Network Working Group
Internet-Draft
Intended status: Informational
Expires: February 8, 2008

P. Saint-Andre
XMPP Standards Foundation
A. Houri
IBM
J. Hildebrand
Jabber, Inc.

August 7, 2007

Basic Messaging and Presence Interworking between the Extensible Messaging and Presence Protocol (XMPP) and Session Initiation Protocol (SIP) for Instant Messaging and Presence Leveraging Extensions (SIMPLE)

draft-saintandre-xmpp-simple-10

Status of this Memo

Copyright Notice

Abstract

This document defines a bi-directional protocol mapping for use by gateways that enable the exchange of presence information and single instant messages between systems that implement the Extensible

Messaging and Presence Protocol (XMPP) and those that implement the basic extensions to the Session Initiation Protocol (SIP) for instant messaging and presence.

Table of Contents

1. Introduction

In order to help ensure interworking between instant messaging and presence systems that conform to the requirements of RFC 2779 [IMP-REQS], it is important to clearly define mappings between such protocols. Within the IETF, work has proceeded on two such protocols:

- o Various extensions to the Session Initiation Protocol ([SIP]) for instant messaging and presence, as developed within the SIP for Instant Messaging and Presence Leveraging Extensions (SIMPLE) Working Group; the relevant specifications are [SIP-PRES] for presence and [SIP-IM] for instant messaging
- o The Extensible Messaging and Presence Protocol (XMPP), which consists of a formalization of the core XML streaming protocols developed originally by the Jabber open-source community; the relevant specifications are [XMPP-CORE] for the XML streaming layer and [XMPP-IM] for basic presence and instant messaging extensions

One approach to helping ensure interworking between these protocols is to map each protocol to the abstract semantics described in [CPIM] and [CPP]; that is the approach taken by [SIMPLE-CPIM] and [XMPP-CPIM]. The approach taken in this document is to directly map semantics from one protocol to another (i.e., from SIP/SIMPLE to XMPP and vice-versa).

The mappings specified in this document cover several areas that address basic instant messaging and presence functionality:

- o Mapping of addresses
- o Mapping of single instant messages
- o Mapping of presence subscriptions
- o Mapping of presence notifications
- o Handling of content types
- o Mapping of error conditions

Mapping of more advanced functionality is out of scope for this document; however, the authors will attempt to address such mappings in future documents devoted to one-to-one messaging sessions, multi-user chat, extended presence, etc.

1.1. Architectural Assumptions

Protocol translation between XMPP and SIMPLE could occur in a number of different entities, depending on the architecture of presence and messaging deployments. For example, protocol translation could occur within a multi-protocol server, within a multi-protocol client, or

within a gateway that acts as a dedicated protocol translator.

This document assumes that the protocol translation will occur within a gateway. (This assumption not meant to discourage protocol translation within multi-protocol clients or servers; instead, this assumption is followed mainly to clarify the discussion and examples so that the protocol translation principles can be more easily understood and can be applied by client and server implementors with appropriate modifications to the examples and terminology.) Specifically, we assume that the protocol translation will occur within an "XMPP-to-SIMPLE gateway" that translates XMPP syntax and semantics on behalf of an XMPP service when communicating with SIMPLE services and/or within a "SIMPLE-to-XMPP gateway" that translates SIP syntax and semantics on behalf of a SIMPLE service when communicating with XMPP services.

Although such a gateway could use the [CPIM] and [CPP] specifications to define the common formats into which the protocols are translated for purposes of interworking (as specified in [SIMPLE-CPIM] and [XMPP-CPIM]), this document assumes that a gateway will translate directly from one protocol to the other. We further assume that protocol translation will occur within a gateway in the source domain, so that messages and presence information generated by the user of an XMPP service will be translated by a gateway within the trust domain of that XMPP service, and messages and presence information generated by the user of a SIMPLE service will be translated by a gateway within the trust domain of that SIMPLE service.

An architectural diagram for a typical gateway deployment is shown below, where the entities have the following significance and the "#" character is used to show the boundary of a trust domain:

- romeo@example.net -- a SIMPLE user.
- example.net -- a SIMPLE service.
- s2x.example.net -- a SIMPLE-to-XMPP gateway.
- juliet@example.com -- an XMPP user.
- example.com -- an XMPP service.
- x2s.example.com -- an XMPP-to-SIMPLE gateway.

```
   #####################################################################
   #                               #                                   #
   #         +-- s2x.example.net---#------------- example.com          #
   #         |                     #                |     |            #
   #   example.net -----------------#--- x2s.example.com  |            #
   #       |                       #                      |            #
   #       |                       #                      |            #
   #   romeo@example.net           #              juliet@example.com  #
   #                               #                                   #
   #####################################################################
```

1.2. Terminology

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

2. Addresses

2.1. Overview

The address formats used to identify XMPP entities are different from those used to identify SIP entities. The XMPP address format is specified in [XMPP-CORE]; as specified in [XMPP-IM], instant messaging and presence applications of XMPP must also support 'im:' and 'pres:' URIs as specified in [CPIM] and [CPP] respectively, although such support may simply involve leaving resolution of such addresses up to an XMPP server. The SIP address format for instant messaging is specified in [SIP-IM]; it may use either 'sip:' or 'sips:' URIs as specified in [SIP] or an 'im:' URI as specified in [CPIM]. The SIP address format for presence is specified in [SIP-PRES]; it may use either 'sip:' or 'sips:' URIs as specified in [SIP] or a 'pres:' URI as specified in [CPP].

In this document we describe mappings for addresses of the form <user@domain> only, ignoring (for the purpose of address mapping) any protocol-specific extensions such as XMPP resource identifiers or SIP telephone numbers and passwords. In addition, we have ruled the mapping of domain names as out of scope for now since that is a matter for the Domain Name System; specifically, the issue for interworking between SIP and XMPP relates to the translation of fully internationalized domain names (which the SIP address format does not allow, but which the XMPP address format does allow via [IDNA]) into non-internationalized domain names. Therefore, in the following sections we discuss local-part addresses only (these are called variously "usernames", "instant inboxes", "presentities", and "node

identifiers" in the protocols at issue).

The sip:/sips:, im:/pres:, and XMPP address schemes allow different sets of characters (although all three allow alphanumeric characters and disallow both spaces and control characters). In some cases, characters allowed in one scheme are disallowed in others; these characters must be mapped appropriately in order to ensure interworking across systems.

