SIMPLE WG
Internet-Draft
Expires: December 28, 2003

A. Niemi, Ed.
Nokia

June 29, 2003

Session Initiation Protocol (SIP) Extension for Presence Publication
draft-ietf-simple-publish-01

Status of this Memo

Copyright Notice

Abstract

This document describes an extension to the Session Initiation Protocol (SIP) for publishing event state used within the framework for SIP Event Notification. The first application of this extension is targeted at the publication of presence information.

The mechanism described in this document can be extended to support publication of any event state, for which there exists an appropriate event package. 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.)

Table of Contents

1. Introduction

The focus of this specification is to provide a framework for the publication of event state from a user agent to an entity that is responsible for compositing this event state and distributing it to interested parties through the SIP events [1] framework. This specification fills a gap in the SIP events framework to allow for a client to push its event state to the state agent that acts on its behalf.

The first application of this mechanism is the publication of presence state by a presence user agent to a presence compositor which has a tightly coupled relationship to the presence agent. The requirements and model for presence publication are documented in [2]. This specification will address each of those requirements.

The mechanism described in this document can be extended to support publication of any event state, for which there exists an appropriate event package as defined in [1]. 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 [7], HTTP [8], etc.)

2. Terminology and Conventions

In addition to the terminology of RFC 3265 [1] and RFC 3261 [3], this document introduces some new concepts:

Event State: The composition state of a resource.

Event Publication Agent (EPA): The UAC which issues a PUBLISH request to publish event state or event state segments. For presence, this corresponds to the PUA.

Event State Compositor (ESC): The UAS which processes PUBLISH requests and is responsible of compositing event state or event state segments into a complete, composite event state. For presence, this corresponds to the PA.

Event State Segment: For some event packages, there exists a natural decomposition of event state into event state segments. For presence, such decomposition is the presence tuple.

Hard State: Hard state is the steady-state or default state version of event state at the ESC, which may be used in the absence of any other soft state publications.

Soft State: Soft state is a version of event state at the ESC, that is published by the EPA. Soft state has a defined lifetime and will expire after a negotiated amount of time.

Version Identifier: A protocol element (i.e., an entity-tag) that is used to identify a specific soft state version of published event state at the ESC.

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 [4] and indicate requirement levels for compliant implementations.

3. Overall Operation

This document defines a new SIP method, PUBLISH, for publishing event state. PUBLISH is analogous to REGISTER in that it allows a user to add, modify, and remove state in another entity which manages this state on behalf of a user. The user may in turn have multiple UAs or endpoints. Each endpoint may publish its own unique state and through a subscription to that event package discover the event state of the other active endpoints.

In the generic sense, a UAC which publishes event state is labelled an Event Publication Agent (EPA). For presence in particular, this is the familiar PUA role as defined in [5]. The entity which processes the PUBLISH request is known as an Event State Compositor (ESC). For presence in particular, this is the familiar PA role as defined in [5].

PUBLISH requests create soft state in the state agent. This state has a defined lifetime and will expire after a negotiated amount of time, requiring the publication to be refreshed by subsequent PUBLISH requests. Local policy at the compositor may in turn define hard-state for a particular event package. That is, the steady-state of this event package in the absence of any other soft state provided through the PUBLISH method.

Typically, the body of a PUBLISH request carries the published event state. In the response to a PUBLISH request, the EPA assigns a version identifier to the published event state or event state segment. This identifier is used by the EPA as part of a version precondition to subsequent refreshing PUBLISH requests of that event state. In the event that the publication refresh is to an outdated version of event state, the versioning precondition will fail. This enables an EPA to detect collisions between new and refresh publications of the same event state among a set of endpoints.

Event state publication inherently involves at least two parties: the source of the publication and the target of the publication. The source of the publication is naturally represented as an address-of-record (AOR).

For some types of event state, namely presence, the target of the publication may not sufficiently be represented by an address-of-record (AOR) alone. Rather, the target is a combination of both an AOR and a unique identifier which acts to represent one of N possible sections of an overall event state for that AOR. In this specification, these sections are referred to as event state segments.

