July 23, 2001

Ms. Magalie Roman Salas
Secretary
Federal Communications Commission
445 12th Street, SW
Washington, DC 20554

Re: **Progress Report on Instant Messaging Interoperability**

Dear Ms. Salas:

Pursuant to the FCC's *Memorandum Opinion and Order* approving the merger between America Online, Inc. ("AOL") and Time Warner Inc. ("Time Warner"),[1] AOL Time Warner Inc. ("AOL Time Warner") hereby submits this progress report to update the Commission on AOL's ongoing efforts to develop a server-to-server IM interoperability solution that will allow a user of one of its IM services to exchange messages with users of unaffiliated IM services in a way that adequately protects IM network performance, privacy, and security.

---

[1] Memorandum Opinion and Order, *In the Matter of Applications for Consent to the Transfer of Control of Licenses and Section 214 Authorizations by Time Warner Inc. and America Online, Inc., Transferors, to AOL Time Warner Inc., Transferee*, CS Docket No. 00-30, FCC 01-12, ¶ 327 (rel. Jan. 22, 2001).

AOL publicly stated last July that it anticipated that it would require approximately one year to develop a server-to-server protocol, to be followed by a period of time to test and refine its interoperability solution. Consistent with this commitment, AOL has largely completed its development of the necessary technology, has recently begun internal testing of that technology, and remains on schedule to begin testing server-to-server interoperability with a leading technology company later this summer.

The Challenge Of IM Interoperability Is To Create A Safe And Secure Solution That Does Not Undermine The Essential Qualities That Have Made IM Popular

AOL attributes much of the success of its IM services to the qualities that distinguish these services from other forms of text-based communication: it is instant, it is reliable, and it is secure and private.[2]

- **Instant.** Messages and other communications are delivered quickly (*i.e.*, in near real-time), and users are notified immediately when their buddies sign on or off the service;[3]
- **Reliable.** IM systems perform at a high quality of service level and are designed to recognize and promptly address network failures (*e.g.*, PC

[2] Indeed, one of the biggest reasons for AOL's success in IM has been its vigilant approach, both in the design and day-to-day operations of its IM services, to protecting the user experience from disruptions, service outages, and/or security lapses that might jeopardize user confidence in its IM offerings.

[3] AOL's IM services today are specifically designed to ensure the prompt transmission of such data. For example, the AIM protocol is a binary protocol that provides more efficient data transmission than text-based protocols. In addition, AOL routes all server-to-server traffic within its IM networks on a private, high-speed LAN, thereby bypassing the threat to immediacy posed by data traffic congestion on the public Internet.

"crashes" and Internet traffic congestion);[4] and

- **Secure and Private**. IM services allow users to assume and control an identity (*i.e.*, a user name and password), and users are able to opt-out of messages they find intrusive.[5]

As explained below, interoperability, by definition, introduces a number of complicating issues that must be addressed in order to maintain these characteristics; otherwise, interoperability risks undermining the very reasons that IM has become as popular as it is today. As a result, it is not altogether surprising that to date others in the industry have yet to implement an interoperability solution, or that the IETF—while having made significant progress—has still not completed its work on server-to-server interoperability standards.

First, interoperability increases the potential for unacceptable delays in the transmission of messages and/or presence information, particularly across services.

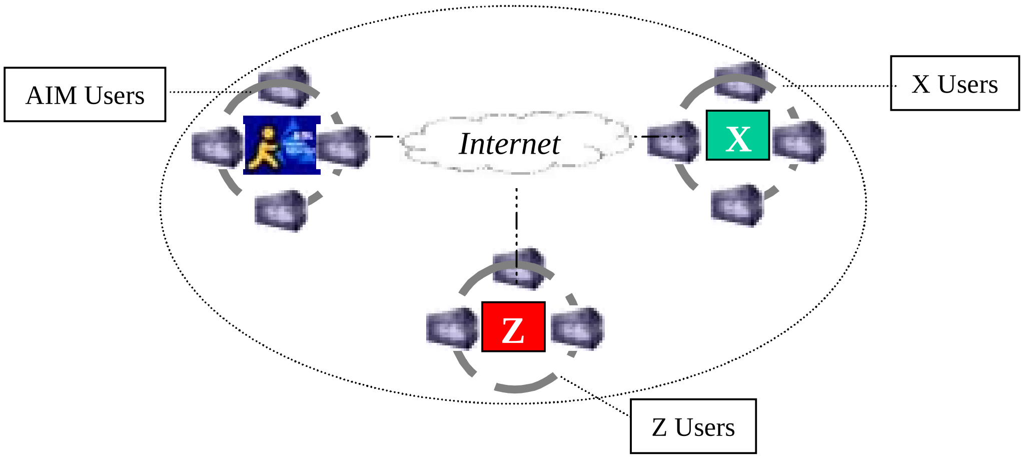
---

4 The AIM network incorporates a number of safeguards designed to minimize threats to its reliability. For example, the AIM network includes hundreds of servers, including back-up servers that are constantly in "alert mode." Moreover, all of AOL's clients and servers communicate frequently to make sure that the connections between them are being maintained, and when AOL's IM clients detect a connection failure, they immediately notify users that they are no longer online.

5 AOL's IM offerings have been specifically designed to provider users with a number of security and privacy features, including: (1) AIM's "knock-knock" feature, which, upon activation, requires user consent before displaying a message from a user not on their buddy list; (2) rate limits and user warnings, which impose limits on behavior within the AIM community; and (3) the IM feature of the AOL online service's "Notify AOL" function, which makes it possible to report offensive subscriber behavior directly to AOL.

By linking IM servers together, interoperability creates a single "virtual host" requiring continuous coordination and exchange of data between services:

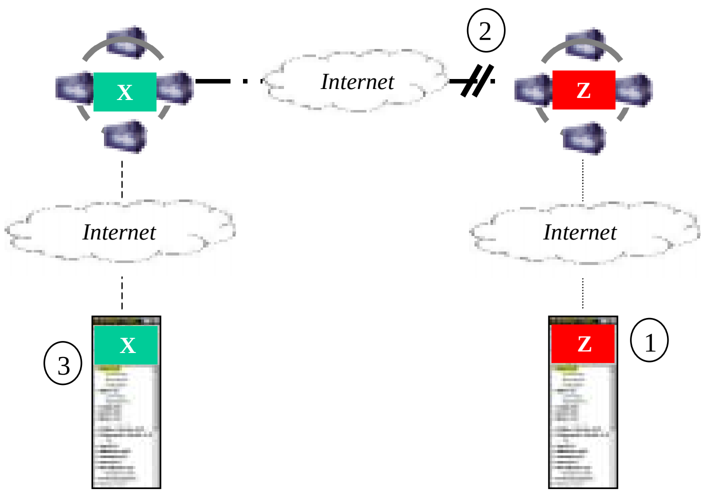
"Virtual Host" System



The problem with the "virtual host" approach, however, is that, to the extent that it relies upon the public Internet for the purpose of server-to-server communications, it potentially could lead to unacceptable delays in the transmission of message and presence data due to the data traffic congestion problems and bandwidth limitations that exist on the public Internet today.

Second, because interoperable IM providers will rely upon each other for accurate information, IM services will be affected by the service performance of all those systems with which they are interoperable—the reliable and unreliable alike. As is the case with email, IM systems participating in an interoperable network will operate to varying standards. Some potentially will suffer from poor performance and service outages. This is not a serious problem in email, because user expectations are more generous and the systems are designed to resend data whose receipt on the other end is not confirmed. In an interoperable IM network, however, failures will be difficult to identify and will cascade inaccurate information throughout the IM systems participating in that network. As a result, the best performing systems could appear to be malfunctioning, potentially as often as those that are actually causing the problems. To illustrate:

Schematic of Interoperable Systems



(1) User logs onto system Z, and X user sees him come online via presence notification.

(2) Z system's Internet connection fails (internal Z system messages still work).

(3) X user tries to IM Z user, but is unable to get message through (since Z system's connection has failed). Calls Z user who says he is still online. X user assumes X is at fault.

Thus, since IM services must rely upon each other for accurate presence and message information, outages will affect all systems—the reliable and unreliable alike.

Third, interoperability introduces potentially vulnerable points of access into IM providers' networks and forces IM providers to depend upon one another in their efforts to protect the privacy and security of their users. The points of vulnerability introduced by interoperability potentially enable bad actors, for example, to spam users with inappropriate images and/or text (*e.g.*, pornography), transmit viruses, impersonate IM users, or intercept messages. That is because interconnection points between two different networks, particularly if they are located on the public Internet, provide hackers with the opportunity to gain unauthorized access to those networks. In addition, interoperability also requires an IM provider to rely upon others to help enforce its policies regarding harassment and other inappropriate conduct.

A viable interoperability approach must adequately address these concerns if it is to enhance the user experience rather than undermine IM's basic appeal. Moreover, if all of these concerns are not fully addressed from day one, there is no way to resolve them at a later date: once a flawed protocol has been implemented, it is virtually impossible—witness email—to undo the damage.

In light of these technical challenges, it is not surprising that none of the efforts others have initiated to allow users of different IM services to exchange messages has been successful to date.[6] Indeed, the IETF, the leading Internet standards-setting body, established the Instant Messaging and Presence Protocol ("IMPP") working group for the purpose of developing a single server-to-server IM interoperability standard. Last summer, however, the IMPP working group abandoned that goal due to its inability to reach consensus support for any single, comprehensive protocol, and has instead limited its efforts to developing common messaging formats which other working groups, subsequently formed by the IETF, are implementing as they develop several different

---

[6] One of these initiatives was launched—during the height of the IM debate before the FCC last summer—by IMUnified. Originally, IMUnified announced that it would "provide a basic framework for detailing the mechanics of IM exchange among our members systems by the end of August [2000], with final implementation across member communities expected by the end of [2000]." *See* IMUnified FAQ <http://www.imunified.org/faq.html>. Both of those deadlines have since passed unmet.

server-to-server interoperability protocols.[7] At this point, there is no announced timetable indicating when the efforts to develop those protocols will be completed.

AOL Has Made Significant Progress Toward Developing A Server-To-Server Interoperability Solution, Has Recently Begun Internal Testing, And Is On Schedule To Conduct A System-To-System Trial With A Leading Technology Company

Last July, AOL publicly stated that it would require approximately twelve months from that date to develop a server-to-server IM interoperability protocol, plus an additional testing period to ensure that that protocol will not undermine AOL's continued ability to protect its IM users' experience from the types of risks described above. Consistent with this commitment (and despite the challenges described above), AOL has assembled the technology necessary to exchange messages and presence information between IM networks, has recently begun internal testing of that technology, and remains on schedule to begin testing server-to-server interoperability with a leading technology company by late Summer 2001.

On July 15, 2000, AOL submitted a white paper to the IETF outlining its proposed framework for server-to-server interoperability. Subsequently, additional working groups were formed within the IETF to implement a number of divergent approaches to server-to-server interoperability.[8] In addition, other server-to-server

---

7 The IMPP working group is working on the following Internet-drafts defining common messaging formats: "Common Presence and Instant Messaging: Message Format," <http://www.ietf.org/internet-drafts/draft-ietf-impp-cpim-msgfmt-03.txt>; and "Date and Time on the Internet: Time Stamps," <http://www.ietf.org/internet-drafts/draft-ietf-impp-datetime-04.txt>. Copies of these documents are attached.

8 As noted above, the IETF originally chartered a single working group, the IMPP

(Continued...)

interoperability efforts have been initiated in the IM marketplace, including open-source projects. AOL has evaluated each of these approaches with respect to its ability to satisfy AOL's requirements—in essence, whether it is capable of ensuring that IM's instant, reliable, and secure and private qualities survive the transition from an environment where a single provider controls the IM network from end to end to an environment in which IM providers depend upon the performance of all other providers' networks with which they are interoperable.

In the end, AOL determined that the optimal approach would be to develop a server-to-server interoperability framework using one of the standards being developed by the IETF. Of those, AOL selected the protocol being developed by the IETF's SIP for Instant Messaging and Presence Leverage ("SIMPLE") working group, which is working on an IM-specific implementation of the IETF's telephony-oriented Session Initiation Protocol ("SIP"). Among its considerations, AOL found that SIMPLE (and/or the SIP protocol from which it is derived) is already supported by a number of hardware and software companies and has a significant following among developers. The IETF Internet-draft describing in technical detail the SIMPLE messaging protocol, "SIP Extensions for Instant Messaging," is attached hereto and is also available at http://search.ietf.org/internet-drafts/draft-ietf-simple-im-00.txt; "SIP Extensions for

---

(...Continued)
working group, to develop a single IM server-to-server Internet standard. Because that working group was unable to achieve consensus support for any single protocol, three additional working groups—APEX, PRIM, and SIMPLE—were established to pursue divergent approaches to server-to-server interoperability. To date, none of these working groups has finished specifying its protocol.

Presence," the IETF Internet-draft describing in technical detail the SIMPLE presence protocol, is attached hereto and is also available at http://search.ietf.org/internet-drafts/draft-ietf-simple-presence-00.txt.

Because the SIMPLE working group has not finalized these protocols, however, AOL has had to resolve certain unsettled issues in the few functional areas where the working group has yet to make its final decisions. In particular, the comprehensive approach to interoperability AOL is working to complete will specify:

- That IM systems may establish dedicated, high-speed connections between their networks, thereby minimizing any bandwidth-related threats to the "instant" nature of IM;
- A quality of service level to which participating systems shall perform; and
- A standardized approach to privacy and security, including measures to protect users from spam and harassment.

