Internet Engineering Task Force   SIMPLE WG
Internet Draft

J.Rosenberg
dynamicsoft
B.Campbell
dynamicsoft

draft-ietf-simple-presencelist-package-00.txt
June 24, 2002
Expires: December 2002

A SIP Event Package for List Presence

Abstract

This document presents a SIP event package for subscribing to a list of presentities. Instead of the subscriber sending a SUBSCRIBE to each presentity individually, the subscriber can subscribe to their presence list as a whole, and then receive notifications when the state of any of the presentities on the list changes.

Table of Contents

1 Introduction

The SIP for presence specification [1] allows a user (the subscriber) to request to be notified of changes in the presence state of a particular user (the presentity) [12]. This is accomplished by having the subscriber generate a SUBSCRIBE request for the presentity, which is processed at a presence agent in the domain of the presentity. Typically, a subscriber has a collection of presentities they are interested in. This collection is called a "presence list", and typically has anywhere from a few to even a hundred members.

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

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

- The presence agent 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 presence agent 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 presence rather than simply subscribing, the latency to obtain the presence state of the entire presence list can be large. The messaging required for each poll can also be substantial.

To solve these problems, this specification defines a presence list event package. A presence list is identified by a SIP URI [2], and it represents a list of zero or more URIs. Each of those URIs, in turn, is either a presentity, or another presence list. The state of the presence list is the presence state of the list of presentities at the leaves of the tree defined by the URI. As a result, instead of subscribing to each presentity, a subscriber can subscribe to the presence list, and obtain the same information.

The notifier for the presence list package is called a "presence list server", or PLS. In order to determine the state of the entire list, the PLS will typically generate a presence list or presence subscription to each element of the list.

The presence 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 PLS, and the second section defines the event package for presence list subscriptions.

2 Overview of Operation

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

When a user wishes to subscribe to the presence of a list of presentities, they create a presence list. This presence list is represented by a SIP URI. The list contains a set of URIs, each of which is either another list or a presentity. The presence list can exist at 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 presence 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 a PLS for that URI. The PLS acts as a notifier, authenticates the subscriber, and accepts the subscription. In order to provide the subscriber with the presence state of the presentities on the list, the PLS itself will subscribe to each element on the list, using the presence list event package. If the subscription is rejected because that package is not supported, the list element is a presentity, and not another list, and it can therefore fall back to a regular presence subscription. Since the PLS is acting on behalf of the user, it will provide the identity of the user in the From field. If the presentities require credentials in order to accept the subscription, the user will have had to provide them to the PLS ahead of time. This requires a trust relationship between the user and PLS.

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

As an example, consider a presence list with two presentities,

```
         Joe              PLS              User A           User B
          |                 |                 |                 |
          |(1) SUBSCRIBE    |                 |                 |
          | list            |                 |                 |
          |---------------->|                 |                 |
          |(2) 200 OK       |                 |                 |
          |<----------------|                 |                 |
          |(3) NOTIFY       |                 |                 |
          |<----------------|                 |                 |
          |(4) 200 OK       |                 |                 |
          |---------------->|                 |                 |
          |                 |(5) SUBSCRIBE a  |                 |
          |                 |---------------->|                 |
          |                 |(6) SUBSCRIBE b  |                 |
          |                 |---------------------------------->|
          |                 |(7) 200 OK       |                 |
          |                 |<----------------|                 |
          |                 |(8) 200 OK       |                 |
          |                 |<----------------------------------|
          |                 |(9) NOTIFY       |                 |
          |                 |<----------------|                 |
          |                 |(10) 200 OK      |                 |
          |                 |---------------->|                 |
          |(11) NOTIFY      |                 |                 |
          | a's state       |                 |                 |
          |<----------------|                 |                 |
          |(12) 200 OK      |                 |                 |
          |---------------->|                 |                 |
          |                 |(13) NOTIFY      |                 |
          |                 |<----------------------------------|
          |                 |(14) 200 OK      |                 |
          |                 |---------------------------------->|
          |(15) NOTIFY      |                 |                 |
          | b's state       |                 |                 |
          |<----------------|                 |                 |
          |(16) 200 OK      |                 |                 |
          |---------------->|                 |                 |
```

Figure 1: Typical Package Usage

sip:userA@a.com and sip:userB@b.com. A typical flow for a subscription to this presence list is shown in Figure 1.

3 Event Package for "presencelist"

The following subsections formally define the presence list event package, following the requirements defined by the SIP events framework [3].

3.1 Event Package Name

The name of this event package is "presencelist".