The local-part address in sip:/sips: URIs inherits from the "userinfo" rule in [URI] with several changes; here we discuss the SIP "user" rule only:

```
   user              =  1*( unreserved / escaped / user-unreserved )
   user-unreserved   =  "&" / "=" / "+" / "$" / "," / ";" / "?" / "/"
   unreserved        =  alphanum / mark
   mark              =  "-" / "_" / "." / "!" / "~" / "*" / "'"
                        / "(" / ")"
```

Here we make the simplifying assumption that the local-part address in im:/pres: URIs inherits from the "dot-atom-text" rule in [RFC2822] rather than the more complicated "local-part" rule:

```
   dot-atom-text =  1*atext *("." 1*atext)
   atext         =  ALPHA / DIGIT / ; Any character except controls,
                    "!" / "#" /     ;  SP, and specials.
                    "$" / "%" /     ;  Used for atoms
                    "&" / "'" /
                    "*" / "+" /
                    "-" / "/" /
                    "=" / "?" /
                    "^" / "_" /
                    "`" / "{" /
                    "|" / "}" /
                    "~"
```

The local-part address in XMPP addresses allows any US-ASCII character except space, controls, and the " & ' / : < > @ characters.

Therefore, following table lists the allowed and disallowed characters in the local-part addresses of each protocol (aside from the alphanumeric, space, and control characters), in order by hexadecimal character number (where the "A" row shows the allowed characters and the "D" row shows the disallowed characters).

Table 1: Allowed and disallowed characters

```
+---+----------------------------------+
| SIP/SIPS CHARACTERS                  |
+---+----------------------------------+
| A | !  $ &'()*+,-./ ; = ?     _    ~ |
| D |  "# %          : < > @[\]^ `{|}  |
+---+----------------------------------+
| IM/PRES CHARACTERS                   |
+---+----------------------------------+
| A | ! #$%&'  *+ - /   = ?    ^_`{|}~ |
| D |  "     ()  , . :;< > @[\]        |
+---+----------------------------------+
| XMPP CHARACTERS                      |
+---+----------------------------------+
| A | ! #$%  ()*+,-.  ; = ? [\]^_`{|}~ |
| D |  "   &'       /: < > @           |
+---+----------------------------------+
```

When transforming a local-part address from one scheme to another, an application SHOULD proceed as follows:

1. Unescape any escaped characters in the source address (e.g., from SIP to XMPP unescape "%2F" to "/" and from XMPP to SIP unescape "\27" to "'").
2. Leave unmodified any characters that are allowed in the destination scheme.
3. Escape any characters that are allowed in the source scheme but reserved in the destination scheme, as escaping is defined for the destination scheme. In particular:
   * Where the destination scheme is a URI (i.e., an im:, pres:, sip:, or sips: URI), each reserved character MUST be percent-encoded to "%hexhex" as specified in Section 2.6 of [URL-GUIDE] (e.g., when transforming from XMPP to SIP, encode "/" as "%2F").
   * Where the destination scheme is a native XMPP address, each reserved character MUST be encoded to "\hexhex" as specified in [XEP-0106] (e.g., when transforming from SIP to XMPP, encode "'" as "\27").

2.2. XMPP to SIP

The following is a high-level algorithm for mapping an XMPP address to a sip:, sips:, im:, or pres: URI:

1. Split XMPP address into node identifier (local-part; mapping described in remaining steps), domain identifier (hostname; mapping is out of scope), and resource identifier (specifier for

particular device or connection; discard this for cross-system interworking).

2. Apply Nodeprep profile of [STRINGPREP] (as specified in [XMPP-CORE]) for canonicalization (OPTIONAL).
3. Translate "\26" to "&", "\27" to "'", and "\2f" to "/" respectively (this is consistent with [XEP-0106]).
4. Determine if the foreign domain supports im: and pres: URIs (discovered via [SRV] lookup as specified in [XMPP-IM]), else assume that the foreign domain supports sip:/sips: URIs.
5. If converting into im: or pres: URI, for each byte, if the byte is in the set (),.;[\] (i.e., the partial complement from Row 3, Column 2 of Table 3 above) or is a UTF-8 character outside the US-ASCII range then transform that byte to %hexhex. If converting into sip: or sips: URI, for each byte, if the byte is in the set #%[\]^`{|} (i.e., the partial complement from Row 3, Column 1 of Table 3 above) or is a UTF-8 character outside the US-ASCII range then transform that byte to %hexhex.
6. Combine resulting local-part with mapped hostname to form local@domain address.
7. Prepend with 'im:' scheme (for XMPP <message/> stanzas) or 'pres:' scheme (for XMPP <presence/> stanzas) if foreign domain supports these, else prepend with 'sip:' or 'sips:' scheme according to local service policy.

2.3. SIP to XMPP

The following is a high-level algorithm for mapping a sip:, sips:, im:, or pres: URI to an XMPP address:

1. Remove URI scheme.
2. Split at the first '@' character into local-part and hostname (mapping the latter is out of scope).
3. Translate %hexhex to equivalent octets.
4. Treat result as a UTF-8 string.
5. Translate "&" to "\26", "'" to "\27", and "/" to "\2f" respectively in order to properly handle the characters disallowed in XMPP addresses but allowed in sip:/sips: URIs and im:/pres: URIs as shown in Column 3 of Table 3 above (this is consistent with [XEP-0106]).
6. Apply Nodeprep profile of [STRINGPREP] (as specified in [XMPP-CORE]) for canonicalization (OPTIONAL).
7. Recombine local-part with mapped hostname to form local@domain address.

3. Instant Messages

3.1. Overview

Both XMPP and IM-aware SIP systems enable entities (often but not necessarily human users) to send "instant messages" to other entities. The term "instant message" usually refers to messages sent between two entities for delivery in close to real time (rather than messages that are stored and forwarded to the intended recipient upon request). Generally there are three kinds of instant message:

- Single messages, which are sent from the sender to the recipient outside the context of any one-to-one chat session or multi-user text conference.
- Chat messages, which are sent from the sender to the recipient in the context of a "messaging session" between the two entities.
- Groupchat messages, which are sent from a sender to multiple recipients in the context of a text conference.

This document covers single messages only, since they form the "lowest common denominator" for instant messaging on the Internet. It is likely that future documents will address one-to-one chat sessions and multi-user chat.

Instant messaging using XMPP message stanzas of type "normal" is specified in [XMPP-IM]. Instant messaging using SIP requests of type MESSAGE (often called "page-mode" messaging) is specified in [SIP-IM].

As described in [XMPP-IM], a single instant message is an XML <message/> stanza of type "normal" sent over an XML stream (since "normal" is the default for the 'type' attribute of the <message/> stanza, the attribute is often omitted). In this document we will assume that such a message is sent from an XMPP client to an XMPP server over an XML stream negotiated between the client and the server, and that the client is controlled by a human user (this is a simplifying assumption introduced for explanatory purposes only; the XMPP sender could be a bot-controlled client, a component such as a workflow application, a server, etc.). Continuing the tradition of Shakespeare examples in XMPP documentation, we will say that the XMPP user has an XMPP address of <juliet@example.com>.

