, the notifications generated by the server MUST be in one of the formats specified in the Accept header in the SUBSCRIBE request.

### 3.5 Authorization Considerations

The watcher information for a particular package contains sensitive information. Therefore, all subscriptions SHOULD be authenticated and then authorized before approval. Authorization policy is at the discretion of the administrator, as always. However, a few recommendations can be made.

It is RECOMMENDED that subscriptions to the watcher information for some package foo for user A be allowed from user A themselves. That is, it is RECOMMENDED that a user be allowed to subscribe to their own watcher information. This is true recursively, so that we recommend a user be able to subscribe to the watcher information for their watcher information for any package.

It is RECOMMENDED that subscriptions to the watcher information for some package foo for user A be allowed from some other user B if B is an authorized subscriber to A within the package foo. However, it is STRONGLY RECOMMENDED that the notifications sent to B only contain the state of B’s own subscription. In other words, it is RECOMMENDED
that a user be allowed to monitor the state of their own subscription.

To avoid infinite recursion, it is RECOMMENDED that only a user A be allowed to subscribe to foo.winfo.winfo for any foo (such a subscription from A is actually needed to approve the blacklisting application). It is also RECOMMENDED that by default, a server does not support any subscriptions to foo.winfo.winfo.winfo or any other deeper sub-packages.

OPEN-ISSUE: Are we sure about these recommendations? Any cases where this is really useful to allow others to subscribe to foo.winfo.winfo?

3.6 Generation of Notifications

Notifications may be generated for watcher information on package foo, when the subscription state for a user on package foo changes. There is an underlying state machine, described below, which governs the subscription state of a user in any package. Notifications are generated on transitions in this state machine.

3.6.1 The Watcherinfo State Machine

The underlying state machine for a subscription is shown in Figure 1. Initially, there is no state allocated for a subscriber (the init state). When a subscription arrives, the subscription FSM is created. The next state depends on whether policy exists for the subscription. If there is an existing policy that determines that the subscription is forbidden, it moves into the terminated state immediately, where the FSM can be destroyed. If there is existing policy that determines that the subscription is authorized, the FSM moves into the active state. This state indicates that the subscriber will receive notifications.

If, when a subscription arrives, there is no authorization policy in existence, the subscription moves into the pending state. In this state, the server is awaiting an authorization decision. No notifications are generated, but the subscription FSM is maintained. If the authorization decision comes back positive, the subscription is approved, and moves into the active state. If the authorization is negative, the subscription is rejected, and the FSM goes into the terminated state. It is possible that the authorization decision can take a very long time. In fact, no authorization decision may arrive until after the subscription itself expires. If a pending subscription expires, it moves into the waiting state. This state is
similar to pending, in that no notifications are generated. However, if its approved or denied, the FSM is destroyed. The purpose of the state is so that a user can fetch watcherinfo state at any time, and learn of any subscriptions that arrived which require authorization policy. Consider an example. A subscribes to B. B has not defined policy about this subscription, so it moves into pending. B is not "online", so that B’s software agent cannot be contacted to approve the subscription. The subscription expires. Lets say it were destroyed. B logs in, and fetches its watcherinfo state. There is no record of the subscription for A, so no policy decision is made about subscriptions from A. B logs off. A refreshes its subscription. Once
more, the subscription is pending since no policy is defined for it. This process could continue indefinitely. The waiting state ensures that B can find out about this subscription attempt.

This state is also needed to allow for authorization of fetch attempts.

Of course, policy may never be specified for the subscription. As a result, the server can timeout the waiting subscription. The value for this timeout is system dependent. It SHOULD be several times larger than the default expiration time for the package being watched.

If, while in the waiting state, the subscription is refreshed through another SUBSCRIBE, it moves back into the pending state.

At any time, the server can deactivate a subscription. Deactivation implies that the subscription is discarded without a change in authorization policy. This may be done in order to trigger refreshes of subscriptions for a graceful shutdown or subscription migration operation.

3.6.2 Applying the state machine

The server MAY generate a notification to subscribers of watcher information on a transition of the state machine. Whether it does or does not is policy dependent. However, several guidelines are defined.

Consider some event package foo. A subscribes to B for events within that package. A also subscribes to foo.winfo for B. In this scenario (where the subscriber to foo.winfo is also a subscriber to foo for the same user), it is RECOMMENDED that A receive notifications only about the changes in its own subscription. Furthermore, it is RECOMMENDED that state machine transitions be reported only if A’s subscription to B for package foo was approved. That is, the first notification that A should ever receive for foo.winfo is its transition from pending to approved. If A’s subscription to B for foo is later rejected, it is RECOMMENDED that A not receive notifications of this, or any other changes. All of these guidelines are to protect the privacy of B; specifically, so that A cannot know about rejections of its subscriptions to B for some particular package unless explicitly permitted by B.