In the context of presence publication, the event state segment is nothing more than the presence tuple associated with the presentity (AOR). It is the role of the compositor to aggregate these segments into a complete event state which is presented to the subscribers of that event state. This composition logic is a matter of local policy.

For some event packages, there is no natural decomposition of event state into these segments and for these packages, an AOR is sufficient to identify the target of the publication.

4. Prerequisites for Event Packages using PUBLISH

In order to make use of the event publication mechanism, certain prerequisites have to be fulfilled for each specific event package. In order to satisfy the requirements of [2], the body of the PUBLISH request must fulfill several requirements as well.

This section outlines these prerequisites, and demonstrates how they are fulfilled specifically for presence publication.

4.1 PUBLISH Bodies

Any application of the PUBLISH mechanism for a given event package MUST support a content type which fulfills the requirements in [2]. Each event package MUST also describe the semantics associated with that content and MUST prescribe a default, mandatory to implement format.

This document defines the semantics of the presence publication requests (event package "presence") when the CPIM PIDF [6] presence document format is used. A PUA which uses PUBLISH to publish presence state to the PA MUST support the CPIM PIDF presence format.

4.2 PUBLISH Response Bodies

The response to a PUBLISH request indicates whether the request was successful or not. In general, the body of such a response will be empty unless the event package defines explicit meaning for such a body.

There is no such meaning for a response to a presence publication when the document format used is CPIM PIDF.

4.3 Partial Event State

The content type MUST provide a way to publish partial state for an event package. The intention is to allow each device or client for an address-of-record to publish event state independently. To accomplish this, the event state that is published by these devices MUST be allowed to be only a portion of the complete state that the state agent advertises for that AOR.

> Note that sources for event state other than those using the PUBLISH mechanism are explicitly allowed. It is beyond the scope of this document to define such interfaces.

For presence in particular, a PUA can publish presence state for just a subset of the tuples that may be composited into the presence document that watchers receive in a NOTIFY. The mechanism by which the ESC aggregates this information is a matter of local policy.

4.4 Event State Decomposition

If the content type allows for event state segments to be represented, the content type MUST provide a means to uniquely identify each unique segment.

For presence, the CPIM PIDF presence document provides a tuple-ID to distinguish the segments of the presence document associated with the encompassing presentity.

> OPEN ISSUE: The specifics of how the tuple-ID is used to identify specific segments of the composite state is still open. Currently, a specific naming convention for the tuple-ID seems like a reasonable approach. However, this naming convention is to be defined.

4.5 Default Expiration of PUBLISH

PUBLISH establishes soft state which expires after a negotiated amount of time. Each event package MUST provide a default expiration value recommendation (SHOULD strength).

For presence publication, it is RECOMMENDED that the ESC use a default value of 3600 seconds (1 hour) for this default expiration value.

5. Constructing the PUBLISH Request

PUBLISH requests create, remove, and modify event state. A PUBLISH request can create new event state in the state agent, associating this event state with an address-of-record and optionally with a unique identifier for segments of event state being published. Publication on behalf of a particular address-of-record may also be performed by a suitably authorized third party. To determine the current published state for a particular address-of-record, the client MAY create a subscription for this address-of-record and event package using the SUBSCRIBE/NOTIFY mechanism of RFC3265 [1].

> Note that in the case the event state is segmented, each segment logically represents an independent publication that may be added, removed, modified, and expired separately.

Except as noted, the construction of the PUBLISH request and the behavior of clients sending a PUBLISH request is identical to the general UAC behavior described in Section 8.1 and Section 17.1 of RFC 3261 [3].

If necessary, clients may probe for the support of PUBLISH using the OPTIONS request defined in SIP [3]. The presence of "PUBLISH" in the "Allow" header field in a response to an OPTIONS request indicates support for the PUBLISH method. In addition, the "Allow-Events" header field indicates the supported event packages.

A PUBLISH request does not establish a dialog. A UAC MAY include a Route header field in a PUBLISH request based on a pre-existing route set as described in Section 8.1 of RFC3261. The Record-Route header field has no meaning in PUBLISH requests or responses, and MUST be ignored if present. In particular, the UAC MUST NOT create a new route set based on the presence or absence of a Record-Route header field in any response to a PUBLISH request. The PUBLISH request MUST NOT contain a Contact header.