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

Package Name: presencelist

Type: package

Contact: Jonathan Rosenberg, jdrosen@dynamicsoft.com

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

OPEN ISSUE: We could potentially make this a template package. In template form, it would represent a "collection" template, that allows you to subscribe to a list of -something-, where -something- has an event package. In the case of presence, the subscription would be for presence.collection.

3.2 Event Package Parameters

This specification does not define any parameters in the Event header for this package.

3.3 SUBSCRIBE Bodies

The SUBSCRIBE message MAY contain a body whose purpose is to define filters on the operation of the buddylist. 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 buddy list server is to reduce the overall messaging volume to a handset, it is RECOMMENDED that the subscription duration to a buddylist be reasonably long. The default, when no duration is specified, is two hours. That reduces the need to refresh too frequently. Of course, the standard techniques [3] can be

used to increase or reduce this amount.

3.5 NOTIFY Bodies

There are two mandatory-to-implement body types for this package.

The first mandatory-to-implement body type in notifications is application/cpim-plidf+xml. This type is defined in Section 4 of this specification. It is almost identical to the presence data format [4], but allows for lists of presenties, rather than a single one. All implementors of this package MUST support this type.

The second mandatory-to-implement body type in notifications is application/cpim-pidf+xml [4]. The PIDF format only supports information for a single presentity. Therefore, its usage is limited to notifications that report a change in state for a single presentity. It is mandated in order to facilitate operation of the PLS. The PLS can simply pass on any presence documents it receives from the presentities in a notification, without modification.

An implementation compliant to this specification MAY support the multipart/mixed type. As described in [4], this allows a notification to contain multiple presence documents. This type, like application/cpim-pidf+xml, can only be used in notifications that report changes in state, not full state. This is described in more detail in Section 4.1.

> OPEN ISSUE: It seems like there are two many choices here. Eliminating pidf and multipart/mixed will require PLS to generate their own documents, which would result in the loss of end-to-end signatures. The other alternative is to eliminate plidf, which is needed for partial state updates.

The absence of an Accept header in the SUBSCRIBE indicates support for both application/cpim-pidf+xml and application/cpim-plidf+xml. If an Accept header is present, both 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 buddy lists SHOULD be authenticated. The use of the SIP HTTP Digest mechanism [2] over TLS is RECOMMENDED.

Once authenticated, the subscription is authorized. Typically, each presence 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 PLS is the means by which it aggregates, rate limits, or optimizes the way in which notifications are generated.

As a baseline behavior, if the PLS acts as a subscriber to determine the state of the presentities on the buddy list, it MAY generate a NOTIFY to the PLS subscriber whenever it receives a NOTIFY about a state change in one or more presentities. The body of the NOTIFY, assuming it's application/cpim-pidf+xml, would merely be copied from that NOTIFY into the NOTIFY sent by the PLS to the subscriber. If a subscription to a presentity is refused, the BLSS MAY generate a new presence document for that presentity, setting its status to "subscription refused", and then pass that NOTIFY to the subscriber. This would give the subscriber a visual clue that its subscription was refused, and that the presentity should probably be removed from the buddy list.

OPEN ISSUE: This seems insufficient. There should probably be an explicit way to indicate that the subscription to a specific presentity has been refused. More generally, there should be an explicit way to pass information on the subscription states to particular presentities. Perhaps an additional element in application/cpim-plidf+xml.

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

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. The details of this mechanism are specific to the particular document type. See Section 4.1 for information on constructing coherent information from an application/cpim-plidf+xml document. If the document received is an application/cpim-pidf+xml document, the procedures of Section 4.1 are followed, as if it was a plidf document, with a version one higher than the current version, and a state of partial.

OPEN ISSUE: This implies that version numbers of the documents are shared across pidf and plidf. Is that what we want?

If the document received is of type multipart/mixed, the procedures of Section 4.1 are followed, as if it was an plidf document, with a version one higher than the previous version, and a state of partial.

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 [3].

3.10 Rate of Notifications

One potential role of the PLS 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 presence list server is nothing more than a state agent for the presence 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 PLS does introduce some security considerations. The end user is no longer the direct subscriber to the presence state of the presentity. If the PA for the presentity demands end-to-end authentication, the PLS will need to be provided the shared secrets for those presentities (assuming Digest is used). This requires a certain level of trust between the user and their PLS. This specification does not describe any particular means of uploading the shared secret for a particular subscriber to the PLS. However, that

upload mechanism MUST ensure privacy of the key data; using HTTPS to fill out a form is a reasonable method.