As described in [SIP-IM], a single instant message is a SIP MESSAGE request sent from a SIP user agent to an intended recipient who is most generally referenced by an Instant Message URI of the form <im:user@domain> but who may be referenced by a SIP or SIPS URI of the form <sip:user@domain> or <sips:user@domain> Here again we introduce the simplifying assumption that the user agent is controlled by a human user, whom we shall dub <romeo@example.net>.

3.2. XMPP to SIP

When Juliet wants to send an instant message to Romeo, she interacts with her XMPP client, which generates an XMPP <message/> stanza. The syntax of the <message/> stanza, including required and optional elements and attributes, is defined in [XMPP-IM]. The following is an example of such a stanza:

Example: XMPP user sends message:

```
|  <message from='juliet@example.com/balcony'
|           to='romeo@example.net'>
|    <body>Art thou not Romeo, and a Montague?</body>
|  </message>
```

Upon receiving such a stanza, the XMPP server to which Juliet has connected either delivers it to a local recipient (if the hostname in the 'to' attribute matches one of the hostnames serviced by the XMPP server) or attempts to route it to the foreign domain that services the hostname in the 'to' attribute. Naturally, in this document we assume that the hostname in the 'to' attribute is an IM-aware SIP service hosted by a separate server. As specified in [XMPP-IM], the XMPP server needs to determine the identity of the foreign domain, which it does by performing one or more [SRV] lookups. For message stanzas, the order of lookups recommended by [XMPP-IM] is to first try the "_xmpp-server" service as specified in [XMPP-CORE] and to then try the "_im" service as specified in [IMP-SRV]. Here we assume that the first lookup will fail but that the second lookup will succeed and return a resolution "_im._simple.example.net.", since we have already assumed that the example.net hostname is running a SIP instant messaging service. (Note: The XMPP server may have previously determined that the foreign domain is a SIMPLE server, in which case it would not need to perform the SRV lookups; the caching of such information is a matter of implementation and local service policy, and is therefore out of scope for this document.)

Once the XMPP server has determined that the foreign domain is serviced by a SIMPLE server, it must determine how to proceed. We here assume that the XMPP server contains or has available to it an XMPP-SIMPLE gateway. The XMPP server would then deliver the message stanza to the XMPP-SIMPLE gateway.

The XMPP-SIMPLE gateway is then responsible for translating the XMPP message stanza into a SIP MESSAGE request from the XMPP user to the SIP user:

Example: XMPP user sends message (SIP transformation):

```
|  MESSAGE sip:romeo@example.net SIP/2.0
|  Via: SIP/2.0/TCP x2s.example.com;branch=z9hG4bK776sgdkse
|  Max-Forwards: 70
|  From: sip:juliet@example.com;tag=49583
|  To: sip:romeo@example.net
|  Call-ID: Hr0zny9l3@example.com
|  CSeq: 1 MESSAGE
|  Content-Type: text/plain
|  Content-Length: 35
|
|  Art thou not Romeo, and a Montague?
```

The mapping of XMPP syntax elements to SIP syntax elements SHOULD be as shown in the following table. (Mappings for elements not mentioned are undefined.)

Table 4: Message syntax mapping from XMPP to SIP

XMPP Element or Attribute	SIP Header or Contents
<body/>	body of MESSAGE
<subject/>	Subject
<thread/>	Call-ID
from	From
id	(no mapping)
to	To
type	(no mapping)
xml:lang	Content-Language

3.3. SIP to XMPP

When Romeo wants to send an instant message to Juliet, he interacts with his SIP user agent, which generates a SIP MESSAGE request. The syntax of the MESSAGE request is defined in [SIP-IM]. The following is an example of such a request:

Example: SIP user sends message:

```
|  MESSAGE sip:juliet@example.com SIP/2.0
|  Via: SIP/2.0/TCP s2x.example.net;branch=z9hG4bKeskdgs677
|  Max-Forwards: 70
|  From: sip:romeo@example.net;tag=38594
|  To: sip:juliet@example.com
|  Call-ID: M4spr4vdu@example.net
|  CSeq: 1 MESSAGE
|  Content-Type: text/plain
|  Content-Length: 44
|
|  Neither, fair saint, if either thee dislike.
```

Section 5 of [SIP-IM] stipulates that a SIP User Agent presented with an im: URI should resolve it to a sip: or sips: URI. Therefore we assume that the To header of a request received by a SIMPLE-XMPP gateway will contain a sip: or sips: URI. The gateway SHOULD resolve that address to an im: URI for SIP MESSAGE requests, then follow the rules in [IMP-SRV] regarding the "_im" SRV service for the target domain contained in the To header. If SRV address resolution fails for the "_im" service, the gateway MAY attempt a lookup for the "_xmpp-server" service as specified in [XMPP-CORE] or MAY return an error to the sender (the SIP "502 Bad Gateway" error seems most appropriate; see Section 7 for details). If SRV address resolution succeeds, the gateway is responsible for translating the request into an XMPP message stanza from the SIP user to the XMPP user and returning a SIP "200 OK" message to the sender:

Example: SIP user sends message (XMPP transformation):

```
|  <message from='romeo@example.net'
|           to='juliet@example.com'>
|    <body>Neither, fair saint, if either thee dislike.</body>
|  </message>
```

The mapping of SIP syntax elements to XMPP syntax elements SHOULD be as shown in the following table. (Mappings for elements not mentioned in the foregoing table are undefined.)

Table 5: Message syntax mapping from SIP to XMPP

SIP Header or Contents	XMPP Element or Attribute
Call-ID	<thread/>
Content-Language	xml:lang
CSeq	(no mapping)
From	from
Subject	<subject/>
To	to
body of MESSAGE	<body/>

Note: When transforming SIP page-mode messages, a SIMPLE-XMPP gateway SHOULD specify no XMPP 'type' attribute or a 'type' attribute whose value is "normal" (alternatively, the value of the 'type' attribute MAY be "chat", although it SHOULD NOT be "headline" and MUST NOT be "groupchat").

Note: See the Content Types (Section 6) of this document regarding handling of SIP message bodies that contain content types other than plain text.

4. Presence Subscriptions

4.1. Overview

Both XMPP and presence-aware SIP systems enable entities (often but not necessarily human users) to subscribe to the presence of other entities. XMPP presence subscriptions are specified in [XMPP-IM]. Presence subscriptions using a SIP event package for presence are specified in [SIP-PRES].