Having thus assembled the components necessary to achieve basic interoperability—*i.e.*, the exchange of presence and message data—with other providers' IM systems, AOL is working to address additional implementation issues that must be resolved before it can introduce its interoperability solution into a real-world environment. At the same time, AOL is currently testing its basic interoperability components internally and is preparing to begin testing its comprehensive interoperability solution with an external partner.

To this end, AOL had to first develop an interoperable version of each component of the AIM service. This involved:

- creating a new version of the AIM client software;

- incorporating the ability to accept and process presence and message information from non-AOL systems into the AIM servers; and
- developing a gateway to translate the internal AIM protocol into the SIMPLE protocol in order to enable communication with other servers.

AOL completed this work in early July, and AOL has since been conducting internal trials intended to confirm its ability to pass presence and message information successfully between two model IM networks.

Once internal testing is completed, AOL intends to conduct a trial of its comprehensive interoperability solution, and is close to finalizing an agreement with a leading technology company that will allow the two companies to conduct a live server-to-server interoperability trial. In addition, AOL is working with this potential partner to draft a contractual agreement that addresses such concerns as performance requirements, cost sharing, and privacy and security policies. Upon successful completion of these tasks, AOL then plans to finalize its gateway, install updated code on its production servers, and begin developing a finished client that supports interoperability.

* * *

We appreciate this opportunity to have updated the Commission on AOL's progress on IM interoperability.

Respectfully submitted,

____

Steven N. Teplitz
Vice President, Communications Policy
And Regulatory Affairs
AOL Time Warner Inc.

cc: Chairman Michael K. Powell
Commissioner Gloria Tristani
Commissioner Kathleen Q. Abernathy
Commissioner Michael J. Copps
Commissioner Kevin J. Martin
W. Kenneth Ferree, Chief, Cable Services Bureau

Attachments:

"SIP Extensions for Instant Messaging"
"SIP Extensions for Presence"
"Common Presence and Instant Messaging: Message Format"
"Date and Time on the Internet: Time Stamps"

Internet Engineering Task Force J. Rosenberg
Internet-Draft D. Willis
Expires: October 11, 2001 R. Sparks
B. Campbell
dynamicsoft
H. Schulzrinne
J. Lennox
Columbia University
C. Huitema
B. Aboba
D. Gurle
Microsoft Corporation
D. Oran
Cisco Systems
April 12, 2001

SIP Extensions for Instant Messaging
draft-ietf-simple-im-00

Status of this Memo

Copyright Notice

Abstract

This document defines a SIP extension (a single new method) that

supports Instant Messaging (IM).

Table of Contents

1. Introduction

This document defines an extension to SIP (RFC2543 [2]) to support Instant Messaging.

Instant messaging is defined as the exchange of content between a set of participants in real time. Generally, the content is short textual messages, although that need not be the case. Generally, the messages that are exchanged are not stored, but this also need not be the case. IM differs from email in common usage in that instant messages are usually grouped together into brief live conversations, consisting of numerous small messages sent back and forth.

Instant messaging as a service has been in existence within intranets and IP networks for quite some time. Early implementations include zephyr [1], the unix talk application, and IRC. More recently, IM has been used as a service coupled with presence and buddy lists; that is, when a friend comes online, a user can be made aware of this and have the option of sending the friend an instant message. The protocols for accomplishing this are all proprietary, which has seriously hampered interoperability. Furthermore, most of these protocols tightly couple presence and IM, due to the way in which the service is offered.

Despite the popularity of presence coupled IM services, IM is a separate application from presence. There are many ways to use IM outside of presence (for example, as part of a voice communications session). Another example are interactive games (possibly established with SIP - SIP can establish any type of session, not just voice or video); IM is already a common component of multiplayer online games. Keeping it apart from presence means it can be used in such ways. Furthermore, keeping them separate allows separate providers for IM and for presence service. Of course, it can always be offered by the same provider, with both protocols implemented into a single client application.

Along a similar vein, the mechanisms needed in an IM protocol are very similar to those needed to establish an interactive session - rapid delivery of small content to a user at their current location, which may, in general, be dynamically changing as the user moves. The similarity of needed function implies that existing solutions for initiation of sessions (namely, the Session Initiation Protocol (SIP) [2]) is an ideal base on which to build an IM protocol.

2. Changes Introduced in draft-ietf-simple-im-00

The draft name changed to reflect its status as a SIMPLE working group item. This version introduces no other changes.

3. Changes Introduced in draft-rosenberg-impp-im-01

This submission serves to track transition of the work on a SIP implementation of IM to the newly formed SIMPLE working group. It endeavors to capture the progress made in IMPP since the original submission (in particular, including the im: URL and the message/cpim body) and detail a set of open issues for the SIMPLE working group to address.

To support those goals, a great deal of the background and motivation material in the original text has been shortened or removed.

4. Terminology

Most of the terminology used here is defined in RFC2778 [4]. However, we duplicate some of the terminology from SIP in order to clarify this document:

User Agent (UA): A UA is a piece of software which is capable of initiating requests, and of responding to requests.

User Agent Server (UAS): A UAS is the component of a UA which receives requests, and responds to them.

User Agent Client (UAC): A UAC is the component of a UA which sends requests, and receives responses.

Registrar: A registrar is a SIP server which can receive and process REGISTER requests. These requests are used to construct address bindings.

5. Overview of Operation

When one user wishes to send an instant message to another, the sender formulates and issues a SIP request using the new MESSAGE method defined by this document. The request URI of this request will normally be the im: URL of the party to whom the message is directed (see CPIM [15]), but can also be a normal SIP URL. The body of the request will contain the message to be delivered. This body can be of any MIME type, including "message/cpim" [16].

The request may traverse a set of SIP proxies using a variety of transport mechanism (UDP, TCP, even SCTP [5]) before reaching its destination. The destination for each hop is located using the address resolution rules detailed in the CPIM and SIP specifications (see Section 6 for more detail). During traversal, each proxy may rewrite the request URI based on available routing information.

Provisional and final responses to the request will be returned to the sender as with any other SIP request. Normally, a 200 OK response will be generated by the user agent of the request's final recipient. Note that this indicates that the user agent accepted the message, not that the user has seen it.

Groups of messages in a common thread may be associated by keeping them in the same session as identified by the combination of the To, From and Call-ID headers. Other potential means of grouping messages are discussed below.

It is possible that a proxy may fork a MESSAGE request based on its available routing mechanism. This draft proposes a mechanism that takes advangage of this, delivering messages in a session to multiple endpoints until one sends a message back. After that, all remaining messages in the session are delivered to the responding agent.

6. The MESSAGE request

This section defines the syntax and semantics of this extension.

6.1 Method Definition

This specification defines a new SIP method, MESSAGE. The BNF for this method is:

```
Message  =  "MESSAGE"
```

As with all other methods, the MESSAGE method name is case sensitive.

Tables 1 and 2 extend Tables 4 and 5 of SIP by adding an additional column, defining the headers that can be used in MESSAGE requests and responses.

	where	enc.	e-e	MESSAGE
Accept	R		e	o
Accept	415		e	o
Accept-Encoding	R		e	o
Accept-Encoding	415		e	o
Accept-Language	R		e	o
Accept-Language	415		e	o
Allow	200		e	o
Allow	405		e	m
Authorization	R		e	o
Authorization	r		e	o
Call-ID	gc	n	e	m
Contact	R		e	m
Contact	2xx		e	o
Contact	3xx		e	o
Contact	485		e	o
Content-Encoding	e		e	o
Content-Length	e		e	m

Content-Type	e		e	*
CSeq	gc	n	e	m
Date	g		e	o
Encryption	g	n	e	o
Expires	g		e	o
From	gc	n	e	m
Hide	R	n	h	o
Max-Forwards	R	n	e	o
Organization	g	c	h	o

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

	where	enc.	e-e	MESSAGE
Priority	R	c	e	o
Proxy-Authenticate	407	n	h	o
Proxy-Authorization	R	n	h	o
Proxy-Require	R	n	h	o
Record-Route	R		h	o
Record-Route	2xx,401,484		h	o
Require	R		e	o
Retry-After	R	c	e	-
Retry-After	404,413,480,486	c	e	o
	500,503	c	e	o
	600,603	c	e	o
Response-Key	R	c	e	o
Route	R		h	o
Server	r	c	e	o
Subject	R	c	e	o
Timestamp	g		e	o
To	gc(1)	n	e	m
Unsupported	420		e	o
User-Agent	g	c	e	o
Via	gc(2)	n	e	m
Warning	r		e	o

WWW-Authenticate	R	c	e	o
WWW-Authenticate	401	c	e	o

(1): copied with possible addition of tag
(2): UAS removes first Via header field

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

A MESSAGE request MAY (Open Issue Section 9.1) contain a body, using the standard MIME headers to identify the content.

Unless stated otherwise in this document, the protocol for emitting and responding to a MESSAGE request is identical to that for a BYE request as defined in [2]. The behavior of SIP entities not implementing the MESSAGE (or any other unknown) method is explicitly defined in [2].

6.2 UAC processing of initial MESSAGE request

A MESSAGE request MUST contain a To, From, Call-ID, CSeq, Via, Content-Length, and Contact header, formatted as specified in [2].

All UAs MUST be prepared to send and receive MESSAGE requests with a body of type text/plain. All UAs wishing to provide the end to end security mechanisms defined in CPIM MUST be prepared to send and receive MESSAGE requests with a body type of message/cpim. All UAs

implementing MESSAGE SHOULD provide the end to end security mechanisms defined in CPIM (Open Issue Section 9.2).

MESSAGE requests MAY contain an Accept header listing the allowable MIME types which may be sent in the response, or in subsequent requests in the reverse direction. The absence of the Accept header implies that the only allowed MIME type is text/plain. This simplifies operation in small devices, such as wireless appliances, which will generally only have support for text, but still allows any other MIME type to be used if both sides support it. (Open Issue Section 9.3)

A UAC MAY send a MESSAGE request within an existing SIP call, established with an INVITE. In this case, the MESSAGE request is processed identically to the INFO method [9]. The only difference is that a MESSAGE request is assumed to be for the purpose of instant messaging as part of the call, whereas INFO is less specific.

A UAC MAY associate sequential MESSAGEs in a common thread by constructing them with common To, From, and Call-ID headers and increasing CSeq values. (Open Issue Section 9.4)

6.3 Finding the next hop

The mechanism used to determine the next hop destination for a SIP MESSAGE request is detailed in [15] and [2]. Briefly, for the URL im:user@host,

1. The UA makes a DNS SRV [12] query for _im._sip.host. If any RRs are returned, they determine the next hop. Otherwise:
2. The UA makes a DNS SRV query for _sip.host. If any RRs are returned, they determine the next hop. Otherwise:
3. The UA makes a DNS A query for host. If any records are returned, they determine the address of the next hop. The desination port is determined from the input URL (if the input was an im: URL, the request is sent to the default SIP port of 5060).

For sip: URLs, the UA starts at step 2.

6.4 Proxy processing of MESSAGE requests

Proxies route requests with method MESSAGE the same as they would any other SIP request (proxy routing in SIP does not depend on the method). Note that the MESSAGE request MAY fork; this allows for delivery of the message to several possible terminals where the user might be.

If a MESSAGE request hits a proxy that uses registrations to route requests, but no registration exists for the target user in the request-URI, the request is rejected with a 404 (Not Found).

Proxies MAY have access rules which prohibit the transmission of instant messages based on certain criteria. Typically, this criteria will be based on the identity of the sender of the instant messages. Establishment of this criteria in the proxy is outside the scope of this extension. We anticipate that such access controls will often be controlled through web pages accessible by users, mitigating the need for standardization of a protocol for defining access rules.

6.5 UAS processing of MESSAGE requests

As specified in RFC 2543, if a UAS receives a request with a body of type it does not understand, it MUST respond with a 415 (Unsupported Media Type) containing an Accept header listing those types which are acceptable. (This brings up Open Issue Section 9.3 again)

Servers MAY reject requests (using a 413 response code) that are too long, where too long is a matter of local configuration. All servers MUST accept requests which are up to 1184 bytes in length.

> 1184 = minimum IPv6 guaranteed length (1280 bytes) minus UDP (8 bytes) minus IPSEC (48 bytes) minus layer one encapsulation (40 bytes).

A UAS receiving a MESSAGE request SHOULD respond with a final response immediately. A 200 OK is sent if the request is acceptable. Note, however, that the UAS is not obliged to display the message to the user either before or after responding with a 200 OK. A 200 class response to a MESSAGE request MAY contain a body, but this will often not be the case, since these responses are generated automatically. (Open Issue Section 9.5)

Like any other SIP request, an IM MAY be redirected, or otherwise responded to with any SIP response code. Note that a 200 OK response to a MESSAGE request does not mean the user has read the message.

A UAS which is, in fact, a message relay, storing the message and forwarding it later on, or forwarding it into a non-SIP domain, SHOULD return a 202 (Accepted) response indicating that the message was accepted, but end to end delivery has not been guaranteed.

6.6 UAS processing of initial MESSAGE response

A 200 OK response to an initial IM may contain Record-Route headers. If present, these MUST be used to construct a Route header for use in subsequent requests for the same call-leg (defined as the combination of remote address, local address, and Call-ID), using the process described in Section 6.29 of SIP [2] as if the request were INVITE. Note that per Section 6.8 the 200 OK response may not contain a Contact header.

A 400 or 500 class response indicates that the message was not delivered successfully. A 600 response means it was delivered successfully, but refused.