The following header fields are included in a PUBLISH request:

Request-URI: The Request-URI initially contains the address-of-record whose publication is to be created, removed, or modified. The address-of-record MUST be a SIP URI or SIPS URI. Unlike the REGISTER request, the Request-URI SHOULD contain both "userinfo" and "@" components.

To: The To header field contains the address-of-record whose publication is to be created, removed, or modified. The To header field and the Request-URI field are typically the same. This address-of-record MUST be a SIP URI or SIPS URI.

From: The From header field contains the address-of-record of the entity responsible for the publication. The value is the same as the To header field unless the request is a third-party publication. This address-of-record MUST be a SIP URI or SIPS URI.

Event: PUBLISH requests MUST contain a single Event header field. The value of this header field indicates the event package for which this request is publishing event state.

Expires: PUBLISH requests SHOULD contain a single Expires header field. This value indicates the lifetime of the event state being published by this request. A special value of "0" indicates the removal of any prior soft state established by a prior PUBLISH request from this EPA.

If-Match: PUBLISH requests MAY contain a single If-Match header field. This header field SHOULD contain one or more entity-tags provided by the ESC, to be used as a versioning precondition to a PUBLISH refresh.

The PUBLISH request MAY contain a body, which contains event state that the client wishes to publish. The content format and semantics are dependent on the event package identified in the Event header.

As with any other SIP message, the PUBLISH mechanism MAY use the content indirection mechanism defined in [9]. There are no additional requirements or restrictions on content indirection as applied to the PUBLISH request. Content indirection is a useful mechanism for communicating large event state information that cannot reasonably be carried directly within the SIP signaling (PUBLISH request).

5.1 Creating Initial Publication

The PUBLISH request created by the EPA and sent to the Event State Compositor (ESC) establishes soft state in the state agent for the event package indicated in the request and bound to the address-of-record in the To header of the request. Additionally, the PUBLISH request may publish event state that is further sub-divided into segments of event state that may be manipulated independently. As an example, presence publication using the CPIM PIDF format may manipulate individual tuples related to a common presentity.

OPEN ISSUE: Atomicity of publication. How exactly is this handled? Can an initial "full" PIDF document be split into separate tuples later on? If the initial publication contains several tuples, do each of them inherit the version identifier? How can the EPA publish presentity level information, e.g., presentity note?

Once the initial PUBLISH request has been processed by the ESC, the EPA MAY send subsequent PUBLISH requests to refresh, modify, or delete the publication state established by the first PUBLISH request. These operations will be described in subsequent sections.

EPAs MUST NOT send a new PUBLISH request (not a re-transmission) until they have received a final response from the state agent for the previous one or the previous PUBLISH request has timed out.

5.2 Setting the Expiration Interval

When a client sends a PUBLISH request, it SHOULD suggest an expiration interval that indicates how long the client would like the publication to be valid. The actual duration of the soft state is defined by local policy at the ESC.

The expiration value is presented in the Expires header of the PUBLISH request. If an Expires header is not present, the client is indicating its desire for the server to choose. It is RECOMMENDED that the PA use a value of 3600 seconds (1 hour) for this default expiration value in the case of presence publication. The default value is generally event package specific.

If an EPA receives a 423 (Interval Too Brief) response to a PUBLISH request, it MAY retry the publication after changing the expiration interval in the Expires header to be equal to or greater than the expiration interval within the Min-Expires header field of the 423(Interval Too Brief) response.

5.3 Refreshing Event State

Each EPA is responsible for refreshing the publications that it has previously established.

The 200 (OK) response from the state agent MUST contain an Expires header indicating the expiration time interval for the publication. The EPA then issues a PUBLISH request for each of its publications before the expiration interval has elapsed.

Also, the 200 (OK) response from the state agent MUST contain an ETag header with a single entity-tag indicating the version information of the publication. To refresh the event state, the EPA MUST include the

received entity-tag in an If-Match header field in a PUBLISH request.

> Note that for the EPA, the entity-tag is simply an opaque string. It carries no further semantics for the EPA.

