B. Campbell
dynamicsoft
S. Olson
Microsoft
J. Peterson
NeuStar, Inc.
J. Rosenberg
dynamicsoft
B. Stucker
Nortel Networks, Inc.
February 24, 2003

SIMPLE Presence Publication Mechanism
draft-ietf-simple-publish-00

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.

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

This Internet-Draft will expire on August 25, 2003.

Copyright Notice

Copyright (C) The Internet Society (2003). All Rights Reserved.

Abstract

This document describes an extension to the Session Initiation Protocol (SIP) [1]. The purpose of this extension to create a means for publishing event state used within the framework for SIP Event Notification (RFC3265 [2]). The first application of this extension

is targeted at the publication of presence information as defined by the SIMPLE [7] working group.

1. Introduction

This document describes a mechanism for event publication in SIP that satisfies the requirements set forward in the SIMPLE publication requirements [4]. A new SIP method, the PUBLISH method, is defined by this document.

The method described in this document allows presence information to be published to a presence agent on behalf of a user. This method can be extended to support publication of other event state, but it is not intended to be a general-purpose mechanism for transport of arbitrary data as there are better suited mechanisms for this purpose (ftp, http, etc.) This method is intended to be a simple, light-weight mechanism that employs SIP in order to support SIMPLE services.

1.1 Why a new SIP method?

In order to satisfy the requirements necessary for publishing event state to an event agent, different SIP protocol elements were evaluated, namely REGISTER and SUBSCRIBE/NOTIFY.

REGISTER solves the problem of publishing the set of contacts for a given address of record. However, the more general requirements of publishing event state to an event agent call for a different solution. Event agents (consumers of published event state) may exist anywhere in the network. With REGISTER, the sole consumer of the data being published is the registrar. For presence publication, there may be more than one event agent that is interested in the published event state. The inability to fork REGISTERs prevents this. As such, the routing requirements for published event state (e.g. a presence document) cannot be covered by the mechanisms available to us through the REGISTER method.

We already have a mechanism for publishing event state throughout the network: SUBSCRIBE/NOTIFY. The subscription mechanism exists to allow a device to assert interest in a piece of state. Typically it is used to allow potentially multiple subscribers to watch a piece of state, where the state agent could not be expected to know in advance all the potential watchers for this state and where the set of watchers changes over time. The desired publication mechanism has a different goal: publishing event state to a small number of locations which are known in advance. The target of the publication request is known in advance while the source of those publication requests are not. SUBSCRIBE/NOTIFY cannot easily solve the problem at hand.

As such, we are left with one option, to create a new method to support publication of event state to a set of possibly unknown (in a routing sense) event agents, who may or may not have expressed prior interest in receiving said data: the PUBLISH method.

1.2 Publication Classes

The sources that are publishing event state can be subdivided into classes. These classes are a logical subdivision that allows composition policy to treat different kinds of inputs in different manners. In some circumstances, the classes may be arbitrary, ephemeral and without fixed semantic value. In others, the classes may be well defined, persistent and even standardized. Examples of the latter might include classifications such as: geolocation publishers, mobile devices, automatons or PDAs. The publisher will indicate its publication class as part of the publication process. The compositor is free to use or ignore this information in conjunction with its local policy for compositing the many inputs it receives.

The publication class names are completely arbitrary, and there may be any number of inputs of any class. We envision that there will be a number of common classes that may be standardized, as well as a number of application specific classes. We will need a mechanism to avoid publication class name collisions.

There is a temptation to associate the idea of class with a tuple ID in the CPIM PIDF document. However, the tuple ID has no semantics (although some examples in early versions of the PIDF document used the tuple ID incorrectly in this fashion). Moreover, other composition applications may exist where this will not work. For example, a geolocation class might get applied across multiple tuples.

> OPEN ISSUE: Does Class overlap with work in RPIDS? Should we look to presence formats to provide their own class identifiers for status or tuple elements?

1.3 Correlating Publications from Multiple Sources

It is sometimes desirable to indicate the specific instance of a publication class that is publishing event state. This instance is intended to be a correlation identifier which is unique and consistent across multiple publications from the same source. This serves a similar purpose to the local or remote tag in a SIP dialog.