6.7 Subsequent MESSAGE requests

Any subsequent MESSAGEs in a session (see Section 9.4 follow the path established by the Route headers computed by the UA. The CSeq header MUST be larger than a CSeq header used in a previous request for the same call leg. Is is strongly RECOMMENDED that the CSeq number be computed as described in Section 6.17 of SIP, using a clock. This allows for the CSeq to increment without requiring the UA to store the previous CSeq values.

6.8 Supporting multiple message destinations

A UAS MAY include a Contact in a 200 class response. Including a Contact header enables end to end messaging, which is good for efficiency. However, it rules out the possibility of effectively supporting more than one terminal which can handle IM simultaneously.

> This odd but seemingly innocuous requirement enables a very important feature. If a user is connected at several hosts, an initial IM will fork, and arrive at each. Each UAS responds with a 200 OK immediately, one of which is arbitrarily forwarded upstream towards the UAC. If another IM is sent for the same call-leg, we still wish for this IM to fork, since we still don't know where the user is currently residing. This information is known when the user sends an IM in the reverse direction. This IM will contain a Contact, and when it arrives at the originator of the initial MESSAGE, will update the Route so that now IMs are delivered only to that one host where the user is residing.

A UAS constructs a set of Route headers from the Record-Route and

Contact headers in the MESSAGE request, as per the procedure defined in [10].

MESSAGE requests for an established IM session MUST contain a Tag in the From field. Responses to an IM SHOULD contain a tag in the To field. This represents a slightly different operation than for INVITE. When a user sends an INVITE, they will receive a 200 OK with a tag. Requests in the reverse direction then contain that tag, and that tag only, in the From field. Here, the response to IM will contain a tag in the To field, and a MESSAGE will contain a tag in the From field. However, the UA may receive MESSAGE requests with tags in the From field that do not match the tag in the 200 OK received to the initial IM. This is because only a single 200 OK is returned to a MESSAGE request, as opposed to multiple 200 OK for

INVITE. Thus, the UA MUST be prepared to receive MESSAGEs with many different tags, each from a different PUA.

A UAS MUST be prepared to update the Route is has stored for an IM session with a Contact received in a request, if that Contact is different from one previously received, or if there was no Contact previously.

6.9 Caller Preferences

User agents SHOULD add the "methods" tag defined in the caller preference specification [8] to Contact headers with SIP URLs placed in REGISTER requests, indicating support for the MESSAGE method. Other elements of caller preferences MAY be supported. For example:

```
REGISTER sip:dynamicsoft.com SIP/2.0
Via: SIP/2.0/UDP mypc.dynamicsoft.com
To: sip:jdrosen@dynamicsoft.com
From: sip:jdrosen@dynamicsoft.com
Call-ID: asidhasd@1.2.3.4
CSeq: 39 REGISTER
Contact: sip:jdrosen@im-pc.dynamicsoft.com;methods="MESSAGE"
Content-Length: 0
```

Registrar/proxies which wish to offer IM service SHOULD implement the proxy processing defined in the caller preferences specification [8].

6.10 Security

End-to-end security concerns for instant messaging were a primary driving force behind the creation of message/cpim [16]. Applications needing end-to-end security should study that work carefully.

SIP provides numerous security mechanisms which can be utilized in addition to those made available through the use of message/cpim.

6.10.1 Privacy

In order to enhance privacy of instant messages, it is RECOMMENDED that between network servers (proxies to proxies, proxies to redirect servers), transport mode ESP [6] or TLS is used to encrypt all traffic. Coupled with persistent connections, this makes it impossible for eavesdroppers on non-UA connections to determine when a particular user has even sent an IM, let alone what the content is. Of course, the content of unencrypted IMs are exposed to proxies.

Between a UAC and its local proxy, TLS [11] is RECOMMENDED. Similarly, TLS SHOULD be used between a proxy and the UAS receiving the IM. The proxy can determine whether TLS is supported by the receiving client based on the transport parameter in the Contact header of its registration. If that registration contains the token "tls" as transport, it implies that the UAS supports TLS. (Open issue Section 9.7)

Furthermore, we allow for the Contact header in the MESSAGE request to contain TLS as a transport. The Contact header is used to route subsequent messages between a pair of entities. It defines the address and transport used to communicate with the user agent for subsequent requests in the reverse direction. If no proxies insert Record-Route headers, the recipient of the original IM, when it wishes to send an IM back, will use the Contact header, and establish a direct TLS connection for the remainder of the IM communications. If a proxy does Record-Route, the situation is different. When the recipient of the original IM (call this participant B) sends an IM back to the originator of the original IM (call this participant A), this will be sent to the proxy closest to B which inserted Record- Route. This proxy, in turn, sends the request to the proxy before it which Record-Routed. The first proxy after A which inserted Record- Route will then use TLS to contact A. Since we suspect that most proxies will not insert Record-Route into instant messages, efficient, secure, direct IM will occur frequently.

If encrypted message/cpim bodies are not available, sensitive data may be protected from being observed by intermediate proxies by using SIP encryption for the transmission of MESSAGE requests. SIP supports PGP based encryption, which does not require the establishment of a session key for encryption of messages within a session (basically, a new session key is established for each message as part of the PGP encryption).

6.10.2 Message Integrity and Authenticity

In addition to the integrity and authenticity protections offered through message/cpim, SIP provides PGP based authentication and message integrity checks (both challenge-response and normal signatures), as well as http basic and digest authentication.

6.10.3 Outbound authentication

When local proxies are used for transmission of outbound messages, proxy authentication is RECOMMENDED. This is useful to verify the identity of the originator, and prevent spoofing and spamming at the originating network.

6.10.4 Replay Prevention

To prevent the replay of old SIP requests, all signed MESSAGE requests and responses SHOULD contain a Date header covered by the message signature. Any message with a date older than several minutes in the past, or which is more than several minutes in the future, SHOULD be answered with a 400 (Incorrect Date or Time) message, unless such messages arrive repeatedly from the same source, in which case they MAY be discarded without sending a response. Obviously, this replay attack prevention mechanism does not work for devices without clocks.

Furthermore, all signed SIP MESSAGE requests MUST contain a Call-ID and CSeq header covered by the message signature. A user agent MAY store a list of Call-ID values, and for each, the higest CSeq seen within that Call-ID. Any message that arrives for a Call-ID that exists, whose CSeq is lower than the highest seen so far, is discarded.

Finally, challenge-response authentication MAY be used to prevent replay protection.

7. Congestion Control

(Open Issue Section 9.8) Discussion needs to take place to populate this section.

8. Example Messages

An example message flow is shown in Figure 1. The message flow shows an initial IM sent from User 1 to User 2, both users in the same domain, "domain", through a single proxy. A second IM, sent in response, flows directly from User 2 to User 1.

```
|  F1 MESSAGE           |                         |
|---------------------> |  F2 MESSAGE             |
|                       | ----------------------->|
|                       |                         |
|                       |  F3 200 OK              |
|                       | <-----------------------|
|  F4 200 OK            |                         |
|<--------------------- |                         |
|                       |                         |
|                       |                         |
|                       |                         |
```

```
                 |                      |   F5 MESSAGE             |
                 | <--------------------|------------------------- |
```



```
                 |                      |                          |
                 |      F6 200 OK       |                          |
                 | ---------------------|-----------------------> |
                 |                      |                          |
                 |                      |                          |
                 |                      |                          |
                 |                      |                          |
                 |                      |                          |
                 |                      |                          |
                 |                      |                          |
                 |                      |                          |
                 |                      |                          |
                 |                      |                          |
                 |                      |                          |
                 |                      |                          |
                 |                      |                          |
                 |                      |                          |


               User 1                 Proxy                    User 2
```

Figure 1: Example Message Flow

Message F1 looks like:

```
MESSAGE im:user2@domain.com SIP/2.0
Via: SIP/2.0/UDP user1pc.domain.com
From: im:user1@domain.com
To: im:user2@domain.com
Contact: sip:user1@user1pc.domain.com
Call-ID: asd88asd77a@1.2.3.4
CSeq: 1 MESSAGE
Content-Type: text/plain
Content-Length: 18

Watson, come here.
```

User1 forwards this message to the server for domain.com (discovered through the combination of SRV and A record processing described in Section 6.3 , using UDP. The proxy receives this request, and recognizes that it is the server for domain.com. It looks up user2 in its database (built up through registrations), and finds a binding from im:user2@domain.com to sip:user2@user2pc.domain.com. It forwards the request to user2, and does not insert a Record-Route header. The resulting message, F2, looks like:

```
MESSAGE sip:user2@domain.com SIP/2.0
Via: SIP/2.0/UDP proxy.domain.com
Via: SIP/2.0/UDP user1pc.domain.com
From: im:user1@domain.com
To: im:user2@domain.com
Contact: sip:user1@user1pc.domain.com
Call-ID: asd88asd77a@1.2.3.4
CSeq: 1 MESSAGE
Content-Type: text/plain
Content-Length: 18

Watson, come here.
```

The message is received by user2, displayed, and a response is generated, message F3, and sent to the proxy:

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP proxy.domain.com
Via: SIP/2.0/UDP user1pc.domain.com
From: im:user1@domain.com
To: im:user2@domain.com;tag=ab8asdasd9
Contact: sip:user2@user1pc.domain.com
Call-ID: asd88asd77a@1.2.3.4
CSeq: 1 MESSAGE
Content-Length: 0
```

Note that most of the header fields are simply reflected in the response. The proxy receives this response, strips off the top Via, and forwards to the address in the next Via, user1pc.domain.com, the result being message F4:

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP user1pc.domain.com
From: im:user1@domain.com
To: im:user2@domain.com;tag=ab8asdasd9
Call-ID: asd88asd77a@1.2.3.4
CSeq: 1 MESSAGE
Content-Length: 0
```

Now, user2 wishes to send an IM to user1, message F5. As there are no Record-Routes in the original IM, it can simply send the IM directly to the address in the Contact header. Note how the To and From fields are now reversed from the response it sent in message F4:

```
MESSAGE sip:user1@user1pc.domain.com SIP/2.0
Via: SIP/2.0/UDP user2pc.domain.com
To: im:user1@domain.com
From: im:user2@domain.com;tag=ab8asdasd9
Contact: sip:user2@user2pc.domain.com
Call-ID: asd88asd77a@1.2.3.4
CSeq: 1 MESSAGE
Content-Type: multipart/signed; boundary=next;
              MDALG=SHA-1; type=application/pkcs7
Content-Length: <however many bytes that is below>

--next
Content-Type: message/cpim

From: <im:user2@domain.com>
To: <im:user1@domain.com>
Date: 2001-02-28T01:20:00-06:00

Content-Type: text/plain

My name is User2, not Watson.

--next
Content-Type: application/pkcs7

(signature stuff)
 :
--next--
```

This is sent directly to user1, who responds with a 200 OK in message F6:

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP user2pc.domain.com
To: im:user1@domain.com;tag=2c09sj3sd9
From: im:user2@domain.com;tag=ab8asdasd9
Call-ID: asd88asd77a@1.2.3.4
CSeq: 1 MESSAGE
Content-Length: 0
```

9. Open Issues

9.1 Must a MESSAGE actually include a message?

Section 6 specifies that a MESSAGE MAY contain a MIME body. Should this be MUST? Does it make sense to have a MESSAGE with no body?

9.2 Should support for message/cpim be mandatory in all UAs?

Section 6 requires that UAs implementing MESSAGE support text/plain bodies as the lowest common denominator. Should this be message/cpim instead? Any UA wishing to support end-end signing or encryption of messages passing across simple/apex/prim boundaries MUST support message/cpim. If, however, end-end security is not desired, clients and messaging can be made a little lighter by not including the message/cpim wrapper. An unsigned message/cpim body can be created from messages from those clients when crossing a boundary that requires one.

9.3 message/cpim and the Accept header

Do we need text to make it clear that a UA should indicate the mime types it supports _inside_ a message/cpim body as well as supporting message/cpim?

9.4 Message Sessions

Several implementations of the -00 version of this draft grouped messages in a common thread by placing them in a "call-leg" (common To, From, and Call-ID). The first message sent or received in a thread established the leg. This has provided enough information to allow user interfaces to present separate threads in separate dialogs. There is some concern that there is no way to formally terminate this "call-leg".

The -00 version noded that there is state at the UA associated with this notion of session, encapsulated in the Call-ID, Route headers, and CSeq numbers. A UA MAY terminate this session at any time, including after each MESSAGE. No messaging is required to terminate it. Any associated state with the session is simply discarded. The idempotency of SIP requests will ensure that if one side (side A) discards session state, and the other (side B) does not, a message from side B will appear as a new IM, and standard processing will reconstitute the session on side A.

- Should we define a way to use INVITE/BYE to surround a group of MESSAGE requests that are part of a logical session?

9.5 What would a body in a 200 OK to a MESSAGE mean?

Section 6.5 states "A 200 class response to a MESSAGE request MAY contain a body, but this will often not be the case, since these responses are generated automatically." If one were to appear, what would it mean?

9.6 The im: URL and RFC2543 proxies and registrars

What are the implications of an im: URL showing up in the request URI in a MESSAGE request received by an RFC2543 proxy, or the To: header of a REGISTER request received by an RFC2543 registrar?

9.7 Providing im: URL in Contact headers

What are the ramifications of a UA providing an im: URL in a Contact: header for a REGISTER method, or a MESSAGE method? For the forseeable future, most SIP endpoints aren't going to have SRV records of the form _im._sip.host or even _sip.host pointing to them. Falling back to A records in that case seems to preclude the use of non-UDP transports.

9.8 Congestion control

Per the amendments made to the SIMPLE charter by the IESG prior to approval, congestion control needs attention. In particular the requirements of BCP 41 must be met by this extension. Specifying the use of transport protocols with congestion control built in, particularly with the recommendation of reuse of connections, is an option. The question is when can we use those that don't (UDP) and what needs to be done in addition to what SIP already does in that case. Among other things, this interacts with Section 9.7

9.9 Mapping to CPIM

This document needs to detail the mapping of this extension onto CPIM.

10. Acknowledgements

The authors would like to thank the following people for their support of the concept of SIP for IM, support for this work, and for their useful comments and insights:

Jon Peterson	Level(3) Communications
Sean Olson	Ericsson
Adam Roach	Ericsson
Billy Biggs	University of Waterloo
Stuart Barkley	UUNet
Mauricio Arango	SUN
Richard Shockey	Shockey Consulting LLC
Jorgen Bjorker	Hotsip
Henry Sinnreich	MCI Worldcom
Ronald Akers	Motorola

References

[1]   DellaFera, C. A., Eichin, M. W., French, R. S., Jedlinski, D. C., Kohl, J. T. and W. E. Sommerfeld, "The Zephyr notification service", in USENIX Winter Conference (Dallas, Texas), Feb. 1988.

[2]   Handley, M., Schulzrinne, H., Schooler, E. and J. Rosenberg, "SIP: Session Initiation Protocol", RFC 2543, March 1999.

[3]   Day, M., Aggarwal, S. and J. Vincent, "Instant Messaging /

Presence Protocol Requirements", RFC 2779, February 2000.

[4] Day, M., Rosenberg, J. and H. Sugano, "A Model for Presence and Instant Messaging", RFC 2778, February 2000.

[5] Rosenberg, J. and H. Schulzrinne, "SCTP as a transport for SIP", draft-rosenberg-sip-sctp-00 (work in progress), June 2000.

[6] Kent, S. and R. Atkinson, "IP Encapsulating Security Payload (ESP)", RFC 2406, November 1998.

[7] Harkins, D. and D. Carrel, "The Internet Key Exchange (IKE)", RFC 2409, November 1998.

[8] Rosenberg, J. and H. Schulzrinne, "SIP caller preferences and callee capabilities", draft-ietf-sip-callerprefs-03 (work in progress), November 2000.

[9] Donovan, S., "The SIP INFO Method", RFC 2976, October 2000.

[10] Handley, M., Schulzrinne, H., Schooler, E. and J. Rosenberg, "SIP: Session Initiation Protocol", RFC 2543, March 1999.

[11] Dierks, T., Allen, C., Treese, W., Karlton, P. L., Freier, A. O. and P. C. Kocher, "The TLS Protocol Version 1.0", RFC 2246, January 1999.

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

[13] Handley, M. and V. Jacobson, "SDP: Session Description Protocol", RFC 2327, April 1998.

[14] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, November 1996.

[15] Crocker, D., Diacakis, A., Mazzoldi, F., Huitema, C., Klyne, G., Rose, M., Rosenberg, J., Sparks, R. and H. Sugano, "A Common Profile for Instant Messaging (CPIM)", draft-ietf-impp-cpim-01 (work in progress), February 2001.

[16] Atkins, D. and G. Klyne, "Common Presence and Instant Messaging Message Format", draft-ietf-impp-cpim-msgfmt-00 (work in progress), February 2001.

Authors' Addresses

Jonathan Rosenberg
dynamicsoft
200 Executive Drive

Suite 120
West Orange, NJ 07052

email: jdrosen@dynamicsoft.com

Dean Willis
dynamicsoft
5100 Tennyson Parkway
Suite 1200
Plano, TX 75024

email: dwillis@dynamicsoft.com

Robert J. Sparks
dynamicsoft
5100 Tennyson Parkway
Suite 1200
Plano, TX 75024

email: rsparks@dynamicsoft.com

Ben Cambpell
dynamicsoft
5100 Tennyson Parkway
Suite 1200
Plano, TX 75024

email: bcampbell@dynamicsoft.com

Henning Schulzrinne
Columbia University
M/S 0401
1214 Amsterdam Ave.
New York, NY 10027-7003

email: schulzrinne@cs.columbia.edu

Jonathan Lennox
Columbia University
M/S 0401
1214 Amsterdam Ave.
New York, NY 10027-7003

email: lennox@cs.columbia.edu

Christian Huitema

Microsoft Corporation
One Microsoft Way
Redmond, WA 98052-6399

email: huitema@microsoft.com

Bernard Aboba
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052-6399

email: bernarda@microsoft.com

David Gurle
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052-6399

email: dgurle@microsoft.com

David Oran
Cisco Systems
170 West Tasman Dr.
San Jose, CA 95134

email: oran@cisco.com

Appendix A. Requirements Evaluation

This section was moved forward verbatim from -00.

RFC 2779 [3] outlines requirements for IM and presence protocols. The document describes both shared requirements and IM and presence specific requirements. Examining each of the IM requirements in turn, we also observe that they are met by this proposal:

"Requirement 2.1.1: The protocols MUST allow a PRESENCE SERVICE to be available independent of whether an INSTANT MESSAGE SERVICE is available, and vice-versa." This requirement is met by the separation of presence and IM which we propose here.

"Requirement 2.1.2. The protocols must not assume that an INSTANT INBOX is necessarily reached by the same IDENTIFIER as that of a PRESENTITY. Specifically, the protocols must assume that some INSTANT INBOXes may have no associated PRESENTITIES, and vice versa." This requirement is also easily met by any architecture which completely separates IM and presence as we propose.

"Requirement 2.1.3. The protocols MUST also allow an INSTANT INBOX to be reached via the same IDENTIFIER as the IDENTIFIER of some PRESENTITY." Same as above.

"Requirement 2.1.4. The administration and naming of ENTITIES within a given DOMAIN MUST be able to operate independently of actions in any other DOMAIN." This requirement is met by SIP. SIP uses email-like identifiers which consist of a user name at a domain. Administration of user names is done completely within the domain, and these user names have no defined rules or organization that needs to be known outside of the domain in order for SIP to operate.

"Requirement 2.1.5. The protocol MUST allow for an arbitrary number of DOMAINS within the NAMESPACE." This requirement is met by SIP. SIP uses standard DNS domains, which are not restricted in number.

"Requirement 2.2.1. It MUST be possible for ENTITIES in one DOMAIN to interoperate with ENTITIES in another DOMAIN, without the DOMAINS having previously been aware of each other." This requirement is met by SIP, as it is essential for establishing sessions as well. DNS SRV records are used to discover servers for a particular service within a domain. They are a generalization of MX records, used for email routing. SIP defines procedures for usage of DNS records to find servers in another domains, which include SRV lookups. This allows domains to communicate without prior setup.

"Requirement 2.2.2: The protocol MUST be capable of meeting its other functional and performance requirements even when there are millions of ENTITIES within a single DOMAIN." Whilst it is hard to judge whether this can be met by examining the architecture of a protocol, SIP has numerous mechanisms for achieving large scales of users within a domain. It allows hierarchies of servers, whereby the namespace can be partitioned among servers. Servers near the top of the hierarchy, used solely for routing, can be stateless, providing excellent scale.

"Requirement 2.2.3: The protocol MUST be capable of meeting its other functional and performance requirements when there are millions of DOMAINS within the single NAMESPACE." The usage of DNS for dividing the namespace into domains provides the same scale as todays email systems, which support millions of DOMAINS.

"Requirement 2.3.5: The PRINCIPAL controlling an INSTANT INBOX MUST be able to control which other PRINCIPALS, if any, can send INSTANT MESSAGES to that INSTANT INBOX." This is provided by access control mechanisms, outside the scope of this extension.

"Requirement 2.3.6: The PRINCIPAL controlling an INSTANT INBOX MUST be able to control which other PRINCIPALS, if any, can read INSTANT MESSAGES from that INSTANT INBOX." This is accomplished through authenticated registration requests. Registrations are used to determine which user gets delivered an instant message. Policy in proxies can allow only certain users to register contact address for a particular inbox (an inbox is defined by the address-of- record in the To field in the registration).

"Requirement 2.4.3: The protocol MUST allow the sending of an INSTANT MESSAGE both directly and via intermediaries, such as PROXIES." This is fundamental to the operation of SIP.

"Requirement 2.4.4: The protocol proxying facilities and transport practices MUST allow ADMINISTRATORS ways to enable and disable protocol activity through existing and commonly-deployed FIREWALLS. The protocol MUST specify how it can be effectively filtered by such FIREWALLS." Although SIP itself runs on port 5060 by default, any other port can be used. It is simple to specify that IM should run on a different port, if so desired.

"Requirement 2.5.1. The protocol MUST provide means to ensure confidence that a received message (NOTIFICATION or INSTANT MESSAGE) has not been corrupted or tampered with." This is supported by SIPs PGP and S/MIME authentication mechanism.

"Requirement 2.5.2. The protocol MUST provide means to ensure confidence that a received message (NOTIFICATION or INSTANT

MESSAGE) has not been recorded and played back by an adversary." This is provided by SIP's challenge response authentication mechanisms, through timestamp-based replay prevention, or through stateful storage of previous transaction identifiers (the combination of To, From, Call-ID, CSeq).

"Requirement 2.5.3. The protocol MUST provide means to ensure that a sent message (NOTIFICATION or INSTANT MESSAGE) is only readable by ENTITIES that the sender allows." This is supported through SIPs end to end and hop by hop encryption mechanisms.

"Requirement 2.5.4. The protocol MUST allow any client to use the means to ensure non-corruption, non-playback, and privacy, but the protocol MUST NOT require that all clients use these means at all times." All algorithms for security in SIP are optional.

"Requirement 4.1.1. All ENTITIES sending and receiving INSTANT MESSAGES MUST implement at least a common base format for INSTANT MESSAGES." We specify text/plain here.

"Requirement 4.1.2. The common base format for an INSTANT MESSAGE MUST identify the sender and intended recipient." This is accomplished with the To and From fields in SIP.

"Requirement 4.1.3. The common message format MUST include a return address for the receiver to reply to the sender with another INSTANT MESSAGE." This is done through the Contact headers defined in SIP.

"Requirement 4.1.4. The common message format SHOULD include standard forms of addresses or contact means for media other than

INSTANT MESSAGES, such as telephone numbers or email addresses." SIP supports any URL format in the Contact headers. Furthermore, the body of a MESSAGE request can be multipart, and contain things like vCards.

"Requirement 4.1.5. The common message format MUST permit the encoding and identification of the message payload to allow for non-ASCII or encrypted content." MIME content labeling is used in SIP.

"Requirement 4.1.6. The protocol must reflect best current practices related to internationalization." SIP uses UTF-8 and is completely internationalized.

"Requirement 4.1.7. The protocol must reflect best current practices related to accessibility." Additional requirements are needed on what is required for accessibility.

"Requirement 4.1.9. The working group MUST determine whether the common message format includes fields for numbering or identifying messages. If there are such fields, the working group MUST define the scope within which such identifiers are unique and the acceptable means of generating such identifiers." This is done with the combination of Call-ID and CSeq. The mechanisms for guaranteeing uniqueness are specified in SIP.

"Requirement 4.1.10. The common message format SHOULD be based on IETF-standard MIME (RFC 2045)[14]." SIP uses MIME.

"Requirement 4.2.1. The protocol MUST include mechanisms so that a sender can be informed of the SUCCESSFUL DELIVERY of an INSTANT MESSAGE or reasons for failure. The working group must determine what mechanisms apply when final delivery status is unknown, such as when a message is relayed to non-IMPP systems." SIP specifies notification of successful delivery through 200 OK. When delivery of requests through gateways, success can be indicated only through the SIP component (if the gateway acts as a UAS/UAC) or through the entire system (if it acts like a proxy).

"Requirement 4.3.1. The transport of INSTANT MESSAGES MUST be sufficiently rapid to allow for comfortable conversational exchanges of short messages." The support for end to end messaging (i.e., without intervening proxies) allows IMs to be delivered as rapidly as possible. The UDP reliability mechanisms also support fast recovery from loss.

Full Copyright Statement

Acknowledgement

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

Internet Engineering Task Force SIMPLE WG
Internet Draft Rosenberg et al.
draft-ietf-simple-presence-00.txt Various Places
March 30, 2001
Expires: September 2001

SIP Extensions for Presence

STATUS OF THIS MEMO

This document is an Internet-Draft and is in full conformance with all provisions of Section 10 of RFC2026.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet- Drafts as reference material or to cite them other than as work in progress.

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt

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

Abstract

This document proposes an extension to SIP for subscriptions and notifications of user presence. User presence is defined as the willingness and ability of a user to communicate with other users on the network. Historically, presence has been limited to "on-line" and "off-line" indicators; the notion of presence here is broader. Subscriptions and notifications of user presence are supported by defining an event package within the general SIP event notification framework. This protocol is also compliant with the Common Presence and Instant Messaging (CPIM) framework.

1 Introduction

Presence is (indirectly) defined in RFC2778 [1] as subscription to and notification of changes in the communications state of a user.

This communications state consists of the set of communications means, communications address, and status of that user. A presence protocol is a protocol for providing such a service over the Internet or any IP network.

This document proposes an extension to the Session Initiation Protocol (SIP) [2] for presence. This extension is a concrete instantiation of the general event notification framework defined for SIP [3], and as such, makes use of the SUBSCRIBE and NOTIFY methods defined there. User presence is particularly well suited for SIP. SIP registrars and location services already hold user presence information; it is uploaded to these devices through REGISTER messages, and used to route calls to those users. Furthermore, SIP networks already route INVITE messages from any user on the network to the proxy that holds the registration state for a user. As this state is user presence, those SIP networks can also allow SUBSCRIBE requests to be routed to the same proxy. This means that SIP networks can be reused to establish global connectivity for presence subscriptions and notifications.

This extension is based on the concept of a presence agent, which is a new logical entity that is capable of accepting subscriptions, storing subscription state, and generating notifications when there are changes in user presence. The entity is defined as a logical one, since it is generally co-resident with another entity, and can even move around during the lifetime of a subscription.

This extension is also compliant with the Common Presence and Instant Messaging (CPIM) framework that has been defined in [4]. This allows SIP for presence to easily interwork with other presence systems compliant to CPIM.

2 Definitions

This document uses the terms as defined in [1]. Additionally, the following terms are defined and/or additionally clarified:

Presence User Agent (PUA): A Presence User Agent manipulates presence information for a presentity. In SIP terms, this means that a PUA generates REGISTER requests, conveying some kind of information about the presentity. We explicitly allow multiple PUAs per presentity. This means that a user can have many devices (such as a cell phone and PDA), each of which is independently generating a component of the overall presence information for a presentity. PUAs push data into the presence system, but are outside of it, in that they do not receive SUBSCRIBE messages, or send NOTIFY.

Presence Agent (PA): A presence agent is a SIP user agent which is capable of receiving SUBSCRIBE requests, responding to them, and generating notifications of changes in presence state. A presence agent must have complete knowledge of the presence state of a presentity. Typically, this is accomplished by co-locating the PA with the proxy/registrar, or the presence user agent of the presentity. A PA is always addressable with a SIP URL.

Presence Server: A presence server is a logical entity that can act as either a presence agent or as a proxy server for SUBSCRIBE requests. When acting as a PA, it is aware of the presence information of the presentity through some protocol means. This protocol means can be SIP REGISTER requests, but other mechanisms are allowed. When acting as a proxy, the SUBSCRIBE requests are proxied to another entity that may act as a PA.

Presence Client: A presence client is a presence agent that is colocated with a PUA. It is aware of the presence information of the presentity because it is co-located with the entity that manipulates this presence information.

3 Overview of Operation

In this section, we present an overview of the operation of this extension.

When an entity, the subscriber, wishes to learn about presence information from some user, it creates a SUBSCRIBE request. This request identifies the desired presentity in the request URI, using either a presence URL or a SIP URL. The subscription is carried along SIP proxies as any other INVITE would be. It eventually arrives at a presence server, which can either terminate the subscription (in which case it acts as the presence agent for the presentity), or proxy it on to a presence client. If the presence client handles the subscription, it is effectively acting as the presence agent for the presentity. The decision about whether to proxy or terminate the SUBSCRIBE is a local matter; however, we describe one way to effect such a configuration, using REGISTER.

The presence agent (whether in the presence server or presence client) first authenticates the subscription, then authorizes it. The means for authorization are outside the scope of this protocol, and we expect that many mechanisms will be used. Once authorized, the presence agent sends a 202 Accepted response. It also sends an immediate NOTIFY message containing the state of the presentity. As the state of the presentity changes, the PA generates NOTIFYs for all

subscribers.

The SUBSCRIBE message effectively establishes a session with the presence agent. As a result, the SUBSCRIBE can be record-routed, and rules for tag handling and Contact processing mirror those for INVITE. Similarly, the NOTIFY message is handled in much the same way a re-INVITE within a call leg is handled.

4 Naming

A presentity is identified in the most general way through a presence URI [4], which is of the form pres:user@domain. These URIs are protocol independent. Through a variety of means, these URIs can be resolved to determine a specific protocol that can be used to access the presentity. Once such a resolution has taken place, the presentity can be addressed with a sip URL of nearly identical form: sip:user@domain. The protocol independent form (the pres: URL) can be thought of as an abstract name, akin to a URN, which is used to identify elements in a presence system. These are resolved to concrete URLs that can be used to directly locate those entities on the network.

When subscribing to a presentity, the subscription can be addressed using the protocol independent form or the sip URL form. In the SIP context, "addressed" refers to the request URI. It is RECOMMENDED that if the entity sending a SUBSCRIBE is capable of resolving the protocol independent form to the SIP form, this resolution is done before sending the request. However, if the entity is incapable of doing this translation, the protocol independent form is used in the request URI. Performing the translation as early as possible means that these requests can be routed by SIP proxies that are not aware of the presence namespace.

The result of this naming scheme is that a SUBSCRIBE request is addressed to a user the exact same way an INVITE request would be addressed. This means that the SIP network will route these messages along the same path an INVITE would travel. One of these entities along the path may act as a PA for the subscription. Typically, this will either be the presence server (which is the proxy/registrar where that user is registered), or the presence client (which is one of the user agents associated with that presentity).

SUBSCRIBE messages also contain logical identifiers that define the originator and recipient of the subscription (the To and From header fields). Since these identifiers are logical ones, it is RECOMMENDED that these use the protocol independent format whenever possible. This also makes it easier to interwork with other systems which recognize these forms.

The Contact, Record-Route and Route fields do not identify logical entities, but rather concrete ones used for SIP messaging. As such, they MUST use the SIP URL forms in both SUBSCRIBE and NOTIFY.

5 Presence Event Package

The SIP event framework [3] defines an abstract SIP extension for subscribing to, and receiving notifications of, events. It leaves the definition of many additional aspects of these events to concrete extensions, also known as event packages. This extension qualifies as an event package. This section fills in the information required by [3].

5.1 Package Name

The name of this package is "presence". This name MUST appear within the Event header in SUBSCRIBE request and NOTIFY request. This section also serves as the IANA registration for the event package "presence".

TODO: Define IANA template in sub-notify and fill it in here.

Example:

```
Event: presence
```

5.2 SUBSCRIBE bodies

The body of a SUBSCRIBE request MAY contain a body. The purpose of the body depends on its type. In general, subscriptions will normally not contain bodies. The request URI, which identifies the presentity, combined with the event package name, are sufficient for user presence.

We anticipate that document formats could be defined to act as filters for subscriptions. These filters would indicate certain user presence events that would generate notifies, or restrict the set of data returned in NOTIFY requests. For example, a presence filter might specify that the notifications should only be generated when the status of the users instant message inbox changes. It might also say that the content of these notifications should only contain the IM related information.

5.3 Expiration

User presence changes as a result of events that include:

- o Turning on and off of a cell phone
- o Modifying the registration from a softphone
- o Changing the status on an instant messaging tool

These events are usually triggered by human intervention, and occur with a frequency on the order of minutes or hours. As such, it is subscriptions should have an expiration in the middle of this range, which is roughly one hour. Therefore, the default expiration time for subscriptions within this package is 3600 seconds. As per [3], the subscriber MAY include an alternate expiration time. Whatever the indicated expiration time, the server MAY reduce it but MUST NOT increase it.

5.4 NOTIFY Bodies

The body of the notification contains a presence document. This document describes the user presence of the presentity that was subscribed to. All subscribers MUST support the presence data format described in [fill in with IMPP document TBD], and MUST list its MIME type, [fill in with MIME type] in an Accept header present in the SUBSCRIBE request.

Other presence data formats might be defined in the future. In that case, the subscriptions MAY indicate support for other presence formats. However, they MUST always support and list [fill in with MIME type of IMPP presence document] as an allowed format.

Of course, the notifications generated by the presence agent MUST be in one of the formats specified in the Accept header in the SUBSCRIBE request.

5.5 Processing Requirements at the PA

User presence is highly sensitive information. Because the implications of divulging presence information can be severe, strong requirements are imposed on the PA regarding subscription processing, especially related to authentication and authorization.

A presence agent MUST authenticate all subscription requests. This authentication can be done using any of the mechanisms defined for SIP. It is not considered sufficient for the authentication to be transitive; that is, the authentication SHOULD use an end-to-end mechanism. The SIP basic authentication mechanism MUST NOT be used.

It is RECOMMENDED that any subscriptions that are not authenticated do not cause state to be established in the PA. This can be accomplished by generating a 401 in response to the SUBSCRIBE, and then discarding all state for that transaction. Retransmissions of the SUBSCRIBE generate the same response, guaranteeing reliability even over UDP.

Furthermore, a PA MUST NOT accept a subscription unless authorization has been provided by the presentity. The means by which authorization are provided are outside the scope of this document. Authorization may have been provided ahead of time through access lists, perhaps specified in a web page. Authorization may have been provided by means of uploading of some kind of standardized access control list document. Back end authorization servers, such as a DIAMETER [5], RADIUS [6], or COPS [7], can also be used. It is also useful to be able to query the user for authorization following the receipt of a subscription request for which no authorization information was present. Appendix A provides a possible solution for such a scenario.

The result of the authorization decision by the server will be reject, accept, or pending. Pending occurs when the server cannot obtain authorization at this time, and may be able to do so at a later time, when the presentity becomes available.

Unfortunately, if the server informs the subscriber that the subscription is pending, this will divulge information about the presentity - namely, that they have not granted authorization and are not available to give it at this time. Therefore, a PA SHOULD generate the same response for both pending and accepted subscriptions. This response SHOULD be a 202 Accepted response.

If the server informs the subscriber that the subscription is rejected, this also divulges information about the presentity - namely, that they have explicitly blocked the subscription previously, or are available at this time and chose to decline the subscription. If the policy of the server is not to divulge this information, the PA MAY respond with a 202 Accepted response even though the subscription is rejected. Alternatively, if the policy of the presentity or the PA is that it is acceptable to inform the subscriber of the rejection, a 603 Decline SHOULD be used.

Note that since the response to a subscription does not contain any useful information about the presentity, privacy and integrity of SUBSCRIBE responses is not deemed important.

5.6 Generation of Notifications

Upon acceptance of a subscription, the PA SHOULD generate an

immediate NOTIFY with the current presence state of the presentity.

If a subscription is received, and is marked as pending or was rejected, the PA SHOULD generate an immediate NOTIFY. This NOTIFY should contain a valid state for the presentity, yet be one which provides no useful information about the presentity. An example of this is to provide an IM URL that is the same form as the presence URL, and mark that IM address as "not available". The reason for this process of "lying" is that without it, a subscriber could tell the difference between a pending subscription and an accepted subscription based on the existence and content of an immediate NOTIFY. The approach defined here ensures that the presence delivered in a NOTIFY generated by a pending or rejected subscription is also a valid one that could have been delivered in a NOTIFY generated by an accepted subscription.

If the policy of the presence server or the presentity is that it is acceptable to divulge information about whether the subscription succeeded or not, the immediate NOTIFY need not be sent for pending or rejected subscriptions.

Of course, once a subscription is accepted, the PA SHOULD generate a NOTIFY for the subscription when it determines that the presence state of the presentity has changed. Section 6 describes how the PA makes this determination.

For reasons of privacy, it will frequently be necessary to encrypt the contents of the notifications. This can be accomplished using the standard SIP encryption mechanisms. The encryption should be performed using the key of the subscriber as identified in the From field of the SUBSCRIBE. Similarly, integrity of the notifications is important to subscribers. As such, the contents of the notifications SHOULD be authenticated using one of the standardized SIP mechanisms. Since the NOTIFY are generated by the presence server, which may not have access to the key of the user represented by the presentity, it will frequently be the case that the NOTIFY are signed by a third party. It is RECOMMENDED that the signature be by an authority over domain of the presentity. In other words, for a user pres:user@example.com, the signator of the NOTIFY SHOULD be the authority for example.com.

5.7 Rate Limitations on NOTIFY

For reasons of congestion control, it is important that the rate of notifications not become excessive. As a result, it is RECOMMENDED that the PA not generate notifications for a single presentity at a rate faster than once every 5 seconds.

5.8 Refresh Behavior

Since SUBSCRIBE is routed by proxies as any other method, it is possible that a subscription might fork. The result is that it might arrive at multiple devices which are configured to act as a PA for the same presentity. Each of these will respond with a 202 response to the SUBSCRIBE. Based on the forking rules in SIP, only one of these responses is passed to the subscriber. However, the subscriber will receive notifications from each of those PA which accepted the subscriptions. The SIP event framework allows each package to define the handling for this case.

The processing in this case is identical to the way INVITE would be handled. The 202 Accepted to the SUBSCRIBE will result in the installation of subscription state in the subscriber. The subscription is associated with the To and From (both with tags) and Call-ID from the 202. When notifications arrive, those from the PA's whose 202's were discarded in the forking proxy will not match the subscription ID stored at the subscriber (the From tags will differ). These SHOULD be responded to with a 481. This will disable the subscriptions from those PA. Furthermore, when refreshing the subscription, the refresh SHOULD make use of the tags from the 202 and make use of any Contact or Record-Route headers in order to deliver the SUBSCRIBE back to the same PA that sent the 202.

The result of this is that a presentity can have multiple PAs active, but these should be homogeneous, so that each can generate the same set of notifications for the presentity. Supporting heterogeneous

PAs, each of which generated notifications for a subset of the presence data, is complex and difficult to manage. If such a feature is needed, it can be accomplished with a B2BUA rather than through a forking proxy.

6 Publication

The user presence for a presentity can be obtained from any number of different ways. Baseline SIP defines a method that is used by all SIP clients - the REGISTER method. This method allows a UA to inform a SIP network of its current communications addresses (ie., Contact addresses) . Furthermore, multiple UA can independently register Contact addresses for the same SIP URL. These Contact addresses can be SIP URLs, or they can be any other valid URL.

Using the register information for presence is straightforward. The address of record in the REGISTER (the To field) identifies the presentity. The Contact headers define communications addresses that describe the state of the presentity. The use of the SIP caller preferences extension [8] is RECOMMENDED for use with UAs that are

interested in presence. It provides additional information about the Contact addresses that can be used to construct a richer presence document. The "description" attribute of the Contact header is explicitly defined here to be used as a free-form field that allows a user to define the status of the presentity at that communications address.

We also allow REGISTER requests to contain presence documents, so that the PUAs can publish more complex information.

Note that we do not provide for locking mechanisms, which would allow a client to lock presence state, fetch it, and update it atomically. We believe that this is not neeeded for the majority of use cases, and introduces substantial complexity. Most presence operations do not require get-before-set, since the SIP register mechanism works in such a way that data can be updated without a get.

The application of registered contacts to presence increases the requirements for authenticity. Therefore, REGISTER requests used by presence user agents SHOULD be authenticated using either SIP authentication mechanisms, or a hop by hop mechanism.

To indicate presence for instant messaging, the UA MAY either register contact addresses that are SIP URLs with the "methods" parameter set to indicate the method MESSAGE, or it MAY register an IM URL.

TODO: This section needs work. Need to define a concrete example of mapping a register to a presence document, once IMPP generates the document format.

6.1 Migrating the PA Function

It is important to realize that the PA function can be colocated with several elements:

- o It can be co-located with the proxy server handling registrations for the presentity. In this way, the PA knows the presence of the user through registrations.

- o It can be co-located with a PUA for that presentity. In the case of a single PUA per presentity, the PUA knows the state of the presentity by sheer nature of its co-location.

- o It can be co-located in any proxy along the call setup path. That proxy can learn the presence state of the presentity by generating its own SUBSCRIBE in order to determine it. In this case, the PA is effectively a B2BUA.

Because of the soft-state nature of the subscriptions, it becomes possible for the PA function to migrate during the lifetime of a subscription. The most workable scenario is for the PA function to migrate from the presence server to the PUA, and back.

Consider a subscription that is installed in a presence server. Assume for the moment that the presence server can determine that a downstream UA is capable of acting as a PA for the presentity. When a subscription refresh arrives, the PA destroys its subscription, and then acts as a proxy for the subscription. The subscription is then routed to the UA, where it can be accepted. The result is that the subscription becomes installed in the PUA.

For this migration to work, the PUA MUST be prepared to accept SUBSCRIBE requests which already contain tags in the To field. Furthermore, the PUA MUST insert a Contact header into the 202, and this header MUST be used by the subscriber to update the contact address for the subscription.

TODO: Does this work? What about getting a Record-Route in place at the PUA. This might only be possible for refreshes that don't use Route or tags.

The presence server determines that a PUA is capable of supporting a PA function through the REGISTER message. Specifically, if a PUA wishes to indicate support for the PA function, it SHOULD include a contact address in its registration with a caller preferences "methods" parameter listing SUBSCRIBE.

7 Mapping to CPIM

This section defines how a SIP for presence messages are converted to CPIM, and how a CPIM messages are converted to SIP for presence. SIP to CPIM conversion occurs when a SIP system sends a SUBSCRIBE request that contains a pres URL or SIP URL that corresponds to a user in a domain that runs a different presence protocol. CPIM to SIP involves the case where a user in a different protocol domain generates a subscription that is destined for a user in a SIP domain.

Note that the process defined below requires that the gateway store subscription state. This unfortunate result is due to the need to remember the Call-ID, CSeq, and Route headers for subscriptions from the SIP side, so that they can be inserted into the SIP NOTIFY generated when a CPIM notification arrives.

7.1 SIP to CPIM

SIP for presnce is converted to CPIM through a SIP to CPIM abstract

gateway service, depicted in Figure 1.