The If-Match header field with the version identifier entity-tag establishes a versioning precondition to the PUBLISH request. If the version identifier matches the version maintained by the ESC, the refresh is successful, and the EPA receives a 200 (OK) response. If there is no matching version at the ESC, i.e., the refreshed event state is out-of-date, the EPA receives a 412 (Precondition Failed) response to the PUBLISH request.

If an EPA receives a 412 (Precondition Failed) response, it MUST NOT reattempt to refresh the event state. Instead, the EPA SHOULD query a principal for further actions.

> OPEN ISSUE: This may need some more thought. It's easy to see that in a presence system, the UA could prompt the user when a refresh fails. But there may be systems consisting of automata only, where such a concept does not make much sense.

Also, to recover from this error, the client MAY determine the current version of the event state at the server by sending a SUBSCRIBE request to the server and re-issue the PUBLISH request if the event state changes again.

A PUBLISH refresh SHOULD NOT contain a body.

5.4 Modifying Event State

Modification of event state is considered a new publication similar to the creation of initial event state. Because the modification of event state is not a refresh publication, the EPA does not include a versioning precondition in the PUBLISH request. Therefore, the PUBLISH request MUST NOT include an If-Match header field, and the EPA MUST discard any previously received version identifier for this event state.

5.5 Removing Event State

PUBLISH establishes soft state which expires unless refreshed. This event state may also be explicitly removed. A UA requests the immediate removal of event state by specifying an Expires value of "0" in the PUBLISH request. Such a request SHOULD NOT contain any body. UAs which support PUBLISH SHOULD support this mechanism for explicitly removing event state.

5.6 Querying the Current Event State

To query the event state that the state agent in fact delivers to the subscribers, the client may SUBSCRIBE to the event package for which it has sent a PUBLISH, indicating the same address-of-record in the To header. An Expires header value of "0" may be used in this SUBSCRIBE request to do a one-time fetch of this event state as defined in RFC3265.

6. Processing PUBLISH Requests

The Event State Compositor (ESC) is a UAS that responds to PUBLISH requests and maintains a list of publications for a given address-of-record. The ESC MUST ignore the Record-Route header field if it is included in a PUBLISH request. The ESC MUST NOT include a Record-Route header field in any response to a PUBLISH request.

The ESC has to know (for example, through configuration) the set of domain(s) for which it maintains event state. PUBLISH requests MUST be processed in the order that they are received. PUBLISH requests MUST also be processed atomically, meaning that a particular PUBLISH request is either processed completely or not at all.

A client may probe the ESC for the support of PUBLISH using the OPTIONS request defined in SIP [3]. In the response to an OPTIONS request, the ESC SHOULD include "PUBLISH" to the list of allowed methods in the "Allow" header field. Also, it SHOULD list the supported event packages in an "Allow-Events" header field.

> The "methods" parameter for Contact may also be used to specifically announce support for PUBLISH messages when registering. (See reference [10] for details on the "methods" parameter).

When receiving a PUBLISH request, the ESC follows these steps:

1. The ESC inspects the Request-URI to determine whether this request is for a domain supported by the ESC. If not, the ESC SHOULD proxy the request to the addressed domain.

2. To guarantee that the ESC supports any necessary extensions, the ESC MUST process the Require header field values as described for UASs in Section 8.2.2 of RFC3261.

3. An ESC SHOULD authenticate the UAC. Possible mechanisms for the authentication of SIP user agents are described in Section 22 of RFC3261.

4. The ESC extracts the address-of-record from the To header field of the request. If the address-of-record is not valid for the domain in the Request-URI, the ESC MUST send a 404 (Not Found) response and skip the remaining steps. Else, the URI MUST then be converted to a canonical form. To do that, all URI parameters MUST be removed (including the user-param), and any escaped characters MUST be converted to their unescaped form. The result serves as an index into the list of publications maintained by the ESC.

5. The ESC SHOULD determine if the authenticated user is authorized to publish for the address-of-record of the To header field. If the authenticated user is not authorized to publish, the ESC MUST return a 403 (Forbidden).

