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 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 [5] working group.

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 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 |
+-----+    +-----+   +-----+
```

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 [4] 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 Publication Class

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. Indeed, this is a perfectly valid application. But other composition applications may exist where this will not work. For example, a GeoLoc class might get applied across multiple tuples.
1.1.4 Publisher Instance

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 to 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.

1.1.5 Publication Facet

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". For this purpose, we introduce the notion of a publication facet or view. A publisher may choose to annotate a publication with a facet identifier. The facet identifier captures the intended destination or grouping of event state from the publishers point of view. 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. The publication facet defines a view into the composite event state. It is best thought of as metadata 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 notion of a publication facet. In this example, the publisher would actually publish several versions of the event state, each marked with a different facet identifier indicating the destination grouping of the
Like the publication class, the publication facet identifier is arbitrary and a possible candidate for an IANA registry. For example, the "anonymous" facet is reasonable to standardize.

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 otherhand, 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 an 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 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.
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 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.

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 is not 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. 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 and facet 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.

\[
PUBLISHm = \%50.55.42.4C.49.53.48 \text{ ; PUBLISH in caps.}
\]

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.

<table>
<thead>
<tr>
<th>Header Field</th>
<th>where</th>
<th>proxy</th>
<th>PUBLISH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accept</td>
<td>R</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Accept</td>
<td>2xx</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Accept</td>
<td>415</td>
<td>m*</td>
<td>m*</td>
</tr>
<tr>
<td>Accept-Encoding</td>
<td>R</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Accept-Encoding</td>
<td>2xx</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Accept-Encoding</td>
<td>415</td>
<td>m*</td>
<td>m*</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Header Field</th>
<th>R</th>
<th>m</th>
<th>o</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accept-Language</td>
<td>R</td>
<td>m*</td>
<td></td>
</tr>
<tr>
<td>Alert-Info</td>
<td>R</td>
<td>m</td>
<td></td>
</tr>
<tr>
<td>Allow</td>
<td>R</td>
<td>o</td>
<td></td>
</tr>
<tr>
<td>Authentication-Info</td>
<td>R</td>
<td>o</td>
<td></td>
</tr>
<tr>
<td>Call-ID</td>
<td>c</td>
<td>r</td>
<td>m</td>
</tr>
<tr>
<td>Call-Info</td>
<td>ar</td>
<td>o</td>
<td></td>
</tr>
<tr>
<td>Class</td>
<td>R</td>
<td>o</td>
<td></td>
</tr>
<tr>
<td>Contact</td>
<td>R</td>
<td>o</td>
<td></td>
</tr>
<tr>
<td>Content-Disposition</td>
<td>o</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Content-Encoding</td>
<td>o</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Content-Language</td>
<td>o</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Content-Length</td>
<td>ar</td>
<td>t</td>
<td></td>
</tr>
<tr>
<td>Content-Type</td>
<td>*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CSeq</td>
<td>c</td>
<td>r</td>
<td>m</td>
</tr>
<tr>
<td>Date</td>
<td>a</td>
<td>o</td>
<td></td>
</tr>
<tr>
<td>Event</td>
<td>a</td>
<td>m</td>
<td></td>
</tr>
<tr>
<td>Error-Info</td>
<td>300-699</td>
<td>a</td>
<td>o</td>
</tr>
<tr>
<td>Expires</td>
<td>o</td>
<td></td>
<td></td>
</tr>
<tr>
<td>From</td>
<td>c</td>
<td>r</td>
<td>m</td>
</tr>
<tr>
<td>In-Reply-To</td>
<td>R</td>
<td>o</td>
<td></td>
</tr>
<tr>
<td>Max-Forwards</td>
<td>R</td>
<td>amr</td>
<td>m</td>
</tr>
<tr>
<td>Organization</td>
<td>ar</td>
<td>o</td>
<td></td>
</tr>
</tbody>
</table>

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

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.