```
                         +-------------+
                         |             |
                         |  SIP to CPIM|
                         |  Conversion |
                         |             |
      SIP                |             |    CPIM
     ---------------->   |             | ---------------->
                         |             |
                         |             |
                         |             |
                         |             |
                         |             |
                         |             |
                         +-------------+
```

Figure 1: SIP to CPIM Conversion

The first step is that a SUBSCRIBE request is received at a gateway. The gateway generates a CPIM subscription request, with its parameters filled in as follows:

- o The watcher identity in the CPIM message is copied from the From field of the SUBSCRIBE. If the From field contains a SIP

URL, it is converted to an equivalent pres URL by dropping all SIP URL parameters, and changing the scheme to pres.

This conversion may not work - what if the SIP URL has no user name. Plus, converting from a URL back to a URN in this fashion may not do it correctly.

o The target identity in the CPIM message is copied from the Request-URI field of the SUBSCRIBE. This may need to be converted to a pres URL as well.

o The duration parameter in the CPIM message is copied from the Expires header in the SUBSCRIBE. If the Expires header specifies an absolute time, it is converted to a delta-time by the gateway. If no Expires header is present, one hour is assumed.

o The transID parameter in the CPIM message is constructed by appending the Call-ID, the URI in the To field, the URI in the From field, the CSeq and the tag in the From field, and the request URI, and computing a hash of the resulting string. This hash is used as the transID. Note that the request URI is included in the hash. This is to differentiate forked requests within the SIP network that may arrive at the same gateway.

The CPIM service then responds with either a success or failure. In the case of success, the SIP to CPIM gateway service generates a 202 response to the SUBSCRIBE. It adds a tag to the To field in the response, which is the same as the transID field in the success response. The 202 response also contains a Contact header, which is the value of the target from the SUBSCRIBE request. It is important that the Contact header be set to the target, since that makes sure that subscription refreshes have the same value in the request URI as the original subscription. The duration value from the CPIM success response is placed into the Expires header of the 202. The gateway stores the Call-ID and Route header set for this subscription.

If the CPIM service responds with a failure, the SIP to CPIM gateway generates a 603 response. It adds a tag to the To field in the response, which is the same as the transID field in the failure response.

When the CPIM system generates a notification request, the SIP to CPIM gateway creates a SIP NOTIFY request. The request is constructed using the standard RFC2543 [2] procedures for constructing a request within a call leg. This will result in the To field containing the watcher field from CPIM, and the From field containing the target field from the CPIM notification. The tag in the From field will contain the transID. The presence information is copied into the body of the notification. The Call-ID and Route headers are constructed from the subscription state stored in the gateway. If no notification has yet been generated for this subscription, an initial CSeq value

is selected and stored.

SUBSCRIBE refreshes are handled identically to initial subscriptions as above.

If a subscription is received with an Expires of zero, the SIP to CPIM gateway generates an unsubscribe message into the the CPIM system. The watcher parameter is copied from the From field of the SUBSCRIBE. The target parameter is copied from the Request URI field of the SUBSCRIBE. The transID is copied from the tag in the To field of the SUBSCRIBE request.

The response to an unsubscribe is either success or failure. In the case of success, a 202 response is constructed in the same fashion as above for a success response to a CPIM subscriber. All subscription state is removed. In the case of failure, a 603 response is constructed in the same fashion as above, and then subscription state is removed, if present.

7.2 CPIM to SIP

CPIM to SIP conversion occurs when a CPIM subscription request arrives on the CPIM side of the gateway. This scenario is shown in Figure 2.

The CPIM subscription request is converted into a SIP SUBSCRIBE request. To do that, the first step is to determine if the subscribe is for an existing subscription. That is done by taking the target in the CPIM subscription request, and matching it against targets for existing subscriptions. If there are none, it is a new subscription, otherwise, its a refresh.

If its a new subscription, the gateway generates a SIP SUBSCRIBE request in the following manner:

- o The From field in the request is set to the watcher field in the CPIM subscription request

- o The To field in the request is set to the target field in the CPIM subscription request

- o The Expires header in the SUBSCRIBE request is set to the duration field in the CPIM subscription request

- o The tag in the From field is set to the transID in the CPIM subscription request.

```
                         +-------------+
                         |             |
                         | CPIM to SIP |
                         |  Conversion |
                         |             |
       SIP SUBSCRIBE     |             |    CPIM subscription request
      <----------------> |             | <----------------->
                         |             |
                         |             |
                         |             |
                         |             |
                         |             |
                         |             |
                         +-------------+
```

Figure 2: CPIM to SIP Conversion

This SUBSCRIBE message is then sent.

If the subscription is a refresh, a SUBSCRIBE request is generated in the same way. However, there will also be a tag in the To field, copied from the subscription state in the gateway, and a Route header, obtained from the subscription state in the gateway.

When a response to the SUBSCRIBE is received, a response is sent to the CPIM system. The duration parameter in this response is copied from the Expires header in the SUBSCRIBE response (a conversion from an absolute time to delta time may be needed). The transID in the response is copied from the tag in the From field of the response. If the response was 202, the status is set to indeterminate. If the response was any other 200 class response, the status is set to sucess. For any other final response, the status is set to failure.

If the response was a 200 class response, subscription state is established. This state contains the tag from the To field in the SUBSCRIBE response, and the Route header set computed from the Record-Routes and Contact headers in the 200 class response. The subscription is indexed by the presentity identification (the To field of the SUBSCRIBE that was generated).

If an unsubscribe request is received from the CPIM side, the gateway checks if the subscription exists. If it does, a SUBSCRIBE is generated as described above. However, the Expires header is set to zero. If the subscription does not exist, the gateway generates a failure response and sends it to the CPIM system. When the response to the SUBSCRIBE request arrives, it is converted to a CPIM response as described above for the initial SUBSCRIBE response. In all cases, any subscription state in the gateway is destroyed.

When a NOTIFY is received from the SIP system, a CPIM notification request is sent. This notification is constructed as follows:

- o The CPIM watcher is set to the URI in the To field of the NOTIFY.
- o The CPIM target is set to the URI in the From field of the NOTIFY.
- o The transID is computed using the same mechanism as for the SUBSCRIBE in Section 7.1
- o The presence component of the notification is extracted from the body of the SIP NOTIFY request.

The gateway generates a 200 response to the SIP NOTIFY and sends it as well.

TODO: some call flow diagrams with the parameters

8 Firewall and NAT Traversal

It is anticipated that presence services will be used by clients and presentities that are connected to proxy servers on the other side of firewalls and NATs. Fortunately, since the SIP presence messages do not establish independent media streams, as INVITE does, firewall and NAT traversal is much simpler than described in [9] and [10].

Generally, data traverses NATs and firewalls when it is sent over TCP or TLS connections established by devices inside the firewall/NAT to devices outside of it. As a result, it is RECOMMENDED that SIP for presence entities maintain persistent TCP or TLS connections to their next hop peers. This includes connections opened to send a SUBSCRIBE, NOTIFY, and most importantly, REGISTER. By keeping the latter connection open, it can be used by the SIP proxy to send messages from outside the firewall/NAT back to the client. It is also recommended that the client include a Contact cookie as described in [10] in their registration, so that the proxy can map the presentity URI to that connection.

Furthermore, entities on either side of a firewall or NAT should record-route in order to ensure that the initial connection established for the subscription is used for the notifications as well.

9 Security considerations

There are numerous security considerations for presence. Many are outlined above; this section considers them issue by issue.

9.1 Privacy

Privacy encompasses many aspects of a presence system:

- Subscribers may not want to reveal the fact that they have subscribed to certain users

- Users may not want to reveal that they have accepted subscriptions from certain users

- Notifications (and fetch results) may contain sensitive data which should not be revealed to anyone but the subscriber

Privacy is provided through a combination of hop by hop encryption and end to end encryption. The hop by hop mechanisms provide scalable privacy services, disable attacks involving traffic analysis, and hide all aspects of presence messages. However, they operate based on transitivity of trust, and they cause message content to be revealed to proxies. The end-to-end mechanisms do not require transitivity of trust, and reveal information only to the desired recipient. However, end-to-end encryption cannot hide all information, and is susceptible to traffic analysis. Strong end to end authentication and encryption also requires that both participants have public keys, which is not generally the case. Thus, both mechanisms combined are needed for complete privacy services.

SIP allows any hop by hop encryption scheme. It is RECOMMENDED that between network servers (proxies to proxies, proxies to redirect servers), transport mode ESP [11] is used to encrypt the entire message. Between a UAC and its local proxy, TLS [12] is RECOMMENDED. Similarly, TLS SHOULD be used between a presence server and the PUA.

The presence server can determine whether TLS is supported by the receiving client based on the transport parameter in the Contact header of its registration. If that registration contains the token "tls" as transport, it implies that the PUA supports TLS.

Furthermore, we allow for the Contact header in the SUBSCRIBE request to contain TLS as a transport. The Contact header is used to route subsequent messages between a pair of entities. It defines the address and transport used to communicate with the user agent. Even though TLS might be used between the subscriber and its local proxy, placing this parameter in the Contact header means that TLS can also be used end to end for generation of notifications after the initial SUBSCRIBE message has been successfully routed. This would provide end to end privacy and authentication services with low proxy overheads.

SIP encryption MAY be used end to end for the transmission of both SUBSCRIBE and NOTIFY requests. SIP supports PGP based encryption, which does not require the establishment of a session key for encryption of messages within a given subscription (basically, a new session key is established for each message as part of the PGP encryption). Work has recently begun on the application of S/MIME [13] for SIP.

9.2 Message integrity and authenticity

It is important for the message recipient to ensure that the message contents are actually what was sent by the originator, and that the recipient of the message be able to determine who the originator really is. This applies to both requests and responses of SUBSCRIBE and NOTIFY. This is supported in SIP through end to end authentication and message integrity. SIP provides PGP based authentication and integrity (both challenge-response and public key signatures), and http basic and digest authentication. HTTP Basic is NOT RECOMMENDED.

9.3 Outbound authentication

When local proxies are used for transmission of outbound messages, proxy authentication is RECOMMENDED. This is useful to verify the identity of the originator, and prevent spoofing and spamming at the originating network.

9.4 Replay prevention

To prevent the replay of old subscriptions and notifications, all signed SUBSCRIBE and NOTIFY requests and responses MUST contain a Date header covered by the message signature. Any message with a date

older than several minutes in the past, or more than several minutes into the future, SHOULD be discarded.

Furthermore, all signed SUBSCRIBE and NOTIFY requests MUST contain a Call-ID and CSeq header covered by the message signature. A user agent or presence server MAY store a list of Call-ID values, and for each, the higest CSeq seen within that Call-ID. Any message that arrives for a Call-ID that exists, whose CSeq is lower than the highest seen so far, is discarded.

Finally, challenge-response authentication (http digest or PGP) MAY be used to prevent replay attacks.

9.5 Denial of service attacks

Denial of service attacks are a critical problem for an open, inter-domain, presence protocol. Here, we discuss several possible attacks, and the steps we have taken to prevent them.

9.5.1 Smurf attacks through false contacts

Unfortunately, presence is a good candidate for smurfing attacks because of its amplification properties. A single SUBSCRIBE message could generate a nearly unending stream of notifications, so long as a suitably dynamic source of presence data can be found. Thus, a simple way to launch an attack is to send subscriptions to a large number of users, and in the Contact header (which is where notifications are sent), place the address of the target.

The only reliable way to prevent these attacks is through authentication and authorization. End users will hopefully not accept subscriptions from random unrecognized users. Also, the presence client software could be programmed to warn the user when the Contact header in a SUBSCRIBE is from a domain which does not match that of the From field (which identifies the subscriber).

Also, note that as described in [3], if a NOTIFY is not acknowledged or was not wanted, the subscription that generated it is removed. This eliminates the amplification properties of providing false Contact addresses.

10 Example message flows

The following subsections exhibit example message flows, to further clarify behavior of the protocol.

10.1 Client to Client Subscription with Presentity State Changes

This call flow illustrates subscriptions and notifications that do not involve a presence server.

The watcher subscribes to the presentity, and the subscription is accepted, resulting in a 202 Accepted response. The presentity subsequently changes state (is on the phone), resulting in a new notification. The flow finishes with the watcher canceling the subscription.

