Network Working Group
Internet-Draft
Expires: April 25, 2003

A. B. Roach
J. Rosenberg
B. Campbell
dynamicsoft

October 25, 2002

A Session Initiation Protocol (SIP) Event Template Package for Collections
draft-roach-sip-list-template-00

Abstract

This document presents a Session Initiation Protocol (SIP) event package template for subscribing to a collection of event packages. Instead of the subscriber sending a SUBSCRIBE to each notifier individually, the subscriber can subscribe to their an entire collection, and then receive notifications when the state of any of the notifiers in the collection changes.

Table of Contents

1. Introduction

The SIP-specific event notification mechanism [2] allows a user (the subscriber) to request to be notified of changes in the state of a particular resource. This is accomplished by having the subscriber generate a SUBSCRIBE request for the resource, which is processed by a notifier that represents the resource. In many cases, a subscriber has a collection of resources they are interested in.

For environments where bandwidth is limited, such as a wireless network, subscribing to each resource individually is problematic. The specific problems are:

- It generates substantial message traffic, in the form of the initial SUBSCRIBE requests for each resource, and the refreshes of each individual subscription.

- The notifier may insist on low refresh intervals, in order to avoid long lived subscription state. This means that the subscriber may need to generate subscriptions faster than it would like to, or has the capacity to.

- The notifier may generate NOTIFY requests more rapidly than the subscriber desires, causing NOTIFY traffic at a greater volume than is desired by the subscriber.

- If a subscriber has only intermittent connectivity, and generally polls for state rather than simply subscribing, the latency to obtain the state of the entire resource can be large. The messaging required for each poll can also be substantial.

To solve these problems, this specification defines a template event package for collections of resources. A resource list is identified by a SIP URI [1], and it represents a list of zero or more URIs. Each of these URIs is the Request URI for an individual resource for which the subscriber wants to receive information.

The notifier for the collection is called an "resource list server", or RLS. In order to determine the state of the entire list, the RLS will typically generate a subscription to each element of the list.

The resource list may exist within the domain of the subscriber, but it can also exist within a third party domain.

The first section provides more detail on the operation of the RLS, and the second section defines the event package for resource list subscriptions.

2. Overview of Operation

This section provides an overview of the typical mode of operation of this event template package. It is not normative.

When a user wishes to subscribe to the resource of a list of resources, they create a resource list. This resource list is represented by a SIP URI. The list contains a set of URIs, each of which represents a resource for which the subscriber wants to receive information. The resource list can exist in any domain. Typically, the user who creates the list (and subsequently subscribes to it) will have a trust relationship with the domain that hosts the list. The specific means by which the list is created and maintained is outside of the scope of this specification. The list could be manipulated through a web page, through a voice response system, or through some protocol.

To learn the resource state of the set of elements on the list, the user sends a single SUBSCRIBE request targeted to the URI of the list. This will be routed to an RLS for that URI. The RLS acts as a notifier, authenticates the subscriber, and accepts the subscription.

The RLS may have direct information about some or all of the resources specified by the list. If it does not, it subscribes to the resource specified by the request URI, for the base event package to which ".list" has been appended.

Since the RLS is acting on behalf of the user, it will provide the identity of the user in the From field. If the resources require credentials in order to accept the subscription, the user will have had to provide them to the RLS ahead of time. This requires a trust relationship between the user and RLS.