For example, a presentity might have multiple PUAs that act as "user"

inputs. The compositor might have policy to combine the state from each user PUA into the composite document. But if the same PUA publishes again, the policy may involve replacing the previous published state of that particular PUA. Doing so requires some manner of correlation identifier (publisher instance). The correlation ID is highly dynamic, and should be globally unique for any associated group of publications.

There is a temptation too have the correlation ID derive from the authentication credentials of a publisher. But there may be applications where each PUA publishes using the credentials of the presentity. This could mean that multiple PUAs would publish with the same credentials.

The PUBLISH method looks to the presence format to provide globally-unique identifiers for particular segments of presence that are in a single stream of publication. In PIDF, this would be the tuple ID. Note that presence formats must also supply a way of ordering presence information (for example, the timestamp element in PIDF).

1.4 Publication to Multiple Destinations

Just as the publication class and publication instance are used to categorize and differentiate the publication source, there is a need to categorize and differentiate the publication "destination". The compositor may then apply policy on behalf of the publisher to limit, transform, or otherwise constrain the composite event state which various watchers may receive from the PA. Some amount of metadata is required that aids in the decisions about composition and dissemination of event state.

For example, a given publisher may wish to publish geolocation information in varying degrees of fidelity. The most trusted watchers of that event state should receive the highest fidelity information. Less trusted, perhaps anonymous, watchers should receive a more restricted view of the composite state. A wide range of authorization policies can be built around this concept. To meet this requirement, the publisher might publish several versions of the event state, each somehow marked with a different identifier indicating the destination grouping of the state, or somehow instruct a presence agent to change event state before distributing it to various destinations.

There is work underway in the SIMPLE working group on a general way to provide authorization instructions to a presence agent regarding the distribution of presence information (see the SIMPLE data manipulation [5] mechanism). Publishers should use this authorization mechanism to manage the selective distribution of

presence information.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [3].

3. The PUBLISH method

The PUBLISH method is used to push data to a set of event agents that may or may not consume the data being published. The method is constructed as an OPTIONS request would be, and is allowed to fork. The Request-URI of the PUBLISH identifies the resource for whom this data is being published. As such, the sender of a PUBLISH may not know all of the endpoints that processed the request successfully, but will know if at least one endpoint accepted the request by way of the forking rules for isomorphic requests within SIP.

A PUBLISH request MAY contain a body, using the standard MIME headers to identify the content. The typical PUBLISH request will contain a body with the event state to publish. The absence of a body in a PUBLISH request may have the semantics of clearing the event state for this publication instance depending on the policy at the compositor.

The following is the BNF definition for the PUBLISH method. As with all other SIP methods, the method name is case sensitive.

```
PUBLISHm = %x50.55.42.4C.49.53.48 ; PUBLISH in caps.
```

Tables 1 and 2 extend Tables 2 and 3 of SIP [1] by adding an additional column, defining the header fields that can be used in PUBLISH requests and responses.

Header Field	where	proxy	PUBLISH
Accept	R		-
Accept	2xx		-
Accept	415		m*
Accept-Encoding	R		-
Accept-Encoding	2xx		-
Accept-Encoding	415		m*
Accept-Language	R		-
Accept-Language	2xx		-

Accept-Language	415		m*
Alert-Info	R		-
Alert-Info	180		-
Allow	R		o
Allow	2xx		o
Allow	r		o
Allow	405		m
Authentication-Info	2xx		o
Authorization	R		o
Call-ID	c	r	m
Call-Info		ar	o
Class	R		o
Contact	R		-
Contact	1xx		-
Contact	2xx		-
Contact	3xx		o
Contact	485		o
Content-Disposition			o
Content-Encoding			o
Content-Language			o
Content-Length		ar	t
Content-Type			*
CSeq	c	r	m
Date		a	o
Event		a	m
Error-Info	300-699	a	o
Expires			o
From	c	r	m
In-Reply-To	R		o
Max-Forwards	R	amr	m
Organization		ar	o

Table 1: Summary of header fields, A--O

Header Field	where	proxy	PUBLISH
Priority	R	ar	o
Proxy-Authenticate	407	ar	m
Proxy-Authenticate	401	ar	o
Proxy-Authorization	R	dr	o
Proxy-Require	R	ar	o
Record-Route		ar	-
Reply-To			o
Require		ar	c
Retry-After	404,413,480,486		o
	500,503		o
	600,603		o
Route	R	adr	o
Server	r		o
Subject	R		o
Timestamp			o
To	c(1)	r	m
Unsupported	420		o
User-Agent			o
Via	R	amr	m
Via	rc	dr	m
Warning	r		o
WWW-Authenticate	401	ar	m
WWW-Authenticate	407	ar	o

3.1 Request-URI

The Request-URI, as previously stated, for a PUBLISH identifies the resource for which the published event state is intended. For example, if we were to take the case of presence, then the Request-URI, and the To could begin as the well known address of the presentity for whom we are publishing a fragment of their presence document.