```
            Watcher                        Presentity
            -------                        -----------
               |       F1 SUBSCRIBE             |
               | ------------------------------>|
               |       F2 202 Accepted          |
               |<-------------------------------|
               |       F3 NOTIFY                |
               |<-------------------------------|
               |       F4 200 OK                |
               |------------------------------->|
               |       F5 NOTIFY                |
               |<-------------------------------|
               |       F6 200 OK                |
               |------------------------------->|
               |       F7 SUBSCRIBE (unsub)     |
               |------------------------------->|
               |       F8 202 Accepted          |
               |<-------------------------------|
```

Message Details

F1 SUBSCRIBE watcher -> presentity

```
SUBSCRIBE sip:presentity@pres.example.com SIP/2.0
Via: SIP/2.0/UDP  watcherhost.example.com:5060
From: User <pres:user@example.com>
To: Resource <pres:presentity@example.com>
Call-ID: 3248543@watcherhost.example.com
CSeq : 1 SUBSCRIBE
Expires: 600
Accept: application/xpidf+xml
Event: presence
Contact: sip:user@watcherhost.example.com
```


F2 202 Accepted presentity->watcher

```
SIP/2.0 202 Accepted
Via: SIP/2.0/UDP watcherhost.example.com:5060
From: User <pres:user@example.com>
To: Resource <pres:presentity@example.com>;tag=88a7s
Call-ID: 3248543@watcherhost.example.com
Cseq: 1 SUBSCRIBE
Event: presence
Expires: 600
Contact: sip:presentity@pres.example.com
```

F3 NOTIFY Presentity->watcher

```
NOTIFY sip:user@watcherhost.example.com SIP/2.0
Via: SIP/2.0/UDP pres.example.com:5060
From: Resource <pres:presentity@example.com>;tag=88a7s
To: User <pres:user@example.com>
Call-ID: 3248543@watcherhost.example.com
CSeq: 1 NOTIFY
Event: presence
Content-Type: application/xpidf+xml
Content-Length: 120

<?xml version="1.0"?>
<presence entityInfo="pres:presentity@example.com">
  <tuple destination="sip:presentity@example.com" status="open"/>
</presence>
```

F4 200 OK watcher->presentity

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP pres.example.com:5060
From: Resource <pres:presentity@example.com>
To: User <pres:user@example.com>
Call-ID: 3248543@watcherhost.example.com
CSeq: 1 NOTIFY
```


F5 NOTIFY Presentity->watcher

```
NOTIFY sip:user@watcherhost.example.com SIP/2.0
Via: SIP/2.0/UDP pres.example.com:5060
From: Resource <pres:presentity@example.com>
To: User <pres:user@example.com>
Call-ID: 3248543@watcherhost.example.com
CSeq: 2 NOTIFY
Event: presence
Content-Type: application/xpidf+xml
Content-Length: 120

<?xml version="1.0"?>
<presence entityInfo="pres:presentity@example.com">
  <tuple destination="sip:presentity@example.com" status="closed"/>
</presence>
```

F6 200 OK watcher->presentity

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP pres.example.com:5060
From: Resource <pres:presentity@example.com>
To: User <pres:user@example.com>
Call-ID: 3248543@watcherhost.example.com
CSeq: 2 NOTIFY
```

F7 SUBSCRIBE watcher -> presentity

```
SUBSCRIBE sip:presentity@pres.example.com SIP/2.0
Via: SIP/2.0/UDP  watcherhost.example.com:5060
From: User <pres:user@example.com>
To: Resource <pres:presentity@example.com>
Call-ID: 3248543@watcherhost.example.com
Event: presence
CSeq : 2 SUBSCRIBE
Expires: 0
Accept: application/xpidf+xml
Contact: sip:user@watcherhost.example.com
```


F8 202 Accepted presentity->watcher

```
SIP/2.0 202 Accepted
Via: SIP/2.0/UDP watcherhost.example.com:5060
From: User <pres:user@example.com>
To: Resource <pres:presentity@example.com>
Call-ID: 3248543@watcherhost.example.com
Event: presence
Cseq: 2 SUBSCRIBE
Expires:0
```

10.2 Presence Server with Client Notifications

This call flow shows the involvement of a presence server in the handling of subscriptions. In this scenario, the client has indicated that it will handle subscriptions and thus notifications. The message flow shows a change of presence state by the client and a cancellation of the subscription by the watcher.

```
                              Presence
       Watcher                 Server                  PUA
          |                      |  F1 REGISTER          |
          |                      |<----------------------|
          |                      |  F2 200 OK            |
          |                      |---------------------->|
          |  F3 SUBSCRIBE        |                       |
          |--------------------->|                       |
          |                      |  F4 SUBSCRIBE         |
          |                      |---------------------->|
          |                      |  F5 202               |
          |                      |<----------------------|
          |  F6 202              |                       |
          |<---------------------|                       |
          |  F7 NOTIFY           |                       |
          |<---------------------------------------------+
          |  F8  200 OK          |                       |
          |--------------------------------------------->|
          |                      |  F9 REGISTER          |
          |                      |<----------------------|
          |                      |  F10 200 OK           |
          |                      |---------------------->|
          |  F11 NOTIFY          |                       |
```



```
          |<---------------------------------------------+
          |  F12 200 OK          |                       |
          |--------------------------------------------->|
```

Message Details

F1 REGISTER PUA->server

```
REGISTER sip:example.com SIP/2.0
Via: SIP/2.0/UDP pua.example.com:5060
To: <sip:resource@example.com>
From: <sip:resource@example.com>
Call-ID: 2818@pua.example.com
CSeq: 1 REGISTER
Contact: <sip:id@pua.example.com>;methods="MESSAGE"
          ;description="open"
Contact: <sip:id@pua.example.com>;methods="SUBSCRIBE"
Expires: 600
```

F2 200 OK   server->PUA

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP pua.example.com:5060
To: <sip:resource@example.com>
From: <sip:resource@example.com>
Call-ID: 2818@pua.example.com
CSeq: 1 REGISTER
Contact: <sip:id@pua.example.com>;methods="MESSAGE"
          ;description="open"
Contact: <sip:id@pua.example.com>;methods="SUBSCRIBE"
Expires: 600
```

F3 SUBSCRIBE watcher->server


```
SUBSCRIBE sip:resource@example.com SIP/2.0
Via: SIP/2.0/UDP  watcherhost.example.com:5060
From: User <pres:user@example.com>
To: Resource <pres:resource@example.com>
Call-ID: 32485@watcherhost.example.com
CSeq : 1 SUBSCRIBE
Expires: 600
Event: presence
Accept: application/xpidf+xml
Contact: sip:user@watcherhost.example.com
```

F4 SUBSCRIBE server->PUA

```
SUBSCRIBE sip:id@pua.example.com SIP/2.0
Via: SIP/2.0/UDP server.example.com:5060
Via: SIP/2.0/UDP watcherhost.example.com:5060
From: User <pres:user@example.com>
To: Resource <pres:resource@example.com>
Call-ID: 32485@watcherhost.example.com
CSeq : 1 SUBSCRIBE
Event: presence
Expires: 600
Accept: application/xpidf+xml
Contact: sip:user@watcherhost.example.com
```

F5 202 Accepted   PUA->server

```
SIP/2.0 202 Accepted
Via: SIP/2.0/UDP server.example.com:5060
Via: SIP/2.0/UDP watcherhost.example.com:5060
From: User <pres:user@example.com>
To: Resource <pres:resource@example.com>;tag=ffd2
Call-ID: 32485@watcherhost.example.com
CSeq : 1 SUBSCRIBE
Event: presence
Expires: 600
```


F6 200 OK   server->watcher

```
SIP/2.0 202 Accepted
Via: SIP/2.0/UDP watcherhost.example.com:5060
From: User <pres:user@example.com>
To: Resource <pres:resource@example.com>;tag=ffd2
Call-ID: 32485@watcherhost.example.com
CSeq : 1 SUBSCRIBE
Event: presence
Expires: 600
```

F7 NOTIFY   PUA->watcher

```
NOTIFY sip:user@watcherhost.example.com SIP/2.0
Via: SIP/2.0/UDP pua.example.com:5060
To: User <pres:user@example.com>
From: Resource <pres:resource@example.com>;tag=ffd2
Call-ID: 32485@watcherhost.example.com
CSeq : 1 NOTIFY
Event: presence
Content-Type: application/xpidf+xml
Content-Length: 120

<?xml version="1.0"?>
<presence entityInfo="pres:resource@example.com">
  <tuple destination="im:resource@example.com" status="open"/>
</presence>
```

F8 200 OK   watcher->PUA

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP pua.example.com:5060
To: User <pres:user@example.com>
From: Resource <pres:resource@example.com>;tag=ffd2
Call-ID: 32485@watcherhost.example.com
CSeq : 1 NOTIFY
```


F9 REGISTER PUA->server

```
REGISTER sip:example.com SIP/2.0
Via: SIP/2.0/UDP pua.example.com:5060
To: <sip:resource@example.com>
From: <sip:resource@example.com>
Call-ID: 2818@pua.example.com
CSeq: 2 REGISTER
Contact: <sip:id@pua.example.com>;methods="MESSAGE"
          ;description="busy"
Contact: <sip:id@pua.example.com>;methods="SUBSCRIBE"
Expires: 600
```

F10 200 OK   server->PUA

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP pua.example.com:5060
To: <sip:resource@example.com>
From: <sip:resource@example.com>
Call-ID: 2818@pua.example.com
CSeq: 2 REGISTER
Contact: <sip:id@pua.example.com>;methods="MESSAGE"
          ;description="busy"
Contact: <sip:id@pua.example.com>;methods="SUBSCRIBE"
Expires: 600
```

F11 NOTIFY   PUA->watcher

```
NOTIFY sip:user@watcherhost.example.com SIP/2.0
```

```
Via: SIP/2.0/UDP pua.example.com:5060
To: User <pres:user@example.com>
From: Resource <pres:resource@example.com>;tag=ffd2
Call-ID: 32485@watcherhost.example.com
CSeq : 2 NOTIFY
Event: presence
Content-Type: application/xpidf+xml
Content-Length: 120
```



```
<?xml version="1.0"?>
<presence entityInfo="pres:resource@example.com">
  <tuple destination="im:resource@example.com" status="busy"/>
</presence>
```

F12 200 OK watcher->PUA

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP pua.example.com:5060
To: User <pres:user@example.com>
From: Resource <pres:resource@example.com>;tag=ffd2
Call-ID: 32485@watcherhost.example.com
CSeq : 2 NOTIFY
```

10.3 Presence Server Notifications

This message flow illustrates how the presence server can be the responsible for sending notifications for a presentity. The presence server will do this if the presentity has not sent a registration indicating an interest in handling subscriptions. This flow assumes that the watcher has previously been authorized to subscribe to this resource at the server.