   Note that this authorization may take into account third party publication of event state.

6. The ESC examines the Event header of the PUBLISH request. If the Event header is missing or contains an event package which the ESC does not support, the ESC MUST respond to the PUBLISH request with a 489 (Bad Event) response.

7. The ESC now processes the Expires header value from the PUBLISH request.

   * If the request has an Expires header field, that value MUST be taken as the requested expiration.

   * Else, a locally-configured default value MUST be taken as the requested expiration.

   * The ESC MAY choose an expiration less than the requested expiration interval. If and only if the requested expiration interval is greater than zero AND less than a locally-configured minimum, the ESC MAY reject the publication with a response of 423 (Interval Too Brief), and skip the rest of the remaining steps. This response MUST contain a Min-Expires header field that states the minimum expiration interval the ESC is willing to honor.

8. The ESC examines the If-Match header of the PUBLISH request. If the If-Match header is absent, the request is a new publication; if the request contains a version precondition in the form of an If-Match header field, the request is a publication refresh. The ESC extracts any entity-tags contained in the If-Match header and then matches those entity-tags against all locally stored entity-tags for this address-of-record and event package. If no

match is found, the ESC MUST reject the publication with a response of 412 (Precondition Failed), and skip the remaining steps.

9. The ESC may then process the body of the PUBLISH request (the actual event state). If the request contains no body (when it should contain one), or the Content-Type of the request does not match the event-package, or is not understood by the ESC, the ESC MUST reject the request with an appropriate response.

   * For each publication, the ESC will record the target of the publication (To URI), the source of the publication (From URI), and the version of the publication.

     Note that this version information will be generated by the ESC when receiving a new publication, and will be present in the If-Match header field in publication refreshes.

   * For new publications, i.e., publications without a version precondition, the ESC MUST generate a locally unique entity-tag, and store it, replacing any existing entity-tags stored for that particular event state. The new entity-tag MUST be delivered to the EPA in an ETag header field of a 200 (OK) response.

     Note that the exact way in which the ESC creates the version entity-tag is a matter of local policy. One reasonable implementation of a version entity-tag is a counter which is incremented by one for each new version.

   * The processing of the PUBLISH request must be atomic. If internal errors (such as the inability to access a back-end database) occur before processing is complete, no portion of the PUBLISH document must be published and the ESC MUST fail with a 500 (Server Error) response.

10. The ESC returns a 200 (OK) response. The response MUST contain an Expires header indicating the expiration interval chosen by the ESC. The response MUST also contain an ETag header indicating the version of the published event state. The state agent associated with this ESC may then issue appropriate NOTIFY requests to any watchers of this event state.

    Note that the timing between the receipt of the PUBLISH request and the issuance of NOTIFY requests is implementation dependent and may also vary according to throttling policies at the state agent.

7. Syntax

This section describes the syntax extensions required for event publication in SIP. Note that the formal syntax definitions described in this section are expressed in the Augmented BNF format used in SIP [3], and contain references to elements defined therein.

7.1 New Methods

7.1.1 PUBLISH Method

"PUBLISH" is added to the definition of the element "Method" in the SIP message grammar. As with all other SIP methods, the method name is case sensitive. PUBLISH is used to publish event state to an entity responsible for compositing this event state.

Table 1 and Table 2 extend Tables 2 and 3 of RFC 3261 [3] by adding an additional column, defining the header fields that can be used in PUBLISH requests and responses.

Header Field	where	proxy	PUB
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
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	PUB
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

Table 2: Summary of header fields, P--Z

7.2 New Response Codes

7.2.1 "412 Precondition Failed" Response Code

The 412 response is added to the "Client-Error" header field definition. "412 Precondition Failed" is used to indicate that the precondition given for the request has failed.

7.3 New Header Fields

Table 3 expands on Table 2 in SIP [3], as amended by the changes in Section 7.1.

Header Field	where	proxy	PUB
ETag	2xx		m
If-Match	R		o

Table 3: Summary of header fields, A--O

7.3.1 "ETag" Header

ETag is added to the definition of the element "general-header" in the SIP message grammar. Usage of this header is described in Section 6.