As notifications arrive from individual resources, the RLS accepts them, extracts the resource information, and generates a notification to the subscriber. The RLS can, at its discretion, buffer notifications that it receives, and send the resource information to the subscriber in batches, rather than individually. This allows the RLS to provide rate limiting for the subscriber.

```
            Joe               RLS              User A            User B
             |                 |                  |                 |
             |(1) SUBSCRIBE    |                  |                 |
             | Event: foo.list |                  |                 |
             |---------------->|                  |                 |
             |(2) 200 OK       |                  |                 |
```

```
             |<----------------|                 |                 |
             |(3) NOTIFY       |                 |                 |
             | Event: foo.list |                 |                 |
             |<----------------|                 |                 |
             |(4) 200 OK       |                 |                 |
             |---------------->|                 |                 |
             |                 |(5) SUBSCRIBE a  |                 |
             |                 | Event: foo      |                 |
             |                 |---------------->|                 |
             |                 |(6) SUBSCRIBE b  |                 |
             |                 | Event: foo      |                 |
             |                 |---------------------------------->|
             |                 |(7) 200 OK       |                 |
             |                 |<----------------|                 |
             |                 |(8) 200 OK       |                 |
             |                 |<----------------------------------|
             |                 |(9) NOTIFY       |                 |
             |                 | Event: foo      |                 |
             |                 |<----------------|                 |
             |                 |(10) 200 OK      |                 |
             |                 |---------------->|                 |
             |(11) NOTIFY      |                 |                 |
             | Event: foo.list |                 |                 |
             | a's state       |                 |                 |
             |<----------------|                 |                 |
             |(12) 200 OK      |                 |                 |
             |---------------->|                 |                 |
             |                 |(13) NOTIFY      |                 |
             |                 | Event: foo      |                 |
             |                 |<----------------------------------|
             |                 |(14) 200 OK      |                 |
             |                 |---------------------------------->|
             |(15) NOTIFY      |                 |                 |
             | Event: foo.list |                 |                 |
             | b's state       |                 |                 |
             |<----------------|                 |                 |
             |(16) 200 OK      |                 |                 |
             |---------------->|                 |                 |
```

As an example, consider a resource list with two resources, sip:userA@a.com and sip:userB@b.com. A typical flow for a subscription to this resource list is shown in Figure 1.

2.1 Recursive Application of the "list" Template

As described in [2], one of the key features of any template package is that it can be applied to any other defined package -- including other templates, and even itself. For many templates, the arbitrary

recursive application of the template to itself may be of questionable value. For the "list" template, however, there is significant utility that can be provided in this fashion. Take the example of applying "list" to the "presence" event package [5]. A user may quite reasonably maintain several lists of users for whom they want to know presence information.

TBD: Addtional motivating text here.

3. Template Event Package for "list"

The following subsections formally define the resource list event package, following the requirements defined by the SIP events framework [2].

3.1 Event Package Name

The name of this event template package is "list".

The following is the information needed to register this event package with IANA:

Package Name: list

Type: template

Contact: Jonathan Rosenberg, jdrosen@dynamicsoft.com

Reference: RFC XXXX [[Note to RFC Editor: replace with the RFC number for this specification]]

3.2 Event Package Parameters

This specification does not define any parameters in the Event header for this package. Any parameters that are present on the Event header, however, are propigated to any SUBSCRIBE messages generated for the base package to which it has been applied.

3.3 SUBSCRIBE Bodies

The SUBSCRIBE message MAY contain a body whose purpose is to define filters on the operation of the list. These filters would include any rate limitation desired for the notifications, or any aggregation that is desired. There is no default or mandatory body type defined for this purpose.

3.4 Subscription Duration

Since the primary benefit of the resource list server is to reduce the overall messaging volume to a handset, it is RECOMMENDED that the subscription duration to a list be reasonably long. The default, when no duration is specified, is two hours, which reduces the need to refresh too frequently. Of course, the standard techniques [2] can be used to increase or reduce this amount.

3.5 NOTIFY Bodies

An implementation compliant to this specification MUST support the multipart/mixed type. This allows a notification to contain multiple resource documents.

The absence of an Accept header in the SUBSCRIBE indicates support for multipart/mixed and the content type(s) used by the base package. If an Accept header is present, these types MUST be included, in additional to any other types supported by the client.

3.6 Notifier Processing of SUBSCRIBE Requests

All subscriptions for resourcce lists SHOULD be authenticated. The use of the SIP HTTP Digest mechanism [1] over TLS is RECOMMENDED.

Once authenticated, the subscription is authorized. Typically, each resource list is associated with a particular user (the one who created it and manages the set of elements in it), and only that user will be allowed to subscribe. Of course, there may be exceptions to this rule, and a notifier MAY use any authorization policy it chooses.

