Internet-Draft
Expires: December 22, 2002

B. Campbell
dynamicsoft

S. Olson
Microsoft

J. Peterson
NeuStar, Inc.

J. Rosenberg
dynamicsoft

B. Stucker
Nortel Networks, Inc.

June 23, 2002

SIMPLE Presence Publication Mechanism
draft-olson-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 December 22, 2002.

Copyright Notice

Copyright (C) The Internet Society (2002). All Rights Reserved.

Abstract

This document describes an extension to the Session Initiation Protocol (SIP) [1]. The purpose of this extension is to create a means for publishing event state (notably presence information) as part of the SIMPLE [4] framework for presence and instant messaging.
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. Introduction

1.1 Publication Model

The following sections outline a model for publication of event state, in particular presence information. This model further defines the problem that this mechanism is attempting to solve.

1.1.1 Presence Composition

Most existing presence services involve a single PUA that has complete presence for a given presentity. This allows for a very simple model where that PUA sends full presence state to a PA, which then distributes it to watchers.

But this is a limited view of presence. In general, the presence state of a presentity may be derived from many different inputs. A complete view of presence for a presentity is likely to be derived from more than one source, where the the complete view of presence state is composed of the presence state from each source. This document proposes a logical model for such presence composition.

Presence composition is a logical function in a presence distribution system. This function is fulfilled by a logical element known as a "presence compositor". A presence compositor accepts presence inputs from one or more PUAs, and composes these inputs into a composite presence document.

```
+-----------------+      +---------+
|Presence         |      |Presence |
|Compositor       +------+Agent    |
|                 |      |         |
|                 |      |         |
+--------+--------+      +---------+  
          |      |         |
          |      |         |
          +--+--+    +--+--+   +--+--+    +-+--+
          |PUA  |    |PUA  |   |PUA  |    |PUA |
          +-----+    +-----+   +-----+    +----+
```

Each PUA publishes its view of presence to the Presence Compositor, which then publishes to a presence agent for distribution. Each PUA
publishes a full view of presence from its perspective--each publication carries full state, and does not depend on previous states for the particular PUA. A PUA does not necessarily even know that it is publishing to a compositor, rather than a presence agent.

The transformations that a presence compositor uses for this composition are entirely a matter of local policy. The policy could be as simple as the creation of a combined CPIM PIDF [3] document where each input represents a separate tuple. It can also involve more complex transformations, such as modifying the information from one source based on information from another source.

1.1.2 Interface between the Compositor and the PA

The interface between a compositor and a PA is also a matter of local policy. The compositor might act as a PUA itself, and publish presence to the PA just like any other PUA might. The compositor and the PA may be collocated, and communicate via local procedure calls. Specification of this interface is beyond the scope of this document.

1.1.3 Compositor Input Slots

The compositor may be further modeled in terms of "input slots". An input slot is a completely logical concept to allow composition policy to treat different kinds of inputs in different manners. For example, a compositor may accept publication from a GeoLoc service, as well as end user devices. It is highly likely that the composition policy would treat each sort of input differently.

```
+----------------+
|Presence        |
|Compositor      |
+----------------+
|Slot            |
|Slot            |
+----------------+
|GeoLoc          |
|User            |
+----------------+
|PUA             |
|PUA             |
+----------------+
```
A compositor input slot has two attributes. One is a "type" that designates what type of input is occurring. In the above example, one slot is of type "geoloc", while the other is of type "user". The type names are completely arbitrary, and there may be any number of inputs of any type. We envision that there will be a number of common types that may be standardized, as well as a number of application specific types. We will need a mechanism to avoid type name collisions.

There is a temptation to associate the idea of type with a tuple ID in the CPIM PIDF document. Indeed, this is a perfectly valid application. But other composition applications may exist where this will not work. For example, a GeoLoc input might get applied across multiple tuples.

The second compositor input attribute is a correlation identifier that allows a compositor to associate multiple publications from the same device. 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. The correlation ID is highly dynamic, and should be globally unique for any associated group of publications.

There is a temptation to have the correlation ID derive from the authentication credentials of a publishing PUA. 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.

1.2 Why SIP?

The problem, then, can be expressed as defining a mechanism for communication of event state between the event publisher and the potential compositor of that event state. Two principal protocol candidates exist for this event state publication: HTTP and SIP.