If the PA for the presentity is using a transitive trust to validate the subscriber, then this works well with the PLS concept. The PLS would authenticate the subscriber, and then MAY use the SIP extensions for network asserted identity [5] [6] to provide an authenticated identity to the PA.

4 Presence List Information Data Format

Presence list information is an XML document [7] that MUST be well-formed and SHOULD be valid. Presence list documents MUST be based on XML 1.0 and MUST be encoded using UTF-8. This specification makes use of XML namespaces for identifying presence list documents and document fragments. The namespace URI for elements defined by this specification is a URN [8], using the namespace identifier 'ietf' defined by [9] and extended by [10]. This URN is:

```
urn:ietf:params:xml:ns:cpim-plidf
```

A presence list information document begins with the root element tag "presence-list". It consists of any number of "presence" sub-elements, each of which is describes a particular presentity. The "presence" element is defined in [4]. Elements from different namespaces MAY be present for the purposes of extensibility; elements or attributes from unknown namespaces MUST be ignored. There are three attributes associated with the "presence-list" element, all of which MUST be present:

version: This attribute allows the recipient of presence list documents to properly order them. Versions start at 0, and increment by one for each new document sent to a subscriber. Versions are scoped within a subscription. Versions MUST be representable using a 32 bit integer.

state: This attribute indicates whether the document contains the full presence list state, or whether it contains only information on those presentities which have changed since the previous document (partial).

entity: This attribute contains a URI that identifies the presence list whose information is reported in the remainder of the document.

4.1 Constructing Coherent Presence State

The presence list subscriber maintains a table for each presence list. The table contains a row for each presentity in the presence list. Each row is indexed by the URI for that presentity. That URI is obtained from the entity attribute of the "presence" element. The contents of each row contain the state of that presentity as conveyed in the presence document. The table is also associated with a version number. The version number MUST be initialized with the value of the "version" attribute from the "presence-list" element in the first document received. Each time a new document is received, the value of the local version number, and the "version" attribute in the new document, are compared. 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 watcherinfo subscriber SHOULD generate a refresh request to trigger a full state notification. If the value in the document is less than the local version, the document is discarded without processing.

The processing of the presence list document depends on whether it contains full or partial state. If it contains full state, indicated by the value of the "state" attribute in the "presence-list" element, the contents of the presence-list table are flushed. They are repopulated from the document. A new row in the table is created for each "presence" element. If the presence list document contains partial state, as indicated by the value of the "state" attribute in the "presence-list" element, the document is used to update the table. For each "presence" element in the document, the subscriber checks to see whether a row exists for that presentity. This check is done by comparing the URI in the "entity" attribute of the "presence" element with the URI associated with the row. If the presentity doesn't exist in the table, a row is added, and its state is set to the information from that "presence" element. If the presentity does exist, its state is updated to be the information from that "presence" 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.

OPEN ISSUE: There is currently nothing that would indicate that the state of a presentity is "terminated"; this doesn't mean the user has been terminated (!), but rather, that the subscription from the PLS to the presentity has been terminated.

4.2 Example

The following is an example presence list document:

```
<?xml version="1.0"?>
<list:presence-list entity="sip:myfriends@example.com"
               version="1"
               state="full"
               xmlns:list="urn:ietf:params:xml:ns:cpim-plidf"
               xmlns:impp="urn:ietf:params:xml:ns:cpim-pidf">
  <impp:presence entity="sip:someone@example.com">
     <impp:tuple id="mobile-phone">
       <impp:status>
         <impp:basic>open</impp:basic>
       </impp:status>
       <impp:contact priority="0.8">tel:09012345678</impp:contact>
     </impp:tuple>
   </impp:presence>
</list:presence-list>
```

4.3 XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           targetNamespace="urn:ietf:params:xml:ns:cpim-plidf"
           xmlns:tns="urn:ietf:params:xml:ns:cpim-plidf">
<xs:import namespace="urn:ietf:params:xml:ns:cpim-pidf"
           schemaLocation="TBD"/>
  <xs:element name="presence-list">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="cpim-pidf:presence" minOccurs="0"
                    maxOccurs="unbounded"/>
        <xs:any namespace="##other" processContents="lax" minOccurs="0"
                maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="version" type="xs:nonNegativeInteger"
                    use="required"/>
      <xs:attribute name="state" use="required">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="full"/>
            <xs:enumeration value="partial"/>
          </xs:restriction>
```