3.7 Notifier Generation of NOTIFY Requests

This specification leaves the choice about how and when to generate NOTIFY requests at the discretion of the implementor. One of the value propositions of the RLS is the means by which it aggregates, rate limits, or optimizes the way in which notifications are generated.

As a baseline behavior, if the RLS acts as a subscriber to determine the state of the resources on the resource list, it MAY generate a NOTIFY to the RLS subscriber whenever it receives a NOTIFY about a state change in one or more resources. The body of the NOTIFY would merely be copied from that NOTIFY into the NOTIFY sent by the RLS to the subscriber.

If a subscription to a resource is terminated by the notifier and the RLS is unable to re-establish the subscription by the recovery mechanisms described in SIP Events [2], that resource is still present in resource list NOTIFY messages as appropriate. The "Subscription-State" body-header is set to "terminated", and the "reason" parameter is copied from the NOTIFY received from the resource.

If a SUBSCRIBE to a resource is refused with a response code that cannot be recovered from, that resource is still present in resource

list NOTIFY messages as appropriate. The resource will be reported with a a "Subscription-State" value of "terminated," and a "reason" parmeter of "rejected".

When the first SUBSCRIBE message for a particular subscription is received by a resource list notifier, the notifier will often not know state information for all of the resources specified by the resource list. The NOTIFY message triggered from the initial SUBSCRIBE message will contain a multipart/mixed message, with one section for each resource in the list. Sections corrsponding to resources for which no state information is yet available will contain zero-byte bodies. The Resource-URI header will be populated, and the Subscription-State header will be set to "unknown".

Sections corresponding to resources for which the resource list notifier does have state SHOULD be populated with correct data (subject, of course, to local policy decisions). This will often occur if the resource list server is colocated with the server for one or more of the resources specified on the list.

Immediate notifications triggered as a result of subsequent SUBSCRIBE messages SHOULD result in the full state of all resources to be present in the NOTIFY. This allows the subscriber to refresh their state, and to recover from lost notifications.

Note that a consequence of the way in which resource list subscriptions work, polling of resource state will often not be particularly useful. While such polls will retrieve the resource list (and potentially even some of the states if a resource on the list is colocated with the resource list server), they will often not contain state for some or all of the resources on the list.

3.8 Subscriber Processing of NOTIFY Requests

The SIP Events framework expects packages to specify how a subscriber processes NOTIFY requests in any package specific ways, and in particular, how it uses the NOTIFY requests to contruct a coherent view of the state of the subscribed resource.

Notifications within this package can convey partial information; that is, they can indicate information about a subset of the state associated with the subscription. This means that an explicit algorithm needs to be defined in order to construct coherent and consistent state.

For this template package, each section of the multipart/mixed document contains a URI which uniquely identifies the resource to which that section corresponds. When a NOTIFY arrives, the recipient

of the NOTIFY updates information for each identified resource. Information for any resources that are not identified in the NOTIFY are not changed, even if they were indicated in previous NOTIFY mesages. See section Section 4.3 for more information.

Note that the package to which the ".list" template has been applied may, in turn, have rules for compositing partial state notification. When processing data related to those packages, their rules apply (i.e. the fact that they were reported as part of a collection does not change their partial notification semantics).

3.9 Handling of Forked Requests

Forking makes little sense with this event package, since the whole idea is a centralization of the source of notifications. Therefore, a subscriber MUST create just a single dialog as a result of a single subscription request, using the techniques described in [2].

3.10 Rate of Notifications

One potential role of the RLS is to perform rate limitations on behalf of the subscriber. As such, this specification does not mandate any particular rate limitation, and rather leaves that to the discretion of the implementation.

3.11 State Agents

Effectively, a resource list server is nothing more than a state agent for the resource event type. A separate event package is needed because of the differing body types which can be used in NOTIFY, and the need to construct complete state from the partial notifications. Furthermore, there are differing values of the subscription interval, differing recommendations on rate limitation, and so on.