As described in [XMPP-IM], XMPP presence subscriptions are managed using XMPP presence stanzas of type "subscribe", "subscribed", "unsubscribe", and "unsubscribed". The main subscription states are "none" (neither the user nor the contact is subscribed to the other's presence information), "from" (the user has a subscription from the contact), "to" (the user has a subscription to the contact's presence information), and "both" (both user and contact are subscribed to each other's presence information).

As described in [SIP-PRES], SIP presence subscriptions are managed through the use of SIP SUBSCRIBE events sent from a SIP user agent to an intended recipient who is most generally referenced by an Instant Message URI of the form <pres:user@domain> but who may be referenced

by a SIP or SIPS URI of the form <sip:user@domain> or <sips:user@domain>.

The subscription models underlying XMPP and SIP are quite different. For instance, XMPP presence subscriptions are long-lived (indeed permanent if not explicitly cancelled), whereas SIP presence subscriptions are short-lived (the default time to live of a SIP presence subscription is 3600 seconds, as specified in Section 6.4 of [SIP-PRES]). These differences are addressed below.

4.2. XMPP to SIP

4.2.1. Establishing

An XMPP user initiates a subscription by sending a subscription request to another entity (conventionally called a "contact"), which request the contact either accepts or declines. If the contact accepts the request, the user will have a subscription to the contact's presence information until (1) the user unsubscribes or (2) the contact cancels the subscription. The subscription request is encapsulated in a presence stanza of type "subscribe":

Example: XMPP user subscribes to SIP contact:

```
|  <presence from='juliet@example.com'
|            to='romeo@example.net'
|            type='subscribe'/>
```

Upon receiving such a stanza, the XMPP server to which Juliet has connected needs to determine the identity of the foreign domain, which it does by performing one or more [SRV] lookups. For presence stanzas, the order of lookups recommended by [XMPP-IM] is to first try the "_xmpp-server" service as specified in [XMPP-CORE] and to then try the "_pres" service as specified in [IMP-SRV]. Here we assume that the first lookup will fail but that the second lookup will succeed and return a resolution "_pres._simple.example.net.", since we have already assumed that the example.net hostname is running a SIP presence service.

Once the XMPP server has determined that the foreign domain is serviced by a SIMPLE server, it must determine how to proceed. We here assume that the XMPP server contains or has available to it an XMPP-SIMPLE gateway. The XMPP server would then deliver the presence stanza to the XMPP-SIMPLE gateway.

The XMPP-SIMPLE gateway is then responsible for translating the XMPP subscription request into a SIP SUBSCRIBE request from the XMPP user to the SIP user:

Example: XMPP user subscribes to SIP contact (SIP transformation):

```
|  SUBSCRIBE sip:romeo@example.net SIP/2.0
|  Via: SIP/2.0/TCP x2s.example.com;branch=z9hG4bKna998sk
|  From: <sip:juliet@example.com>;tag=ffd2
|  To: <sip:romeo@example.net>
|  Call-ID: l04th3s1p@example.com
|  Event: presence
|  Max-Forwards: 70
|  CSeq: 123 SUBSCRIBE
|  Contact: <sip:sipgate.example.com;transport=tcp>
|  Accept: application/pidf+xml
|  Expires: 3600
|  Content-Length: 0
```

The SIP user then SHOULD send a response indicating acceptance of the subscription request:

Example: SIP accepts subscription request:

```
|  SIP/2.0 200 OK
|  Via: SIP/2.0/TCP s2x.example.net;branch=z9hG4bKna998sk
|  From: <sip:romeo@example.net>;tag=ffd2
|  To: <sip:juliet@example.com>;tag=j89d
|  Call-ID: l04th3s1p@example.com
|  CSeq: 234 SUBSCRIBE
|  Contact: <sip:simple.example.net;transport=tcp>
|  Expires: 3600
|  Content-Length: 0
```

In accordance with [SIP-EVENT], the XMPP-SIMPLE gateway should consider the subscription state to be "neutral" until it receives a NOTIFY message. Therefore the SIP user or SIP-XMPP gateway at the SIP user's domain SHOULD immediately send a NOTIFY message containing a "Subscription-State" header whose value contains the string "active" (see Section 5).

Example: SIP user sends presence notification:

```
|  NOTIFY sip:192.0.2.1 SIP/2.0
|  Via: SIP/2.0/TCP simple.example.net;branch=z9hG4bKna998sk
|  From: <sip:romeo@example.net>;tag=yt66
|  To: <sip:juliet@example.com>;tag=bi54
|  Call-ID: l04th3s1p@example.com
|  Event: presence
|  Subscription-State: active;expires=499
|  Max-Forwards: 70
|  CSeq: 8775 NOTIFY
|  Contact: <sip:simple.example.net;transport=tcp>
|  Content-Type: application/pidf+xml
|  Content-Length: 193
|
|  <?xml version='1.0' encoding='UTF-8'?>
|  <presence xmlns='urn:ietf:params:xml:ns:pidf'
|            entity='pres:romeo@example.net'>
|    <tuple id='orchard'>
|      <status>
|        <basic>open</basic>
|      </status>
|    </tuple>
|  </presence>
```

Upon receiving the first NOTIFY with a subscription state of active, the XMPP-SIMPLE gateway MUST generate a presence stanza of type "subscribed":

Example: XMPP user receives acknowledgement from SIP contact:

```
|  <presence to='romeo@example.net'
|            from='juliet@example.com'
|            type='subscribed'/>
```

For information about handling of the NOTIFY message, see Section 5.

4.2.2. Refreshing

It is the responsibility of the XMPP-SIMPLE gateway to set the value of the "Expires" header and to periodically renew the subscription on the SIMPLE side of the gateway so that the subscription appears to be permanent to the XMPP user (e.g., the XMPP-SIMPLE gateway SHOULD send a new SUBSCRIBE request to the SIP user whenever the XMPP user sends initial presence to its XMPP server, i.e., upon initiating a presence session with the XMPP server). See the Security Considerations (Section 8) of this document for important information and requirements regarding the security implications of this

functionality.