7.3.2 "If-Match" Header

If-Match is added to the definition of the element "general-header" in the SIP message grammar. Usage of this header is described in Section 5.

7.4 Augmented BNF Definitions

This section describes the Augmented BNF definitions for the various new and modified syntax elements. The notation is as used in SIP [3] and the documents to which it refers.

```
PUBLISHm            = %x50.55.42.4C.49.53.48 ; PUBLISH in caps.
extension-method    = PUBLISHm / token
ETag                = "ETag" HCOLON entity-tag
If-Match            = "If-Match" HCOLON entity-tag
                      * (COMMA entity-tag)
entity-tag          = quoted-string
```

8. IANA Considerations

This document registers a new method name, a new response code and two new header field names.

8.1 Methods

This document registers a new SIP method, defined by the following information which is to be added to the method and response-code sub-registry under http://www.iana.org/assignments/sip-parameters.

Method Name: PUBLISH
Reference: [RFCYYYY]

(Note to RFC Editor: Replace YYYY with the RFC number of this document when published).

8.2 Response Codes

This document registers a new response code. This response code is defined by the following information, which is to be added to the method and response-code sub-registry under http://www.iana.org/assignments/sip-parameters.

Response Code Number: 412
Default Reason Phrase: Precondition Failed

8.3 Header Field Names

This document registers two new SIP header field names. These headers are defined by the following information, which is to be added to the header sub-registry under http://www.iana.org/assignments/sip-parameters.

Header Name: ETag
Compact Form: (none)

Header Name: If-Match
Compact Form: (none)

9. Security Considerations

The state agent SHOULD authenticate the Event Publication Agent (EPA), and SHOULD apply its authorization policies to all requests. The composition model makes no assumptions that all input sources for a compositor (ESC) are on the same network, or in the same administrative domain.

The ESC 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 ESC can help limit refreshes of event state. Additional throttling and debounce logic at the ESC is advisable to further reduce the notification traffic produced as a result of a PUBLISH method.

Integrity protection and privacy of the PUBLISH requests can be ensured using the S/MIME mechanisms outlined in section 23 of RFC3261. Integrity protection of the To, From, Call-ID, CSeq, Event, ETag, If-Match, Route, and Expires headers should be done at a minimum.

If the ESC receives a PUBLISH request which is integrity protected using a security association that is not with the ESC (for example, end-to-end S/MIME integrity protection), the state agent coupled with the ESC MUST NOT modify the event state before exposing it to the watchers of this event state in a NOTIFY request(s). This is to preserve the end-to-end integrity of the event state.

10. Examples

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. The PUA will SUBSCRIBE to its own presence to see the composite presence state exposed by the PA. This is an optional but likely step for the PUA.

```
       PUA                     PA                     WATCHER
      (EPA)                   (ESC)
        |                       |                         |
        |                       | <---- M1: SUBSCRIBE --- |
        |                       |                         |
        |                       | ----- M2: 200 OK -----> |
        |                       |                         |
        |                       | ----- M3: NOTIFY -----> |
        |                       |                         |
        |                       | <---- M4: 200 OK ------ |
        |                       |                         |
        | --- M5: SUBSCRIBE --> |                         |
        |                       |                         |
        |<--- M6: 200 OK    --> |                         |
        |                       |                         |
        |<--- M7: NOTIFY  ----- |                         |
        |                       |                         |
        | --- M8: 200 OK    --> |                         |
        |                       |                         |
        | --- M9: PUBLISH ----> |                         |
        |                       |                         |
        | <-- M10: 200 OK ----  |                         |
        |                       |                         |
        |                       | ----- M11: NOTIFY ----> |
        |                       |                         |
```

```
        |                       | <---- M12: 200 OK ----- |
        |                       |                         |
        |                       |                         |
        |<---- M13: NOTIFY ---- |                         |
        |                       |                         |
        |----- M14: 200 OK  --> |                         |
        |                       |                         |
```

Message flow:

M1: The watcher initiates a new subscription to the presentity@domain.com's presence agent.

```
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
```

M2: The presence agent for presentity@domain.com processes the subscription request and creates a new subscription. A 200 (OK) response is sent to confirm the subscription.

```
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:pa@domain.com>
Expires: 3600
Content-Length: 0
```