HTTP is well suited for moving data around in the form of MIME body parts. An HTTP client-to-server publication solution would not require much work to specify. A SOAP over HTTP solution would additionally allow complex transaction semantics with little additional work. HTTP, however, does not have a well-defined routing model at the application level. It works fine if the publication point is well known and fairly static, but it will require additional work to deal with situations where the publication point changes dynamically.
SIP, on the other hand, is built around the concept of mobility of endpoints. The SIP proxy, registrar, and location service concepts provide a rich mechanism of finding a dynamically changing endpoint from and address of record. The application-level routing capabilities of SIP can be very useful for presence publication. If all PUAs for a given presentity exist in the same administrative domain, then they can most likely publish directly to a compositor. But if PUAs exist outside the administrative domain, it is likely they will not be able to do so.

For example, suppose that Alice uses a presence service that allows multiple PUAs to publish to a compositor inside the service provider network. Further suppose that Alice wishes to incorporate presence information from an external provider, that has no business relationship with her primary provider. For this example, Alice wishes to use a shared browsing service that tracks the "location" she is currently browsing in the web. That service acts as a PUA on Alice’s behalf, and publishes the information to her primary presence provider. Other users of the shared browsing service can subscribe to her presence information, and determine when they are browsing the same site.

The presence provider is highly unlikely to allow the external PUA to send data directly to the compositor. But if Alice registers a contact with a methods parameter value of "PUBLISH", that PUA can send a publish request to an edge proxy in the presence providers network, and use Alice’s address of record as the requestURI. This AoR could be her normal SIP URI, or it could be a special AoR for the purposes of presence publication. The proxy forwards the request to the compositor, without the external PUA having to talk directly to the compositor, or even know its IP address.

Now consider that Alice’s primary provider is actually an enterprise. That enterprise has different compositors for different departments. The external provider has no way of knowing this internal organization, nor does it know what department Alice works in. Still, Alice register’s her publication contact at an enterprise registrar, the external provider sends the publish request to Alice’s address of record, and the companies internal SIP network handles things from there, eventually getting the request to the correct compositor.

The composition model does not at first appear to require publishing to dynamically changing PAs. But a very powerful, but often forgotten, aspect of SIMPLE is it allows a PA to exist on an end-user device. Indeed, some early implementations of SIMPLE rely on exactly that model. It is reasonable to expect the composition model above to co-exist with end-user device PAs, where the PA location changes
dynamically.

For example, imagine Alice hosts a PA on her PC, which acquires its IP address via DHCP. This address changes relatively frequently, and registers a publish contact with an enterprise registrar. Now, imagine she also has a mobile phone which contains a PUA. She wants her presence document to show a combined view of her PCs concept of her presence and her mobile phone service’s concept of her presence. She cannot simply tell the mobile service her PC address since it changes often. Instead, she tells the service an AoR to publish presence to. The mobile service publishes presence state to that AoR, which resolves to a SIP proxy or redirect server. Normal SIP proxy or redirect behavior is invoked to get the publication to Alice’s PC based on her publish contact registration.

It is our opinion that SIP style routing is very useful for presence publication. Without the application level routing capabilities of SIP, it would be difficult to build these sort of services. It is more appropriate to add a publication mechanism to SIP than to standardize SIP-style routing features for HTTP proxies.

1.3 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 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. This mechanism is used to keep a set of distributed state machines in synchrony regarding a particular set of events. However, the action of sending a notification implies that a subscription exists. This may have been created explicitly, by sending a SUBSCRIBE, or through some implied means. In either case, the semantics of subsequent NOTIFY messages (which are of interest in that they publish event state to elements in the network that are watching the given event agent) are such that they are generated by
event agents. In the strictest terms, publishing data for more general purposes does not require that the publisher is an event agent or knows prior to publishing, who, if anyone, is interested in the data they wish to publish. Therefore, although SUBSCRIBE/NOTIFY is obviously well suited to pushing event state throughout the network FROM an event agent, its semantics are not well suited to publishing event state TO an event agent in the general case. SUBSCRIBE/NOTIFY are intended to push data FROM the event agent to endpoints that may not serve the given resource, but are interested in its state. This means that its not intended to publish data only to those elements which are acting as event agents.

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.
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 [2].
3. 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 would be, and is allowed to fork. The request URI of the PUBLISH identifies the resource for whom this data is being published for. 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.