The usage of the RLS does introduce some security considerations. The end user is no longer the direct subscriber to the state of the resource. If the notifier for the resource demands end-to-end authentication, the RLS will need to be provided appropriate credentials to access those resources (e.g. shared secrets for Digest authentication). This requires a certain level of trust between the user and their RLS. This specification does not describe any particular means of providing such credentials to the RLS (such as uploading a shared secret). However, any such upload mechanism MUST ensure privacy of the key data; using HTTPS to fill out a form is a reasonable method.

If the notifier for the resource is using a transitive trust model to

validate the subscriber, then this works well with the RLS concept. The RLS would authenticate the subscriber, and then MAY use the SIP extensions for network asserted identity [3][4] to provide an authenticated identity to the PA.

4. Using multipart/mixed to Convey Aggregate State

In order to convey the state of multiple resources, the list template package uses the "multipart/mixed" mime type. The syntax for multipart/mixed is defined in MIME Part 1 [6].

Because the document itself must contain several pieces of information that aren't conveyed by default, multipart/mixed bodies used for delivering collections of state have a few additional requirements beyond those describe in MIME.

4.1 Preamble Headers

The preamble section of a multipart/mixed body used in a list notification MUST contain two headers. Note that these headers follow the same syntax rules as defined for headers in SIP [1], with the distinction that the list of headers is not required to end with CRLF.

The first mandatory preamble header is "Version", which contains a number from 0 to 2^32-1. This version number MUST be 0 for the first NOTIFY message sent within a subscription (typically in response to a SUBSCRIBE request), and MUST increase by exactly one for each subsequent NOTIFY sent within a subscription.

The second mandatory preamble header is "State". The "State" header indicates whether the NOTIFY message contains one section for each resource in the collection. If it does, the value of the header is "full"; otherwise, it is "partial". Note that the first NOTIFY sent in a subscription MUST contain full state, as must the first NOTIFY sent after receipt of a SUBSCRIBE request for the subscription.

4.2 Body-Part Headers

Each body part of mime/multipart documents contains zero or more headers that convey information about the contents of that section. When used in list templates, each body part also MUST contain two addtional headers.

The first mandatory body-part header is "Resource-URI". This header contains the URI that uniquely identifies the resource whose state is contained in the body part. Note that "Resource-URI" might also contain a name-addr syntax; this can be used to convey a human-readable description of the resource that is identified by the URI in the "Resource-URI" header (if the RLS has such a description).

The second mandatory body-part header is "Subscription-State". This header contains the "Subscription-State" for the individual resource

defined in this body-part. The syntax and meaning for this header are defined in SIP Events [2]. In a resource list, however, the "expires" and "retry-after" parameters have no semantics associated with them, and the "reason" parameter is included for informational purposes only. That is, the subscriber to a resource list does not take action based on the "reason" parameter in a body-part "Subscrtiption-State" header.

In the simplest case (i.e. the RLS issues literal SUBSCRIBE requests and receives literal NOTIFY requests), an RLS can copy the "Subscription-State" header from the NOTIFY received for that resource into the body-part headers. The RLS MAY remove the "expires" and "retry-after" parameters from the "Subscription-State" header when it does so.

4.3 Constructing Coherent Resource State

The resource list subscriber maintains a table for each resource list. The table contains a row for each resource in the resource list. Each row is indexed by the URI for that resource. That URI is obtained from the "Resource-URI" body-part header. The contents of each row contain the state of that resource as conveyed in the resource document.

For resources that provide versioning information (which is mandated by [2] for any formats that allow partial notification), each row also contains a version number. The version number of the row is initialized with the version specified in the first document received, as defined by the corrsponding event package.

Each time a new document for a resource is received, the value of the local version number is compared to the version number of the new document. If the value in the new document is one higher than the local version number, the local version number is increased by one, and the document is processed. If the value in the document is more than one higher than the local version number, the local version number is set to the value in the new document, the document is processed, and the .list subscriber SHOULD generate a refresh request to trigger a full state notification. If the value in the document is less than or equal to the local version, the document is discarded without processing.