OPEN ISSUE: Is this actually what we want to do? Or is a compositor's URI is the correct destination of a PUBLISH request?

3.2 Class (Publication Class) Header

As part of the presence publication model that PUBLISH belongs to, the document that is being published may become part of a larger composite document consisting of multiple parts. This is not to be confused with multipart MIME, however. An example of this would be a presence document that spans several devices for which each presence

tuple could be considered a "part" of the overall presence document. The exact definition of what entails a recognizable portion of the overall document being published is left entirely up to the semantics of the content type being operated on.

The reverse may also be true, in that we may wish to publish a single piece of data, which the event agent compositor is expected to apply to multiple components of a composite document.

Because of this, simply identifying the resource party (TO) for which the data is intended may be insufficient in order to correctly process the document or document fragment being published. The Class (publication class) header is used to denote a token for which the published content is to be applied. Multiple tokens may be denoted in the Class header, each being separated by a comma. This is an optional header. In the absence of a Class header, the compositor may use local policy to determine an appropriate class to sort the publication information into.

```
        Class = "Class" HCOLON (token *(COMMA token))

        Example:
            Class: geoloc, mobile
```

3.3 Expires Header

The event state that is published through the PUBLISH method to a compositor/event agent is soft-state. As such, the PUBLISH SHOULD contain an expiration value for the event state data it is publishing. The intention is to inform the compositor of the expected duration of this event state. This is a separate concern from informing the watchers of this event state of the duration of the composite state.

The publication state expiration should be carried through the standard Expires: header as defined in RFC3261. The value of this expiration may be decreased by the compositor from the expiration given by the publisher, but SHOULD NOT be increased. The final response to the PUBLISH request MUST carry the expiration value chosen by the compositor in an Expires: header. In the absence of an Expires: header, the compositor is free to choose a reasonable default. It is RECOMMENDED that a default of 3600 seconds or one hour be used. The default expiration may vary from event package to event package depending on the semantics of the particular package.

When the event state expires, the publisher MAY choose to refresh the publication state by sending another PUBLISH request. When the event state expires, the compositor should apply local policy to determine the new composite event state based on the removal or expiration of this particular publication input. This will typically result in the generation of new notifications for the watchers of the composite event state.

3.4 Event Header

Every PUBLISH request MUST contain an Event: header indicating the event package for which this publication is carrying event state. In the absence of an Event: header, the compositor MUST return a 489 Bad Event response. The publish mechanism described in this document is only intended to be applied to state associated with an event package. This is the rationale behind requiring the presence of an Event: header.

When presence information is sent in a PUBLISH method, the 'presence' event is specified. When a compositor that supports presence sends a 489 Bad Event response, it MUST indicate that it supports the 'presence' event.

3.5 PUBLISH and Presence Formats

All SIP implementations that support the PUBLISH method, and use the 'presence' event, MUST implement the Presence Information Data Format (PIDF [6]) as a MIME body type that can be sent in a PUBLISH method.

If a compositor does not support the presence format provided by a publisher, it MUST return a 415 Unsupported Media Type with an Accept header listing the presence formats it does support (including 'application/cpim-pidf+xml', the media type of PIDF).