Additionally, unlike the SUBSCRIBE method, the PUBLISH method does not create a SIP dialog as part of its processing. Creation of a dialog implies that the sender and recipient need to track the state of the PUBLISH itself, which need not be necessary for its proper operation. Therefore, there are no requirements on use/reuse of the Call-Id, or tags from PUBLISH to PUBLISH outside of the normal rules of SIP.

A PUBLISH request MAY contain a body, using the standard MIME headers to identify the content.

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.

3.2 PType (Publication Type) 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 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 PType (publication-type) header is used to denote a token for which the published content is to be applied. Multiple tokens may be denoted in the PType header, each being separated by a comma. This is an optional header.

Example:

PType: mobile, geopriv

3.3 PStream (Publication-Stream) Header

It is thought to be a general property of any event subscription system whose notifications contain data to be pushed to watchers, that the context of that data is dependent on the sequence in which it arrives. For instance, with presence, the temporal order in which status changes are processed can obviously have consequences on a user’s status (e.g. if an offline indication is processed before an online indication, the user may show up as being online when they are in fact, offline).

For SUBSCRIBE/NOTIFY, a dialog is created by reusing the Call-ID/tags between each message, and the sequence of events is preserved through the normal use of the CSeq counter. However, creating a dialog is not required in order to preserve the sequence of events in the general case. In the case of application/CPIM-PIDF+XML, the timestamp portion of the presence tuple can easily be used to preserve temporal ordering instead of the CSeq. For registrations, there is the recommendation that the same Call-ID and tags are reused for each registration during a power-cycle for a particular client device. However, this does not guarantee that clients will follow the rules, and thus, sequencing may be lost as a result.

Because a dialog is not required in order to correctly sequence multiple PUBLISH transactions, there is no compelling reason to require the Call-IDs and tags to be restricted such that the CSeq can be used to infer correct sequencing of the content of the PUBLISH.

In order to guarantee correct sequencing, when the content of the publish has no such mechanism, or to be used in lieu of any such mechanisms by the compositor, a publication stream identifier is introduced. The publication stream identifier (PStream) simply is a globally unique token (much like a Call-ID) that simply identifies
the stream of publications to which the current publication is a part. The compositor may use other header information, or information in the message body to further guarantee correct sequencing. For instance, if temporal sequencing of publications is important (as is very likely), then a timestamp or synthetic clock may be introduced as part of the message body content. Likewise, a Date header can be used in lieu of the message body to identify the correct temporal ordering of publications.

Example:

PStream: 12345678@10.0.0.1

If ordering or versioning is important to the application, then it MUST be captured in the content itself if it cannot be easily derived through existing SIP headers.

Applications MUST NOT rely on TCP, or other reliable transports in order to guarantee that their publications will arrive, and be understood by the compositor as coming from the same publication source. The transport characteristics of the first hop does not guarantee the same characteristics on later hops.

3.4 PState (Publication-State) Header

The event state that is published through the PUBLISH method to a compositor/event agent is soft-state. As such, the PUBLISH may contain an expiration value for the event state data it is publishing. The response to the PUBLISH MUST contain the Publication-State header in order to provide the publisher the duration for which the publication is considered valid. The value of this may be decreased from the expiration given by the publisher in the original PUBLISH method, but SHOULD NOT be increased.

Example:

PState: expires=1800
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 ------>
|                       | ----- 200 OK ------>
|                       | ----- 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="mobile-phone">
  <status>
    <basic>open</basic>
  </status>
</tuple>
<tuple id="desktop">
  <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
PType: mobile
PStream: 1@pua.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="mobile-phone">
    <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
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"?>
<pres xmlns="urn:ietf:params:xml:ns:cpim-pidf"
entity="pres:presentity@domain.com">
<tuple id="mobile-phone">
  <status>
    <basic>closed</basic>
  </status>
</tuple>
<tuple id="desktop">
  <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. Security Considerations

A presence compositor should authenticate publishing PUAs, and may apply authorization policies. The composition model makes no assumptions that all input sources for a compositor are on the same network, or in the same administrative domain.
6. Open Issues

1. Should we restrict the presence compositor to only decrease the publication expiration value and not increase it?

2. What is a reasonable set of "slots" to standardize on?

3. How do we handle publication of "large" data? (greater than 1200 bytes)

4. Are there additional security concerns to be addressed?

5. Do we need to support discovery of slots?
Normative References


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 (2002). 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.