4.2.3. Cancelling

At any time after subscribing, the XMPP user may unsubscribe from the contact's presence. This is done by sending a presence stanza of type "unsubscribe":

Example: XMPP user unsubscribes from SIP contact:

```
|  <presence from='juliet@example.com'
|            to='romeo@example.net'
|            type='unsubscribe'/>
```

The XMPP-SIMPLE gateway is responsible for translating the unsubscribe command into a SIP SUBSCRIBE request with the "Expires" header set to a value of zero:

Example: XMPP user unsubscribes from SIP contact (SIP transformation):

```
|  SUBSCRIBE sip:romeo@example.net SIP/2.0
|  Via: SIP/2.0/TCP s2x.example.net;branch=z9hG4bKna998sk
|  From: <sip:juliet@example.com>;tag=j89d
|  To: <sip:romeo@example.net>;tag=xfg9
|  Call-ID: 1ckm32@example.com
|  Event: presence
|  Max-Forwards: 70
|  CSeq: 789 SUBSCRIBE
|  Contact: <sip:x2s.example.com;transport=tcp>
|  Accept: application/pidf+xml
|  Expires: 0
|  Content-Length: 0
```

Upon sending the transformed unsubscribe, the XMPP-SIMPLE gateway SHOULD a presence stanza of type "unsubscribed" to the XMPP user:

Example: XMPP user receives unsubscribed notification:

```
|  <presence to='romeo@example.net'
|            from='juliet@example.com'
|            type='unsubscribed'/>
```

4.3. SIP to XMPP

4.3.1. Establishing

A SIP user initiates a subscription to a contact's presence information by sending a SIP SUBSCRIBE request to the contact. The following is an example of such a request:

Example: SIP user subscribes to XMPP contact:

```
|  SUBSCRIBE sip:juliet@example.com SIP/2.0
|  Via: SIP/2.0/TCP s2x.example.net;branch=z9hG4bKna998sk
|  From: <sip:romeo@example.net>;tag=xfg9
|  To: <sip:juliet@example.com>;tag=ur93
|  Call-ID: 4wcm0n@example.net
|  Event: presence
|  Max-Forwards: 70
|  CSeq: 263 SUBSCRIBE
|  Contact: <sip:simple.example.net;transport=tcp>
|  Accept: application/pidf+xml
|  Content-Length: 0
```

Upon receiving such a request, a SIMPLE-XMPP gateway is responsible for translating it into an XMPP subscription request from the SIP user to the XMPP user:

Example: SIP user subscribes to XMPP contact (XMPP transformation):

```
|  <presence from='romeo@example.net'
|            to='juliet@example.com'
|            type='subscribe'/>
```

Notice that the "Expires" header was not included in the SUBSCRIBE request; this means that the default value of 3600 (i.e., 3600 seconds = 1 hour) applies.

4.3.2. Refreshing

It is the responsibility of the SIMPLE-XMPP gateway to properly handle the difference between short-lived SIP presence subscriptions and long-lived XMPP presence subscriptions. The gateway has two options when the SIP user's subscription expires:

- o Cancel the subscription (i.e., treat it as temporary) and send an XMPP presence stanza of type "unsubscribe" to the XMPP contact; this honors the SIP semantic but will seem rather odd to the XMPP contact.
- o Maintain the subscription (i.e., treat it as long-lived) and (1) send a SIP NOTIFY request to the SIP user containing a PIDF document specifying that the XMPP contact now has a basic status

of "closed", including a Subscription-State of "terminated" and (2) send an XMPP presence stanza of type "unavailable" to the XMPP contact; this violates the letter of the SIP semantic but will seem more natural to the XMPP contact.

Which of these options the SIMPLE-XMPP gateway chooses is up to the implementation.

If the implementation chooses the first option, the protocol generated would be as follows:

Example: SIP subscription expires (treated as temporary by gateway):

```
   |  <presence from='romeo@example.net'
   |            to='juliet@example.com'
   |            type='unsubscribe'/>
```

If the implementation chooses the second option, the protocol generated would be as follows:

Example: SIP subscription expires (treated as long-lived by gateway):

```
   |  NOTIFY sip:192.0.2.2 SIP/2.0
   |  Via: SIP/2.0/TCP s2x.example.net;branch=z9hG4bKna998sk
   |  From: <sip:juliet@example.com>;tag=ur93
   |  To: <sip:romeo@example.net>;tag=pq72
   |  Call-ID: j4s0h4vny@example.com
   |  Event: presence
   |  Subscription-State: terminated;reason=timeout
   |  Max-Forwards: 70
   |  CSeq: 232 NOTIFY
   |  Contact: <sip:sipgate.example.com;transport=tcp>
   |  Content-Type: application/pidf+xml
   |  Content-Length: 194
   |
   |  <?xml version='1.0' encoding='UTF-8'?>
   |  <presence xmlns='urn:ietf:params:xml:ns:pidf'
   |            entity='pres:juliet@example.com'>
   |    <tuple id='balcony'>
   |      <status>
   |        <basic>closed</basic>
   |      </status>
   |    </tuple>
   |  </presence>
```

Example: SIP subscription expires (treated as long-lived by gateway):

```
|  <presence from='romeo@example.net'
|            to='juliet@example.com'
|            type='unavailable'/>
```

4.3.3. Cancelling

At any time, the SIP user may cancel the subscription by sending a SUBSCRIBE message whose "Expires" header is set to a value of zero ("0"):

Example: SIP user cancels subscription:

```
|  SUBSCRIBE sip:192.0.2.1 SIP/2.0
|  Via: SIP/2.0/TCP simple.example.net;branch=z9hG4bKna998sk
|  From: <sip:romeo@example.net>;tag=yt66
|  To: <sip:juliet@example.com>;tag=bi54
|  Call-ID: 1tsn1ce@example.net
|  Event: presence
|  Max-Forwards: 70
|  CSeq: 8775 SUBSCRIBE
|  Contact: <sip:simple.example.net;transport=tcp>
|  Expires: 0
|  Content-Length: 0
```

As above, upon receiving such a request, a SIMPLE-XMPP gateway is responsible for doing one of the following:

- Cancel the subscription (i.e., treat it as temporary) and send an XMPP presence stanza of type "unsubscribe" to the XMPP contact.
- Maintain the subscription (i.e., treat it as long-lived) and (1) send a SIP NOTIFY request to the SIP user containing a PIDF document specifying that the XMPP contact now has a basic status of "closed", (2) send a SIP SUBSCRIBE request to the SIP user with an "Expires" header set to a value of "0" (zero) when it receives XMPP presence of type "unavailable" from the XMPP contact, and (3) send an XMPP presence stanza of type "unavailable" to the XMPP contact.

5. Presence Notifications

5.1. Overview

Both XMPP and presence-aware SIP systems enable entities (often but not necessarily human users) to send presence notifications to other entities. At a minimum, the term "presence" refers to information

about an entity's availability for communication on a network (on/off), often supplemented by information that further specifies the entity's communications context (e.g., "do not disturb"). Some systems and protocols extend this notion even further and refer to any relatively ephemeral information about an entity as a kind of presence; categories of such "extended presence" include geographical location (e.g., GPS coordinates), user mood (e.g., grumpy), user activity (e.g., walking), and ambient environment (e.g., noisy). In this document, we focus on the "least common denominator" of network availability only, although future documents may address broader notions of presence, including extended presence.