4. Examples of Use

The following section shows an example of the usage of the PUBLISH method in the case of publishing the presence document from a presence user agent to a presence agent. The watcher in this case is watching the PUA's presentity, and has previously subscribed successfully.

```
          PUA                      PA                      WATCHER
           |                        |                         |
           |                        | <---- 1. SUBSCRIBE ---- |
           |                        |                         |
           |                        | -----    200 OK ------> |
           |                        |                         |
           |                        | ----- 2. NOTIFY ------> |
           |                        |                         |
           |                        | <----    200 OK ------- |
           |                        |                         |
           | ---- 3. PUBLISH ----> |                         |
           |                        |                         |
           | <--- 4. 200 OK ------ |                         |
           |                        |                         |
           |                        | ----- 5. NOTIFY ------> |
           |                        |                         |
           |                        | <----    200 OK ------- |
           |                        |                         |
```

Message flow:

1. The watcher initiates a new subscription to the presentity@domain.com's presence agent.

2. The presence agent for presentity@domain.com processes the subscription request and creates a new subscription. In order to complete the process the presence agent sends the watcher a NOTIFY with the current presence state of the presentity.

3. A presence user agent for the presentity detects a change in the user's presence state. It initiates a PUBLISH to the presentity's presence agent in order to update it with the new presence information.

4. The presence agent receives, and accepts the presence information. The published data is incorporated into the presentity's presence document.

5. The presence agent determines that a reportable change has been made to the presentity's presence document, and sends another notification to those watching the presentity to update their information regarding the presentity's current presence status.

Messages:

```
SUBSCRIBE sip:presentity@domain.com SIP/2.0
Via: SIP/2.0/UDP 10.0.0.1:5060;branch=z9hG4bKnashds7
To: <sip:presentity@domain.com>
From: <sip:watcher@domain.com>;tag=12341234
Call-ID: 12345678@10.0.0.1
CSeq: 1 SUBSCRIBE
Expires: 3600
Event: presence
Contact: <sip:watcher@domain.com>
Content-Length: 0
```

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP 10.0.0.1:5060;branch=z9hG4bKnashds7
To: <sip:presentity@domain.com>;tag=abcd1234
From: <sip:watcher@domain.com>;tag=12341234
Call-ID: 12345678@10.0.0.1
CSeq: 1 SUBSCRIBE
Contact: <sip:watcher@domain.com>
Expires: 3600
Content-Length: 0
```

```
NOTIFY sip:presentity@domain.com SIP/2.0
Via: SIP/2.0/UDP presence.domain.com;branch=z9hG4bK8sdf2
To: <sip:watcher@domain.com>;tag=12341234
From: <sip:presentity@domain.com>;tag=abcd1234
Call-ID: 12345678@10.0.0.1
CSeq: 1 NOTIFY
Event: presence
Subscription-State: active; expires=3599
Content-Type: application/cpim-pidf+xml
Content-Length: ...

<?xml version="1.0" encoding="UTF-8"?>
<presence xmlns="urn:ietf:params:xml:ns:cpim-pidf"
          entity="pres:presentity@domain.com">
   <tuple id="j599ab8xx">
      <status>
         <basic>open</basic>
      </status>
   </tuple>
   <tuple id="pl813rt4yh">
      <status>
         <basic>open</basic>
      </status>
   </tuple>
</presence>
```

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP presence.domain.com;branch=z9hG4bK8sdf2
To: <sip:watcher@domain.com>;tag=12341234
From: <sip:presentity@domain.com>;tag=abcd1234
Call-ID: 12345678@10.0.0.1
CSeq: 1 NOTIFY
```

```
PUBLISH sip:presentity@domain.com SIP/2.0
Via: SIP/2.0/UDP pua.domain.com;branch=z9hG4bK652hsge
To: <sip:presentity@domain.com>;tag=1a2b3c4d
From: <sip:presentity@domain.com>;tag=1234wxyz
Call-ID: 12345678@pua.domain.com
CSeq: 1 PUBLISH
Expires: 3600
Event: presence
Class: mobile
Stream: 1@pua.domain.com
Facet: <sip:watcher@domain.com>
Content-Type: application/cpim-pidf+xml
Content-Length: ...

<?xml version="1.0" encoding="UTF-8"?>
<presence xmlns="urn:ietf:params:xml:ns:cpim-pidf"
          entity="pres:presentity@domain.com">
   <tuple id="j599ab8xx">
      <status>
         <basic>closed</basic>
      </status>
   </tuple>
</presence>
```

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP pua.domain.com;branch=z9hG4bK652hsge
To: <sip:presentity@domain.com>;tag=1a2b3c4d
From: <sip:presentity@domain.com>;tag=1234wxyz
Call-ID: 12345678@pua.domain.com
CSeq: 1 PUBLISH
Expires: 1800
```

```
NOTIFY sip:presentity@domain.com SIP/2.0
Via: SIP/2.0/UDP presence.domain.com;branch=z9hG4bK4cd42a
To: <sip:watcher@domain.com>;tag=12341234
From: <sip:presentity@domain.com>;tag=abcd1234
Call-ID: 12345678@10.0.0.1
CSeq: 2 NOTIFY
Event: presence
Subscription-State: active; expires=3599
Content-Type: application/cpim-pidf+xml
Content-Length: ...