<table>
<thead>
<tr>
<th>Header Field</th>
<th>where</th>
<th>proxy</th>
<th>PUBLISH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>R</td>
<td>ar</td>
<td>o</td>
</tr>
<tr>
<td>Proxy-Authenticate</td>
<td>407</td>
<td>ar</td>
<td>m</td>
</tr>
<tr>
<td>Proxy-Authenticate</td>
<td>401</td>
<td>ar</td>
<td>o</td>
</tr>
<tr>
<td>Proxy-Authorization</td>
<td>R</td>
<td>dr</td>
<td>o</td>
</tr>
<tr>
<td>Proxy-Require</td>
<td>R</td>
<td>ar</td>
<td>o</td>
</tr>
<tr>
<td>Record-Route</td>
<td>ar</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Reply-To</td>
<td>o</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Require</td>
<td>ar</td>
<td>c</td>
<td></td>
</tr>
<tr>
<td>Retry-After</td>
<td>404,413,480,486</td>
<td>o</td>
<td></td>
</tr>
<tr>
<td></td>
<td>500,503</td>
<td>o</td>
<td></td>
</tr>
<tr>
<td></td>
<td>600,603</td>
<td>o</td>
<td></td>
</tr>
<tr>
<td>Route</td>
<td>R</td>
<td>adr</td>
<td>o</td>
</tr>
<tr>
<td>Server</td>
<td>r</td>
<td>o</td>
<td></td>
</tr>
<tr>
<td>Stream</td>
<td>a</td>
<td>m*</td>
<td></td>
</tr>
<tr>
<td>Subject</td>
<td>R</td>
<td>o</td>
<td></td>
</tr>
<tr>
<td>Timestamp</td>
<td>o</td>
<td></td>
<td></td>
</tr>
<tr>
<td>To</td>
<td>c(1)</td>
<td>r</td>
<td>m</td>
</tr>
<tr>
<td>Unsupported</td>
<td>420</td>
<td>o</td>
<td></td>
</tr>
<tr>
<td>User-Agent</td>
<td>o</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Via</td>
<td>R</td>
<td>amr</td>
<td>m</td>
</tr>
<tr>
<td>Via</td>
<td>rc</td>
<td>dr</td>
<td>m</td>
</tr>
<tr>
<td>Warning</td>
<td>r</td>
<td>dr</td>
<td>o</td>
</tr>
<tr>
<td>WWW-Authenticate</td>
<td>401</td>
<td>ar</td>
<td>m</td>
</tr>
<tr>
<td>WWW-Authenticate</td>
<td>407</td>
<td>ar</td>
<td>o</td>
</tr>
</tbody>
</table>
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 Stream (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 (eg. if an offline indication is processed before an online indication, the user may show up as being online when they are in fact, offline).

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 (Stream) 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 publication stream identifier indicates the publication instance which is the source of the publication.

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.

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. The Call-ID may change freely from one PUBLISH request to the next.

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 a connection oriented or reliable transport 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.

The usage of the Stream header by the compositor is a matter of local policy. For example, the compositor may choose to track the Stream header value for the duration of the publication state for the purposes of correlation in the same way a tag may be used for correlation with a SIP dialog. The compositor may also choose to treat the Stream header as a more persistent identifier for a publication source, much as a URI may be a persistent identifier for a user.

Stream = "Stream" HCOLON word [ "@" word ]

Example:
Stream: 12345678@192.168.1.1

3.4 Facet (Publication Facet) Header

For situations where the publisher wishes to publish different event state depending on the listener for that event, we introduce a Facet (publication facet) header. The Facet header contains a comma separated list of name-addr(s) which indicates the intended target watcher or group of watchers that this event state is being published for. This is an optional header. In the absence of a Facet header, the compositor MAY assume that the event state may be restricted based on local policy at the compositor. Multiple Facet headers are
allowed in a single PUBLISH request. While the most common case will be to include a SIP(S) URI in the Facet header, it is also forseen that the use of URNs to describe watcher roles rather than specific watchers is a possibility. The use of a wildcard ("*" ) indicates that the event state is intended for all watchers.

Facet = "Facet" HCOLON facet-restriction *(COMMA facet-restriction)
facet-restriction = ( name-addr / hostname / "*" )

Examples:
Facet: <sip:joe.smith@domain.com>, localdomain.com
Facet: *

3.5 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.

3.6 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.
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.

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>

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

PUBLISH sip:presententity@domain.com SIP/2.0
Via: SIP/2.0/UDP pua.domain.com;branch=z9hG4bK652hsge
To: <sip:presententity@domain.com>;tag=1a2b3c4d
From: <sip:presententity@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:presententity@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:presententity@domain.com>;tag=1a2b3c4d
From: <sip:presententity@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"?>
<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. IANA Considerations

OPEN ISSUE: What IANA registries are appropriate for the PUBLISH mechanism?
6. Security Considerations

A presence compositor should authenticate publishing User Agents, 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.

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 Facet and Class headers can factor heavily into policy at the compositor. For this reason, it is important to protect the integrity and potentially the privacy of these headers. It is RECOMMENDED that appropriate SIP integrity and privacy measures be taken such as the use of S/MIME and TLS.
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
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.