As a general rule, when a subscriber is authorized to receive notifications about more than one watcher, it is RECOMMENDED that notifications contain information about those watchers which have changed state (and thus triggered a notification), instead of
delivering the current state of every watcher in every notification. However, notifications triggered as a result of a fetch operation (a SUBSCRIBE with Expires of 0) SHOULD result in the full state of all watchers (of course, only those watchers that have been authorized to be divulged to the subscriber) to be present in the NOTIFY.

3.7 Rate Limitations on NOTIFY

For reasons of congestion control, it is important that the rate of notifications not become excessive. As a result, it is RECOMMENDED that the server not generate notifications for a single subscriber at a rate faster than once every 5 seconds.

4 Example Usage

The following section discusses some sample applications and call flows using the watcherinfo package.

4.1 Presence Authorization

In this example, a user Joe, sip:joe@bar.com provides presence through the bar.com presence server. Joe subscribes to his own watcher information, in order to learn about people who subscribe to his presence, in order to approve them. Joe sends the following SUBSCRIBE request:

```
SUBSCRIBE sip:joe@bar.com SIP/2.0
Via: SIP/2.0/UDP pc34.bar.com
From: sip:joe@bar.com;tag=123
To: sip:joe@bar.com
Call-ID: 9987@1.2.3.4
CSeq: 9887 SUBSCRIBE
Contact: sip:joe@pc34.bar.com
Event: presence.winfo
```

The server responds with a 401 to authenticate, and Joe resubmits the SUBSCRIBE with credentials. The server then authorizes the subscription, since it allows Joe to subscribe to his own watcher information for presence. It responds with a 200 OK:

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP pc34.bar.com;received=1.2.3.4
From: sip:joe@bar.com;tag=123
To: sip:joe@bar.com;tag=xyz
```
The server then sends a NOTIFY with the current state of presence.winfo for joe@bar.com:

NOTIFY sip:joe@pc34.bar.com SIP/2.0
Via: SIP/2.0/UDP server19.bar.com
From: sip:joe@bar.com;tag=xyz
To: sip:joe@bar.com;tag=123
Call-ID: 9987@1.2.3.4
CSeq: 1288 NOTIFY
Contact: sip:server19.bar.com
Event: presence.winfo
Content-Type: text/xml+winfo
Content-Length: ...

<watcherinfo>
  <presentity uri="sip:joe@bar.com">
    <watcher uri="sip:A@foo.com" status="pending"/>
  </presentity>
</watcherinfo>

B then responds with a 200 OK to the NOTIFY:

SIP/2.0 200 OK
Via: SIP/2.0/UDP server19.bar.com
From: sip:joe@bar.com;tag=xyz
To: sip:joe@bar.com;tag=123
Call-ID: 9987@1.2.3.4
CSeq: 1288 NOTIFY

The NOTIFY tells B that A currently has a pending subscription. B goes to a web page, and enters form data which tells the server to allow As subscription. This causes a change in the status of the subscription (which moves from pending to active), and the delivery of another notification:
NOTIFY sip:joe@pc34.bar.com SIP/2.0
Via: SIP/2.0/UDP server19.bar.com
From: sip:joe@bar.com;tag=xyz
To: sip:joe@bar.com;tag=123
Call-ID: 998701.2.3.4
CSeq: 1289 NOTIFY
Contact: sip:server19.bar.com
Event: presence.winfo
Content-Type: text/xml+winfo
Content-Length: ...

<watcherinfo>
  <presence uri="sip:joe@bar.com">
    <watcher uri="sip:A@foo.com" status="active" event="approved" />
  </presence>
</watcherinfo>

B then responds with a 200 OK to the NOTIFY:

SIP/2.0 200 OK
Via: SIP/2.0/UDP server19.bar.com
From: sip:joe@bar.com;tag=xyz
To: sip:joe@bar.com;tag=123
Call-ID: 998701.2.3.4
CSeq: 1289 NOTIFY

5 Security Considerations

Watcher information is very sensitive information. As such, authentication and authorization of subscriptions to this package are critical, and are described in detail throughout this document.

6 Authors Addresses

Jonathan Rosenberg
dynamicsoft
72 Eagle Rock Avenue
First Floor
East Hanover, NJ 07936
email: jdrosen@dynamicsoft.com
7 Bibliography