Presence using XMPP presence stanzas of type "available" or "unavailable" is specified in [XMPP-IM]. SIP presence using a SIP event package for presence is specified in [SIP-PRES].

As described in [XMPP-IM], presence information about an entity is communicated by means of an XML <presence/> stanza sent over an XML stream. In this document we will assume that such a presence stanza is sent from an XMPP client to an XMPP server over an XML stream negotiated between the client and the server, and that the client is controlled by a human user (again, this is a simplifying assumption introduced for explanatory purposes only). In general, XMPP presence is sent by the user to the user's server and then broadcasted to all entities who are subscribed to the user's presence information.

As described in [SIP-PRES], presence information about an entity is communicated by means of a SIP NOTIFY event sent from a SIP user agent to an intended recipient who is most generally referenced by an Instant Message URI of the form <pres:user@domain> but who may be referenced by a SIP or SIPS URI of the form <sip:user@domain> or <sips:user@domain>. Here again we introduce the simplifying assumption that the user agent is controlled by a human user.

5.2. XMPP to SIP

When Juliet interacts with her XMPP client to modify her presence information (or when her client automatically updates her presence information, e.g. via an "auto-away" feature), her client generates an XMPP <presence/> stanza. The syntax of the <presence/> stanza, including required and optional elements and attributes, is defined in [XMPP-IM]. The following is an example of such a stanza:

Example: XMPP user sends presence notification:

```
|  <presence from='juliet@example.com/balcony'/>
```

Upon receiving such a stanza, the XMPP server to which Juliet has

connected broadcasts it to all subscribers who are authorized to receive presence notifications from Juliet (this is similar to the SIP NOTIFY method). For each subscriber, broadcasting the presence notification involves either delivering it to a local recipient (if the hostname in the subscriber's address matches one of the hostnames serviced by the XMPP server) or attempting to route it to the foreign domain that services the hostname in the subscriber's address. Naturally, in this document we assume that the hostname is a SIP presence service hosted by a separate server. As specified in [XMPP-IM], the XMPP server needs to determine the identity of the foreign domain, which it does by performing one or more [SRV] lookups. For presence stanzas, the order of lookups recommended by [XMPP-IM] is to first try the "_xmpp-server" service as specified in [XMPP-CORE] and to then try the "_pres" service as specified in [IMP-SRV]. Here we assume that the first lookup will fail but that the second lookup will succeed and return a resolution "_pres._simple.example.net.", since we have already assumed that the example.net hostname is running a SIP presence service. (Note: The XMPP server may have previously determined that the foreign domain is a SIMPLE server, e.g., when it sent a SIP SUBSCRIBE to the SIP user when Juliet sent initial presence to the XMPP server, in which case it would not need to perform the SRV lookups; the caching of such information is a matter of implementation and local service policy, and is therefore out of scope for this document.)

Once the XMPP server has determined that the foreign domain is serviced by a SIMPLE server, it must determine how to proceed. We here assume that the XMPP server contains or has available to it an XMPP-SIMPLE gateway. The XMPP server would then deliver the presence stanza to the XMPP-SIMPLE gateway.

The XMPP-SIMPLE gateway is then responsible for translating the XMPP presence stanza into a SIP NOTIFY request and included PIDF document from the XMPP user to the SIP user.

Example: XMPP user sends presence notification (SIP transformation):

```
|  NOTIFY sip:192.0.2.2 SIP/2.0
|  Via: SIP/2.0/TCP x2s.example.com;branch=z9hG4bKna998sk
|  From: <sip:juliet@example.com>;tag=gh19
|  To: <sip:romeo@example.net>;tag=yt66
|  Call-ID: j4s0h4vny@example.com
|  Event: presence
|  Subscription-State: active;expires=599
|  Max-Forwards: 70
|  CSeq: 157 NOTIFY
|  Contact: <sip:sipgate.example.com;transport=tcp>
|  Content-Type: application/pidf+xml
|  Content-Length: 192
|
|  <?xml version='1.0' encoding='UTF-8'?>
|  <presence xmlns='urn:ietf:params:xml:ns:pidf'
|            entity='pres:juliet@example.com'>
|    <tuple id='balcony'>
|      <status>
|        <basic>open</basic>
|      </status>
|    </tuple>
|  </presence>
```

The mapping of XMPP syntax elements to SIP syntax elements SHOULD be as shown in the following table. (Mappings for elements not mentioned are undefined.)

Table 6: Presence syntax mapping from XMPP to SIP

XMPP Element or Attribute	SIP Header or PIDF Data
<presence/> stanza	"Event: presence" [1]
XMPP resource identifer	tuple 'id' attribute
from	From
id	Call-ID
to	To
type	basic status [2][3]
xml:lang	Content-Language
<priority/>	PIDF priority for tuple
<show/>	(no mapping)
<status/>	note [4]

Note the following regarding these mappings:

1. Only a presence stanza that lacks a 'type' attribute or whose 'type' attribute has a value of "unavailable" should be mapped by an XMPP-SIMPLE gateway to a SIP NOTIFY request, since those are the only presence stanzas that represent notifications.
2. Because the lack of a 'type' attribute indicates that an XMPP entity is available for communications, the gateway SHOULD map that information to a PIDF <basic/> status of "open". Because a 'type' attribute with a value of "unavailable" indicates that an XMPP entity is not available for communications, the gateway SHOULD map that information to a PIDF <basic/> status of "closed".
3. When the XMPP-SIMPLE gateway receives XMPP presence of type "unavailable" from the XMPP contact, it SHOULD (1) send a SIP NOTIFY request to the SIP user containing a PIDF document specifying that the XMPP contact now has a basic status of "closed" and (2) send a SIP SUBSCRIBE request to the SIP user with an "Expires" header set to a value of "0" (zero).
4. The character data of the XMPP <status/> element MAY be mapped to the character data of the PIDF <note/> element.