M3: In order to complete the process, the presence agent sends the watcher a NOTIFY with the current presence state of the presentity.

```
NOTIFY sip:presentity@domain.com SIP/2.0
Via: SIP/2.0/UDP pa.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>
            <timestamp>2003-02-01T16:49:29Z</timestamp>
         </tuple>
         <tuple id="desktop">
            <status>
               <basic>open</basic>
            </status>
            <timestamp>2003-02-01T12:21:29Z</timestamp>
         </tuple>
      </presence>
```

M4: The watcher confirms receipt of the NOTIFY request.

```
      SIP/2.0 200 OK
      Via: SIP/2.0/UDP pa.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
      Contact: <sip:watcher@domain.com>
```

M5: To view its composite presence state, the PUA issues a SUBSCRIBE to the PA for itself.

```
      SUBSCRIBE sip:presentity@domain.com SIP/2.0
      Via: SIP/2.0/UDP 10.0.0.2:5060;branch=z9hG4bKjjsdfj
      To: <sip:presentity@domain.com>
      From: <sip:presentity@domain.com>;tag=43214321
      Call-ID: 87654321@10.0.0.2
      CSeq: 1 SUBSCRIBE
      Expires: 3600
      Event: presence
      Contact: <sip:pua@domain.com>
      Content-Length: 0
```

M6: The presence agent for presentity@domain.com processes the subscription request and creates a new subscription. A 200 (OK) response is sent to confirm the subscription.

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP 10.0.0.2:5060;branch=z9hG4bKjjsdfj
To: <sip:presentity@domain.com>;tag=abcd1235
From: <sip:watcher@domain.com>;tag=43214321
Call-ID: 87654321@10.0.0.2
CSeq: 1 SUBSCRIBE
Contact: <sip:pa@domain.com>
Expires: 3600
Content-Length: 0
```

M7: In order to complete the process, the presence agent sends the PUA a NOTIFY with the current presence state of the presentity.

```
NOTIFY sip:presentity@domain.com SIP/2.0
Via: SIP/2.0/UDP pa.domain.com;branch=z9hG4bK8sdfk
To: <sip:watcher@domain.com>;tag=abcd1235
From: <sip:presentity@domain.com>;tag=43214321
Call-ID: 87654321@10.0.0.2
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>
      <timestamp>2003-02-01T16:49:29Z</timestamp>
   </tuple>
   <tuple id="desktop">
      <status>
         <basic>open</basic>
      </status>
      <timestamp>2003-02-01T12:21:29Z</timestamp>
   </tuple>
</presence>
```

M9: 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. The timestamp element is updated to indicate the time of the change. The Expires header indicates the desired duration of this soft state. The "entity" attribute of the presence element in the PIDF document matches the To AOR.

```
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: 81818181@pua.domain.com
CSeq: 1 PUBLISH
Expires: 3600
Event: presence
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>
      <timestamp>2003-02-01T17:00:19Z</timestamp>
   </tuple>
</presence>
```

M10: The presence agent receives, and accepts the presence information. The published data is incorporated into the presentity's presence document. A 200 (OK) response is sent to confirm the publication.

```
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: 81818181@pua.domain.com
CSeq: 1 PUBLISH
ETag: "dx200xyz"
Expires: 1800
```

M11: 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.

```
      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=3400
      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>
            <timestamp>2003-02-01T17:00:19Z</timestamp>
         </tuple>
         <tuple id="desktop">
            <status>
               <basic>open</basic>
            </status>
            <timestamp>2003-02-01T12:21:29Z</timestamp>
         </tuple>
      </presence>
```

M12: The watcher confirms receipt of the NOTIFY request.

```
      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
      Content-Length: 0
```

M13: The presence agent also sends a NOTIFY to the PUA.