<?xml version="1.0" encoding="UTF-8"?>
<presence xmlns="urn:ietf:params:xml:ns:cpim-pidf"
          entity="pres:presentity@domain.com">
   <tuple id="j599ab8xx">
      <status>
         <basic>closed</basic>
      </status>
   </tuple>
   <tuple id="pl813rt4yh">
      <status>
         <basic>open</basic>
      </status>
   </tuple>
</presence>
```

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP presence.domain.com;branch=z9hG4bK4cd42a
To: <sip:watcher@domain.com>;tag=12341234
From: <sip:presentity@domain.com>;tag=abcd1234
Call-ID: 12345678@10.0.0.1
CSeq: 2 NOTIFY
```

5. IANA Considerations

This document introduces no considerations for the IANA.

6. Security Considerations

Like all SIP entities, implementations of the PUBLISH method MUST meet all of the security implementation requirements of RFC3261 26.3.1.

A presence compositor should use the standard SIP security mechanisms to authenticate publishing user agents, and may apply authorization policies for the distribution of presence information (following the model described by SIMPLE data manipulation [5]). The composition model makes no assumptions that all input sources for a compositor are on the same network, or in the same administrative domain.

The compositor should throttle incoming publications and the corresponding notifications resulting from the changes in event state. As a first step, careful selection of default Expires: values for the supported event packages at a compositor can help limit refreshes of event state. Additional throttling and debounce logic at the compositor is advisable to further reduce the notification traffic produced as a result of a PUBLISH method.

The Class header can factor heavily into policy at the compositor. For this reason, it is important to protect the integrity and potentially the privacy of the PUBLISH headers. It is recommended that appropriate SIP integrity and privacy measures be used be employed by publishers and compositors.

Normative References

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

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

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

[4] Campbell, B., Olson, S., Peterson, J., Rosenberg, J. and B. Stucker, "SIP Presence Publication Mechanism Requirements", draft-ietf-simple-publish-reqs-00 (work in progress), February 2003.

[5] Rosenberg, J. and M. Isomaki, "Requirements for Manipulation of Data Elements in SIMPLE Systems", draft-ietf-simple-data-reqs-00 (work in progress), October 2002.

[6] Sugano, H., Fujimoto, S., Klyne, G., Bateman, A. and W. Carr, "Common Presence and Instant Messaging (CPIM) Presence Information Data Format", draft-ietf-impp-cpim-pidf-05 (work in progress), May 2002.

[7] <http://www.ietf.org/html.charters/simple-charter.html>

Authors' Addresses

Ben Campbell
dynamicsoft
5100 Tennyson Parkway
Suite 1200
Plano, TX 75025
US

EMail: bcampbell@dynamicsoft.com

Sean Olson
Microsoft
One Microsoft Way
Redmond, WA 98052
US

Phone: +1-425-707-2846
EMail: seanol@microsoft.com
URI: http://www.microsoft.com/rtc

Jon Peterson
NeuStar, Inc.
1800 Sutter St
Suite 570
Concord, CA 94520
US

Phone: +1-925-363-8720
EMail: jon.peterson@neustar.biz
URI: http://www.neustar.biz

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

EMail: jdrosen@dynamicsoft.com

Brian Stucker
Nortel Networks, Inc.
2380 Performance Drive
Richardson, TX 75082
US

EMail: bstucker@nortelnetworks.com

Full Copyright Statement

Copyright (C) The Internet Society (2003). All Rights Reserved.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself may not be modified in any way, such as by removing the copyright notice or references to the Internet Society or other Internet organizations, except as needed for the purpose of developing Internet standards in which case the procedures for copyrights defined in the Internet Standards process must be followed, or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by the Internet Society or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Acknowledgement

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