5.3. SIP to XMPP

When Romeo changes his presence, his SIP user agent generates a SIP NOTIFY request for any active subscriptions. The syntax of the NOTIFY request is defined in [SIP-PRES]. The following is an example of such a request:

Example: SIP user sends presence notification:

```
|  NOTIFY sip:192.0.2.1 SIP/2.0
|  Via: SIP/2.0/TCP simple.example.net;branch=z9hG4bKna998sk
|  From: <sip:romeo@example.net>;tag=yt66
|  To: <sip:juliet@example.com>;tag=bi54
|  Call-ID: j0sj4sv1m@example.net
|  Event: presence
|  Subscription-State: active;expires=499
|  Max-Forwards: 70
|  CSeq: 8775 NOTIFY
|  Contact: <sip:simple.example.net;transport=tcp>
|  Content-Type: application/pidf+xml
|  Content-Length: 193
|
|  <?xml version='1.0' encoding='UTF-8'?>
|  <presence xmlns='urn:ietf:params:xml:ns:pidf'
|            entity='pres:romeo@example.net'>
|    <tuple id='orchard'>
|      <status>
|        <basic>open</basic>
|      </status>
|    </tuple>
|  </presence>
```

Upon receiving such a request, a SIMPLE-XMPP gateway is responsible for translating it into an XMPP presence stanza from the SIP user to the XMPP user:

Example: SIP user sends presence notification (XMPP transformation):

```
|  <presence from='romeo@example.net'
|            to='juliet@example.com/balcony'
|            type='unavailable'/>
```

The mapping of SIP syntax elements to XMPP syntax elements SHOULD be as shown in the following table. (Mappings for elements not mentioned are undefined.)

Table 7: Presence syntax mapping from SIP to XMPP

SIP Header or PIDF Data	XMPP Element or Attribute
basic status	type [1]
Content-Language	xml:lang
CSeq	id (OPTIONAL)
From	from
priority for tuple	<priority/>
To	to
body of MESSAGE	<body/>

Note the following regarding these mappings:

1. A PIDF basic status of "open" SHOULD be mapped to no 'type' attribute, and a PIDF basic status of "closed" SHOULD be mapped to a 'type' attribute whose value is "unavailable".

6. Content Types

SIP requests of type MESSAGE may contain essentially any content type and SIP requests of type NOTIFY normally contain presence information encapsulated using the "application/pidf+xml" content type. The recommended procedures for SIMPLE-to-XMPP gateways to use in handling these content types are specified in the following sections.

6.1. Messages

A SIMPLE-to-XMPP gateway MUST process SIP messages that contain message bodies of type "text/plain" and MUST encapsulate such message bodies as the XML character data of the XMPP <body/> element.

A SIMPLE-to-XMPP gateway SHOULD process SIP messages that contain message bodies of type "text/html"; if so, a gateway MUST transform the "text/html" content into XHTML content that conforms to the XHTML 1.0 Integration Set specified in [XEP-0071].

A SIMPLE-to-XMPP gateway MAY process SIP messages that contain message bodies of types other than "text/plain" and "text/html" but handling of such content types is a matter of implementation.

6.2. Presence

The "application/pidf+xml' content type is specified in [PIDF]. The Presence Information Data Format defines a common data format for

presence protocols that conform to the Common Profile for Presence ([CPP]), enabling presence information to be transferred across CPP-compliant protocol boundaries without modification, with attendant benefits for end-to-end encryption and performance. Because the syntax for the "application/pidf+xml" content type is Extensible Markup Language ([XML]), it is straightforward to send PIDF data over the Extensible Messaging and Presence Protocol ([XMPP-CORE]), since XMPP is simply an XML streaming protocol.

In addition to following the syntax mappings specified in Section 5, a SIMPLE-to-XMPP gateway MAY encapsulate PIDF data within an "extended namespace" contained in an XMPP presence stanza. The RECOMMENDED method is to include the PIDF <presence/> element as a child of the XMPP <presence/> stanza. Although it may appear that this would be potentially confusing, the inclusion of the 'urn:ietf:params:xml:ns:pidf' namespace ensures that PIDF data is kept separate from XMPP presence data (in accordance with [XML-NAMES]). The following is a simple example of encapsulating PIDF data within an "extended namespace" in XMPP:

A basic example of PIDF over XMPP:

```
<presence from='romeo@example.net/orchard' xml:lang='en'>
  <show>dnd</show>
  <status>Wooing Juliet</status>
  <presence xmlns='urn:ietf:params:xml:ns:pidf'
            entity='pres:romeo@example.net'>
    <tuple id='orchard'>
      <status>
        <basic>open</basic>
      </status>
    </tuple>
  </presence>
</presence>
```

7. Error Conditions

SIP response codes are specified in [SIP] and XMPP error conditions are specified in [XMPP-CORE].

7.1. XMPP to SIP

Table 8: Mapping of XMPP error conditions to SIP response codes

XMPP Error Condition	SIP Response Code
<bad-request/>	400
<conflict/>	400
<feature-not-implemented/>	501
<forbidden/>	403
<gone/>	410
<internal-server-error/>	500
<item-not-found/>	404
<jid-malformed/>	484
<not-acceptable/>	406
<not-allowed/>	405
<not-authorized/>	401
<payment-required/>	402
<recipient-unavailable/>	480
<redirect/>	300
<registration-required/>	407
<remote-server-not-found/>	502
<remote-server-timeout/>	504
<resource-constraint/>	500
<service-unavailable/>	503
<subscription-required/>	407
<undefined-condition/>	400
<unexpected-request/>	491

7.2. SIP to XMPP

The mapping of SIP response codes to XMPP error conditions SHOULD be as follows (note that XMPP does not include 100-series or 200-series response codes, only error conditions):

Table 9: Mapping of SIP response codes to XMPP error conditions

SIP Response Code	XMPP Error Condition
300	<redirect/>
301	<gone/>
302	<redirect/>
305	<redirect/>
380	<not-acceptable/>
400	<bad-request/>
401	<not-authorized/>
402	<payment-required/>
403	<forbidden/>
404	<item-not-found/>
405	<not-allowed/>
406	<not-acceptable/>
407	<registration-required/>
408	<service-unavailable/>
410	<gone/>
413	<bad-request/>
414	<bad-request/>
415	<bad-request/>
416	<bad-request/>
420	<bad-request/>
421	<bad-request/>
423	<bad-request/>
480	<recipient-unavailable/>
481	<item-not-found/>
482	<not-acceptable/>
483	<not-acceptable/>
484	<jid-malformed/>
485	<item-not-found/>
486	<service-unavailable/>
487	<service-unavailable/>
488	<not-acceptable/>
491	<unexpected-request/>
493	<bad-request/>
500	<internal-server-error/>
501	<feature-not-implemented/>
502	<remote-server-not-found/>
503	<service-unavailable/>
504	<remote-server-timeout/>
505	<not-acceptable/>
513	<bad-request/>
600	<service-unavailable/>
603	<service-unavailable/>
604	<item-not-found/>
606	<not-acceptable/>

8. Security Considerations

Detailed security considerations for instant messaging and presence protocols are given in [IMP-REQS], specifically in Sections 5.1 through 5.4. Detailed security considerations for XMPP are given in [XMPP-CORE]. Detailed security considerations for SIP-based messaging are given in [SIP-IM] and for SIP-based presence are given in [SIP-PRES] (see also the security considerations for the Session Initiation Protocol given in [SIP]).