```
      NOTIFY sip:presentity@domain.com SIP/2.0
      Via: SIP/2.0/UDP presence.domain.com;branch=z9hG4bK4cd42b
      To: <sip:watcher@domain.com>;tag=abcd1235
      From: <sip:presentity@domain.com>;tag=43214321
      Call-ID: 87654321@10.0.0.2
      CSeq: 2 NOTIFY
      Event: presence
      Subscription-State: active; expires=3400
```

```
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>
      <timestamp>2003-02-01T17:00:19Z</timestamp>
   </tuple>
   <tuple id="desktop">
      <status>
         <basic>open</basic>
      </status>
      <timestamp>2003-02-01T12:21:29Z</timestamp>
   </tuple>
</presence>
```

M14: The PUA confirms receipt of the NOTIFY request.

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP presence.domain.com;branch=z9hG4bK4cd42b
To: <sip:watcher@domain.com>;tag=abcd1235
From: <sip:presentity@domain.com>;tag=43214321
Call-ID: 87654321@10.0.0.2
CSeq: 2 NOTIFY
```

11. Open Issues

- Atomicity of publication. Should the segments of event state (presence tuples) be sent in separate PUBLISH requests or is it enough to treat these as implicitly separate publication requests?

- The exact naming convention used for the tuple-ID when publishing tuples.

- In case a refresh publication fails, what should the EPA do? Current suggestion is to query the principal, i.e., "prompt the user", but this is not quite specific.

- Does end-to-end S/MIME integrity protection make sense when an event compositor is used? Does it indicate that the segment should be carried to the watcher intact, or is another mechanism for this needed?

- The examples seem a bit elaborate, and don't even cover the publication refresh case. We should probably work on them.

- Do we need another response code (new or some existing one) for the case when an EPA tries to refresh a publication, but the ESC has lost all version information it has, e.g., after e reboot? This seems a slightly different scenario from the usual "Precondition Failed".

12. Contributors

The original contributors to this specification are:

Ben Campbell
dynamicsoft

Sean Olson
Microsoft

Jon Peterson
Neustar, Inc.

Jonathan Rosenberg
dynamicsoft

Brian Stucker
Nortel Networks, Inc.

13. Changes from "draft-ietf-simple-publish-00"

The following changes were made since the last version:

- Merged with "draft-olson-simple-publish-02"

- Removed usage of Call-ID and CSeq for ordering

- Removed timestamp based versioning

- Added versioning based on entity-tag version information (ETag), and request precondition (If-Match)

- Changed reference to content-indirection as Informative

- Added section for ABNF definitions

- Editorial corrections, restructuring of document to improve

readability

- o Moved the original authors into a new "Contributors" section
- o Added new definitions in Terminology, and clarified EPA and ESC definitions
- o Strengthened the IANA considerations section.
- o Added text for announcing/probing support for publish, namely OPTIONS and "methods" parameter usage.

Normative References

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

[2] Campbell, B., "SIMPLE Presence Publication Requirements", draft-ietf-simple-publish-reqs-00 (work in progress), February 2003.

[3] 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.

[4] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[5] Rosenberg, J., "A Presence Event Package for the Session Initiation Protocol (SIP)", draft-ietf-simple-presence-10 (work in progress), January 2003.

[6] Fujimoto, S. and H. Sugano, "Common Presence and Instant Messaging (CPIM)Presence Information Data Format", draft-ietf-impp-cpim-pidf-07 (work in progress), January 2003.

Informative References

[7] Postel, J. and J. Reynolds, "File Transfer Protocol", STD 9, RFC 959, October 1985.

[8] Fielding, R., Gettys, J., Mogul, J., Nielsen, H., Masinter, L., Leach, P. and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[9] Olson, S., "A Mechanism for Content Indirection in SIP Messages", draft-olson-sip-content-indirect-mech-01 (work in progress), August 2002.

[10] Rosenberg, J., Schulzrinne, H. and P. Kyzivat, "Caller Preferences and Callee Capabilities for the Session Initiation Protocol (SIP)", draft-ietf-sip-callerprefs-08 (work in progress), March 2003.

Author's Address

Aki Niemi (editor)
Nokia
P.O. Box 321
NOKIA GROUP, FIN 00045
Finland

Phone: +358 50 389 1644
EMail: aki.niemi@nokia.com

Intellectual Property Statement

Full Copyright Statement

Acknowledgement

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