The processing of the resource list notification depends on whether it contains full or partial state. If it contains full state, indicated by the value of the preable header "State", the contents of the resource-list table are flushed. They are repopulated from the document. A new row in the table is created for each "resource" element.

If the resource list document contains partial state, as indicated by the value of the preable header "State", the document is used to update the table. For each resource listed in the document, the subscriber checks to see whether a row exists for that resource. This check is done by comparing the Resource-URI value with the URI associated with the row. If the resource doesn't exist in the table, a row is added, and its state is set to the information from that "resource" element. If the resource does exist, its state is updated to be the information from that "resource" element. If a row is updated or created such that its state is now "terminated," that entry MAY be removed from the table at any time.

4.4 Syntax

This section uses ABNF to define the additional syntactic elements required by this document. It uses elements from the base SIP specification [1], the SIP Events document [2], and MIME [6].

```
Preamble-Headers =  ( Version CRLF State *CRLF ) /
                    ( State CRLF Version *CRLF )

Version          =  "Version" HCOLON 1*DIGIT

State            =  "State" HCOLON ( "full" / "partial" )


Bodypart-Headers =  * (    Resource-URI
                       / Subscription-State
                       / content
                       / description
                       / encoding
                       / id
                      ) CRLF

Resource-URI     =  "Resource-URI" HCOLON ( name-addr / addr-spec )
```

Further, this document defines an additional substate-value of "unknown" for the "Subscription-State" header.

4.5 Examples

TBD: Add examples.

5. Security Considerations

This package does introduce some security considerations, which are discussed in Section Section 3.11.

6. IANA Considerations

This document defines a new Template Event Package, as described in [2]. The information necessary to register this Template Event Package is given in section Section 3.1.

OPEN ISSUE: Do we need to register the headers we define for the preamble and body parts? It's not clear where we would do so. Do we need to create a new registry?

7. Open Issues

- Technically, MIME [6] talks about multipart/mixed representing ordered body parts, while multipart/parallel is used for unordered body parts. The delivery of state of a collection of resources seems unordered; should we be using multipart/parallel instead of multipart/mixed?

- Technically, MIME [6] says that the preamble portion of multitype bodies is ignored. However, this definition of multitype bodies was aimed more at mail delivery than use in other contexts. We have the need to convey body-level information that applies to all parts of a multitype document, and have elected to use the preamble for this purpose. Are there any problems with doing so?

- Similaraly, MIME [6] also says that any headers appearing in body parts other than Content-* cannot be depended on. The rationale for this restriction, however, is that intervening mail gateways may discard such headers. Of course, this reasoning does not have any chance of applying to the use described in this document, so we reason that such headers *can* be relied upon in this context. Are there any flaws in this reasoning?

- Do the new headers require any additional IANA action? (section Section 6)

References

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

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

[3] Watson, M., Peterson, J. and C. Jennings, "Private Extensions to the Session Initiation Protocol (SIP) for Asserted Identity within Trusted Networks", draft-ietf-sip-asserted-identity-02 (work in progress), August 2002.

[4] Peterson, J., "Enhancements for Authenticated Identity Management in the Session Initiation Protocol (SIP)", draft-peterson-sip-identity-01 (work in progress), July 2002.

[5] Rosenberg, J., "Session Initiation Protocol (SIP) Extensions for Presence", draft-ietf-simple-presence-07 (work in progress), May 2002.

[6] Borenstein, N. and N. Freed, "MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies", RFC 1521, September 1993.

Authors' Addresses

Adam Roach
dynamicsoft
5100 Tennyson Pkwy
Suite 1200
Plano, TX 75024
US

EMail: adam@dynamicsoft.com

Jonathan Rosenberg
dynamicsoft
72 Eagle Rock Ave.
First Floor
East Hanover, NJ 07936
US

EMail: jdrosen@dynamicsoft.com

Ben Campbell
dynamicsoft
5100 Tennyson Pkwy
Suite 1200
Plano, TX 75024
US

EMail: bcampbell@dynamicsoft.com

Full Copyright Statement

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.