This document specifies methods for exchanging instant messages and presence information through a gateway that translates between SIP and XMPP. Such a gateway MUST be compliant with the minimum security requirements of the instant messaging and presence protocols for which it translates (i.e., SIP and XMPP). The introduction of gateways to the security model of instant messaging and presence specified in [IMP-REQS] introduces some new risks. In particular, end-to-end security properties (especially confidentiality and integrity) between instant messaging and presence user agents that interface through a SIMPLE-XMPP gateway can be provided only if common formats are supported. Specification of those common formats is out of scope for this document, although it is recommended to use [MSGFMT] for instant messages and [PIDF] for presence.

[IMP-REQS] requires that conformant technologies shall include methods for blocking communications from unwanted addresses. Such blocking is the responsibility of conformant technology (e.g., XMPP or SIP) and is out of scope for this memo.

The mismatch between long-lived XMPP presence subscriptions and short-lived SIP presence subscriptions introduces the possibility of an amplification attack launched from the XMPP network against a SIP presence server. To help prevent such an attack, access to an XMPP-SIMPLE gateway that is hosted on the XMPP network SHOULD be restricted to XMPP users associated with a single domain or trust realm (e.g., a gateway hosted at simple.example.com should allow only users within the example.com domain to access the gateway, not users within example.org, example.net, or any other domain); if a SIP presence server receives communications through an XMPP-SIMPLE gateway from users who are not associated with a domain that is so related to the hostname of the gateway, it MAY (based on local service provisioning) refuse to service such users or refuse to communicate with the gateway. Furthermore, whenever an XMPP-SIMPLE gateway seeks to refresh an XMPP user's long-lived subscription to a SIP user's presence, it MUST first send an XMPP <presence/> stanza of type "probe" from the address of the gateway to the "bare JID" (user@domain.tld) of the XMPP user, to which the user's XMPP server MUST respond in accordance with [XMPP-IM]; however, the administrator

of an XMPP-SIMPLE gateway MAY (based on local service provisioning) exempt "known good" XMPP servers from this check (e.g., the XMPP server associated with the XMPP-SIMPLE gateway as described above).

9. Acknowledgements

The authors wish to thank Nathaniel Borenstein and Rohan Mahy for suggestions and encouragement; Daniel-Constantin Mierla for earlier work on SIMPLE-XMPP interworking; Jack Erwin, Tory Patnoe, and Sandeep Sharma for feedback based on implementation experience; and Dave Cridland, Johann Daigremont, Alan Johnston, Benny Prijono, and Adam Roach for their helpful comments.

10. References

10.1. Normative References

[IMP-SRV]   Peterson, J., "Address Resolution for Instant Messaging and Presence", RFC 3861, August 2004.

[PIDF]   Sugano, H., Fujimoto, S., Klyne, G., Bateman, A., Carr, W., and J. Peterson, "Presence Information Data Format (PIDF)", RFC 3863, August 2004.

[SIP]   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.

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

[SIP-IM]   Campbell, B., Rosenberg, J., Schulzrinne, H., Huitema, C., and D. Gurle, "Session Initiation Protocol (SIP) Extension for Instant Messaging", RFC 3428, December 2002.

[SIP-PRES]
Rosenberg, J., "A Presence Event Package for the Session Initiation Protocol (SIP)", RFC 3856, August 2004.

[SRV]   Gulbrandsen, A., Vixie, P., and L. Esibov, "A DNS RR for specifying the location of services (DNS SRV)", RFC 2782, February 2000.

[STRINGPREP]

Hoffman, P. and M. Blanchet, "Preparation of Internationalized Strings ("STRINGPREP")", RFC 3454, December 2002.

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

[URI] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

[URL-GUIDE]
Hansen, T., Hardie, T., and L. Masinter, "Guidelines and Registration Procedures for New URI Schemes", RFC 4395, February 2006.

[XML] Bray, T., Paoli, J., Sperberg-McQueen, C., and E. Maler, "Extensible Markup Language (XML) 1.0 (2nd ed)", W3C REC-xml, October 2000, <http://www.w3.org/TR/REC-xml>.

[XML-NAMES]
Bray, T., Hollander, D., and A. Layman, "Namespaces in XML", W3C REC-xml-names, January 1999, <http://www.w3.org/TR/REC-xml-names>.

[XMPP-CORE]
Saint-Andre, P., "Extensible Messaging and Presence Protocol (XMPP): Core", RFC 3920, October 2004.

[XMPP-IM] Saint-Andre, P., "Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence", RFC 3921, October 2004.

10.2. Informative References

[CPIM] Peterson, J., "Common Profile for Instant Messaging (CPIM)", RFC 3860, August 2004.

[CPP] Peterson, J., "Common Profile for Presence (CPP)", RFC 3859, August 2004.

[IDNA] Faltstrom, P., Hoffman, P., and A. Costello, "Internationalizing Domain Names in Applications (IDNA)", RFC 3490, March 2003.

[IMP-REQS]
Day, M., Aggarwal, S., and J. Vincent, "Instant Messaging / Presence Protocol Requirements", RFC 2779,

February 2000.

[RFC2822] Resnick, P., "Internet Message Format", RFC 2822, April 2001.

[XEP-0071] Saint-Andre, P., "XHTML-IM", XSF XEP 0071, January 2006.

[XEP-0106] Saint-Andre, P. and J. Hildebrand, "JID Escaping", XSF XEP 0106, May 2005.

[MSGFMT] Klyne, G. and D. Atkins, "Common Presence and Instant Messaging (CPIM): Message Format", RFC 3862, August 2004.

[SIMPLE-CPIM] Rosenberg, J. and B. Campbell, "CPIM Mapping of SIMPLE Presence and Instant Messaging", draft-ietf-simple-cpim-mapping-01 (work in progress), June 2002.

[XMPP-CPIM] Saint-Andre, P., "Mapping the Extensible Messaging and Presence Protocol (XMPP) to Common Presence and Instant Messaging (CPIM)", RFC 3922, October 2004.

Authors' Addresses

Peter Saint-Andre
XMPP Standards Foundation
P.O. Box 1641
Denver, CO 80201
USA

Email: stpeter@jabber.org

Avshalom Houri
IBM
Building 18/D, Kiryat Weizmann Science Park
Rehovot 76123
Israel

Email: avshalom@il.ibm.com

Joe Hildebrand
Jabber, Inc.
1899 Wynkoop Street, Suite 600
Denver, CO 80202
USA

Email: jhildebrand@jabber.com

Acknowledgment

Funding for the RFC Editor function is provided by the IETF Administrative Support Activity (IASA).