```
          </xs:simpleType>
        </xs:attribute>
        <xs:attribute name="entity" type="xs:anyURI" use="required" />
      </xs:complexType>
    </xs:element>
  </xs:schema>
```

OPEN ISSUE: In order to properly import the PIDF schema, it needs to have a well defined location. This presumably requires PIDF to have registered its schema with IANA, but it currently does not.

5 Security Considerations

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

OPEN ISSUE: Need to discuss the security issues with the choice of document formats; i.e., multipart/mixed vs. application/cpim-plidf+xml and their impact on end-to-end security.

6 IANA Considerations

This document registers a new MIME type, application/cpim-plidf+xml and registers a new XML namespace.

6.1 application/cpim-plidf+xml MIME Registration

MIME media type name: application

MIME subtype name: cpim-plidf+xml

Mandatory parameters: none

Optional parameters: Same as charset parameter application/xml as specified in RFC 3023 [11].

Encoding considerations: Same as encoding considerations of application/xml as specified in RFC 3023 [11].

Security considerations: See Section 10 of RFC 3023 [11] and Section 5 of this specification.

Interoperability considerations: none.

Published specification: This document.

Applications which use this media type: This document type has been used to support subscriptions to lists of presentities.

Additional Information:

Magic Number: None

File Extension: .plidf or .xml

Macintosh file type code: "TEXT"

Personal and email address for further information: Jonathan Rosenberg, <jdrosen@jdrosen.net>

Intended usage: COMMON

Author/Change controller: The IETF.

6.2 URN Sub-Namespace Registration for urn:ietf:params:xml:ns:plidf

This section registers a new XML namespace, as per the guidelines in [10].

URI: The URI for this namespace is urn:ietf:params:xml:ns:plidf

Registrant Contact: IETF, SIMPLE working group, <simple@mailman.dynamicsoft.com>, Jonathan Rosenberg <jdrosen@jdrosen.net>.

XML:

```
BEGIN
<?xml version="1.0"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML Basic 1.0//EN"
          "http://www.w3.org/TR/xhtml-basic/xhtml-basic10.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <meta http-equiv="content-type"
     content="text/html;charset=iso-8859-1"/>
  <title>Presence List Information Namespace</title>
</head
<body>
  <h1>Namespace for Presence List Information</h1>
```

```
               <h2>application/cpim-plidf+xml</h2>
               <p>See <a href="[[[URL of published RFC]]]">RFCXXXX</a>.</p>
             </body>
             </html>
             END
```

7 Author's Addresses

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

Ben Campbell
dynamicsoft
5100 Tennyson Parkway
Suite 1200
Plano, Texas 75024
email: bcampbell@dynamicsoft.com

8 Normative References

[1] J. Rosenberg, "Session initiation protocol (SIP) extensions for presence," Internet Draft, Internet Engineering Task Force, May 2002. Work in progress.

[2] J. Rosenberg, H. Schulzrinne, et al. , "SIP: Session initiation protocol," Internet Draft, Internet Engineering Task Force, Feb. 2002. Work in progress.

[3] A. Roach, "SIP-specific event notification," Internet Draft, Internet Engineering Task Force, Mar. 2002. Work in progress.

[4] H. Sugano, S. Fujimoto, et al. , "Common presence and instant messaging (CPIM)presence information data format," Internet Draft, Internet Engineering Task Force, May 2002. Work in progress.

[5] C. Jennings, J. Peterson, and M. Watson, "Private extensions to the session initiation protocol (SIP) for asserted identity within trusted networks," Internet Draft, Internet Engineering Task Force, June 2002. Work in progress.

[6] J. Peterson, "Enhancements for authenticated identity management in the session initiation protocol (SIP)," Internet Draft, Internet Engineering Task Force, Apr. 2002. Work in progress.

[7] W. W. W. C. (W3C), "Extensible markup language (xml) 1.0." The XML 1.0 spec can be found at http://www.w3.org/TR/1998/REC-xml-19980210.

[8] R. Moats, "URN syntax," RFC 2141, Internet Engineering Task Force, May 1997.

[9] R. Moats, "A URN namespace for IETF documents," RFC 2648, Internet Engineering Task Force, Aug. 1999.

[10] M. Mealling, "The IANA XML registry," Internet Draft, Internet Engineering Task Force, Nov. 2001. Work in progress.

[11] M. Murata, S. S. Laurent, and D. Kohn, "XML media types," RFC 3023, Internet Engineering Task Force, Jan. 2001.

9 Informative References

[12] M. Day, J. Rosenberg, and H. Sugano, "A model for presence and instant messaging," RFC 2778, Internet Engineering Task Force, Feb. 2000.