```
Watcher              Server                  PUA
   | F1 SUBSCRIBE      |                      |
   |------------------>|                      |
   | F2 202 Accepted   |                      |
   |<------------------|                      |
   | F3 NOTIFY         |                      |
   |<------------------|                      |
   | F4 200 OK         |                      |
   |------------------>|                      |
```

```
   |                    |  F5 REGISTER       |
   |                    |<-------------------|
   |                    |  F6 200 OK         |
   |                    |------------------->|
```



```
   | F7 NOTIFY          |                    |
   |<-------------------|                    |
   | F8 200 OK          |                    |
   |------------------->|                    |
```

Message Details

F1 SUBSCRIBE   watcher->server

```
SUBSCRIBE sip:resource@example.com SIP/2.0
Via: SIP/2.0/UDP watcherhost.example.com:5060
To: <pres:resource@example.com>
From: <pres:user@example.com>
Call-ID: 2010@watcherhost.example.com
CSeq: 1 SUBSCRIBE
Event: presence
Accept: application/xpidf+xml
Contact: <sip:user@watcherhost.example.com>
Expires: 600
```

F2 202 OK   server->watcher

```
SIP/2.0 202 Accepted
Via: SIP/2.0/UDP watcherhost.example.com:5060
To: <pres:resource@example.com>;tag=ffd2
From: <pres:user@example.com>
Call-ID: 2010@watcherhost.example.com
CSeq: 1 SUBSCRIBE
Event: presence
Expires: 600
Contact: sip:example.com
```

F3 NOTIFY server-> watcher

```
NOTIFY sip:user@watcherhost.example.com SIP/2.0
Via: SIP/2.0/UDP server.example.com:5060
```

```
      From: <pres:resource@example.com>;tag=ffd2
      To: <pres:user@example.com>
      Call-ID: 2010@watcherhost.example.com
      Event: presence
      CSeq: 1 NOTIFY
      Content-Type: application/xpidf+xml
      Content-Length: 120

      <?xml version="1.0"?>
      <presence entityInfo="pres:resource@example.com">
        <tuple destination="im:resource@example.com" status="open"/>
      </presence>
```

F4 200 OK

```
      SIP/2.0 200 OK
      Via: SIP/2.0/UDP server.example.com:5060
      From: <pres:resource@example.com>;tag=ffd2
      To: <pres:user@example.com>
      Call-ID: 2010@watcherhost.example.com
      CSeq: 1 NOTIFY
```

F5 REGISTER

```
      REGISTER sip:example.com SIP/2.0
      Via: SIP/2.0/UDP pua.example.com:5060
      To: <sip:resource@example.com>
      From: <sip:resource@example.com>
      Call-ID: 110@pua.example.com
      CSeq: 2 REGISTER
      Contact: <sip:id@pua.example.com>;methods="MESSAGE"
                ;description="Away from keyboard"
      Expires: 600
```

F6 200 OK

```
Via: SIP/2.0/UDP pua.example.com:5060
To: <sip:resource@example.com>
From: <sip:resource@example.com>
Call-ID: 110@pua.example.com
CSeq: 2 REGISTER
Contact: <sip:id@pua.example.com>;methods="MESSAGE"
          ; description="Away from keyboard"
          ; expires=600
```

F7 NOTIFY

```
NOTIFY sip:user@watcherhost.example.com SIP/2.0
Via: SIP/2.0/UDP server.example.com:5060
From: <pres:resource@example.com>;tag=ffd2
To: <pres:user@example.com>
Call-ID: 2010@watcherhost.example.com
CSeq: 2 NOTIFY
Event: presence
Content-Type: application/xpidf+xml
Content-Length: 120

<?xml version="1.0"?>
<presence entityInfo="pres:resource@example.com">
  <tuple destination="im:resource@example.com" status="Away from keyboard"/>
</presence>
```

F8 200 OK

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP server.example.com:5060
From: <sip:resource@example.com>;tag=ffd2
To: <pres:user@example.com>
Call-ID: 2010@watcherhost.example.com
CSeq: 2 NOTIFY
```

11 Open Issues

The following is the list of known open issues:

- o This draft recommends that the To and From field are populated with presence URLs rather than sip URLs. Is that reasonable? Will this lead to incompatibilities in proxies? Is there any issues with CPIM if the SIP URL format is used? This depends on what components of a message are signed in CPIM.

- o Rate limitations on NOTIFY: do we want that? How do we pick a value? 5 seconds is arbitrary.

- o Merging of presence data from multiple PA has been removed. Is that OK?

- o Placing IM URLs in the Contact header of a REGISTER: is that OK? What does it mean?

- o The process of migrating subscriptions from a presence server to PUA is not likely to work in the case where subscription refreshes use tags and Route headers. So, we have a choice. Either migration is disallowed, and we keep with leg oriented subscriptions, or migration is allowed, and there is no tags or Route's associated with subscriptions.

- o Converting SIP URLs back to pres URLs.

- o The SIP to CPIM and CPIM to SIP gateways are not stateless, because of the need to maintain Route, Call-ID, CSeq, and other parameters. Perhaps we can ask CPIM to define a token value which is sent in a CPIM request and returned in a CPIM response. Would that help?

- o Need to specify how to take Contacts from REGISTER and build a presence document. One obvious thing is that the contact addresses don't go in there directly; you probably want to put the address of record, otherwise calls might not go through the proxy.

12 Changes from -00

The document has been completely rewritten, to reflect the change from a sales pitch and educational document, to a more formal protocol specification. It has also been changed to align with the SIP event architecture and with CPIM. The specific protocol changes resulting from this rewrite are:

- o The Event header must now be used in the SUBSCRIBE and NOTIFY requests.

- o The SUBSCRIBE message can only have a single Contact header.

-00 allowed for more than one.

o The From and To headers can contain presence URIs.

o The Request-URI can contain a presence URI.

o Subscriptions are responded to with a 202 if they are pending or accepted.

o Presence documents are not returned in the body of the SUBSCRIBE response. Rather, they are sent in a separate NOTIFY. This more cleanly separates subscription and notification, and is mandated by alignment with CPIM.

o Authentication is now mandatory at the PA. Authorization is now mandatory at the PA.

o Fake NOTIFY is sent for pending or rejected subscriptions.

o A rate limit on notifications was introduced.

o Merging of presence data has been removed.

o The subscriber rejects notifications received with tags that don't match those in the 202 response to the SUBSCRIBE. This means that only one PA will hold subscription state for a particular subscriber for a particular presentity.

o IM URLs allowed in Contacts in register

o CPIM mappings defined.

o Persistent connections recommended for firewall traversal.

Obtaining Authorization

When a subscription arrives at a PA, the subscription needs to be authorized by the presentity. In some cases, the presentity may have provided authorization ahead of time. However, in many cases, the subscriber is not pre-authorized. In that case, the PA needs to query the presentity for authorization.

In order to do this, we define an implicit subscription at the PA. This subscription is for a virtual presentity, which is the "set of

subscriptions for presentity X", and the subscriber to that virtual presentity is X itself. Whenever a subscription is received for X, the virtual presentity changes state to reflect the new subscription for X. This state changes for subscriptions that are approved and for ones that are pending. As a result of this, when a subscription arrives for which authorization is needed, the state of the virtual presentity changes to indicate a pending subscription. The entire state of the virtual presentity is then sent to the subscriber (the presentity itself). This way, the user behind that presentity can see that there are pending subscriptions. It can then use some non-SIP means to install policy in the server regarding this new user. This policy is then used to either accept or reject the subscription.

A call flow for this is shown in Figure 3.

In the case where the presentity is not online, the problem is also straightforward. When the user logs into their presence client, it can fetch the state of the virtual presentity for X, check for pending subscriptions, and for each of them, upload a new policy which indicates the appropriate action to take.

A data format to represent the state of these virtual presentities can be found in [14].

A Acknowledgements

We would like to thank the following people for their support and comments on this draft:

```
Rick Workman     Nortel
Adam Roach       Ericsson
Sean Olson       Ericsson
Billy Biggs      University of Waterloo
Stuart Barkley   UUNet
Mauricio Arango  SUN
Richard Shockey  Shockey Consulting LLC
Jorgen Bjorkner  Hotsip
Henry Sinnreich  MCI Worldcom
Ronald Akers     Motorola
```


B Authors Addresses

Jonathan Rosenberg


```
          |  SUBSCRIBE X          |                        |
          | --------------------> |                        |
          |                       |                        |
          |  202 Accepted         |                        |
          | <-------------------- | NOTIFY X-subscriptions|
          |                       |----------------------->|
          |                       |                        |
          |                       | 200 OK                 |
```

```
          |                      |<----------------------|
          |                      |                       |
          |                      |                       |
          |                      | HTTP POST w/ policy   |
          |                      |<----------------------|
          |                      |                       |
          |                      | 200 OK                |
          |                      |---------------------->|
          |                      |                       |
          |                      |                       |
          |                      |                       |
```

Figure 3: Sequence diagram for online authorization

dynamicsoft
72 Eagle Rock Avenue
First Floor
East Hanover, NJ 07936
email: jdrosen@dynamicsoft.com

Dean Willis
dynamicsoft
5100 Tennyson Parkway
Suite 1200
Plano, Texas 75024
email: dwillis@dynamicsoft.com

Robert Sparks
dynamicsoft
5100 Tennyson Parkway

Suite 1200
Plano, Texas 75024
email: rsparks@dynamicsoft.com

Ben Campbell
5100 Tennyson Parkway
Suite 1200
Plano, Texas 75024
email: bcampbell@dynamicsoft.com

Henning Schulzrinne
Columbia University
M/S 0401
1214 Amsterdam Ave.
New York, NY 10027-7003
email: schulzrinne@cs.columbia.edu

Jonathan Lennox
Columbia University
M/S 0401
1214 Amsterdam Ave.
New York, NY 10027-7003
email: lennox@cs.columbia.edu

Christian Huitema
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052-6399
email: huitema@microsoft.com

Bernard Aboba
Microsoft Corporation

One Microsoft Way
Redmond, WA 98052-6399
email: bernarda@microsoft.com

David Gurle
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052-6399
email: dgurle@microsoft.com

David Oran
Cisco Systems
170 West Tasman Dr.
San Jose, CA 95134
email: oran@cisco.com

C Bibliography

[1] M. Day, J. Rosenberg, and H. Sugano, "A model for presence and instant messaging," Request for Comments 2778, Internet Engineering Task Force, Feb. 2000.

[2] M. Handley, H. Schulzrinne, E. Schooler, and J. Rosenberg, "SIP: session initiation protocol," Request for Comments 2543, Internet Engineering Task Force, Mar. 1999.

[3] A. Roach, "Event notification in SIP," Internet Draft, Internet Engineering Task Force, Oct. 2000. Work in progress.

[4] D. Crocker et al. , "A common profile for instant messaging (CPIM)," Internet Draft, Internet Engineering Task Force, Nov. 2000. Work in progress.

[5] P. Calhoun, A. Rubens, H. Akhtar, and E. Guttman, "DIAMETER base protocol," Internet Draft, Internet Engineering Task Force, Sept. 2000. Work in progress.

[6] C. Rigney, S. Willens, A. Rubens, and W. Simpson, "Remote authentication dial in user service (RADIUS)," Request for Comments 2865, Internet Engineering Task Force, June 2000.

[7] J. Boyle, R. Cohen, D. Durham, S. Herzog, R. Rajan, and A. Sastry, "The COPS (common open policy service) protocol," Request for Comments 2748, Internet Engineering Task Force, Jan. 2000.

[8] H. Schulzrinne and J. Rosenberg, "SIP caller preferences and callee capabilities," Internet Draft, Internet Engineering Task Force, July 2000. Work in progress.

[9] J. Rosenberg, D. Drew, and H. Schulzrinne, "Getting SIP through firewalls and NATs," Internet Draft, Internet Engineering Task Force, Feb. 2000. Work in progress.

[10] J. Rosenberg and H. Schulzrinne, "SIP traversal through enterprise and residential NATs and firewalls," Internet Draft, Internet Engineering Task Force, Nov. 2000. Work in progress.

[11] S. Kent and R. Atkinson, "IP encapsulating security payload (ESP)," Request for Comments 2406, Internet Engineering Task Force, Nov. 1998.

[12] T. Dierks and C. Allen, "The TLS protocol version 1.0," Request for Comments 2246, Internet Engineering Task Force, Jan. 1999.

[13] B. Ramsdell and Ed, "S/MIME version 3 message specification," Request for Comments 2633, Internet Engineering Task Force, June 1999.

[14] J. Rosenberg et al. , "An XML based format for watcher

information," Internet Draft, Internet Engineering Task Force, June 2000. Work in progress.

Table of Contents

Network Working Group D. Atkins, Telcordia Technologies
Internet Draft G. Klyne, Baltimore Technologies
13 June 2001
Expires: December 2001

Common Presence and Instant Messaging: Message Format
<draft-ietf-impp-cpim-msgfmt-03.txt>

Status of this memo

Copyright Notice

Abstract

This memo defines the mime type 'message/cpim', a message format for protocols that conform to the Common Profile for Instant Messaging (CPIM) specification.

Discussion of this document

Please send comments to: <impp@iastate.edu>. To subscribe: send a message with the body 'subscribe' to <impp-request@iastate.edu>. The mailing list archive is at <http://www.imppwg.org>.

Table of Contents

1. INTRODUCTION

This memo defines the mime content-type 'message/cpim. This is a common message format for CPIM-compliant messaging protocols [14].

While being prepared for CPIM, this format is quite general and may be reused by other applications with similar requirements. Application specifications that adopt this as a base format should answer the questions rasied in section 6 of this document.

1.1 Motivation

The Common Profile for Instant Messaging (CPIM) [14] specification defines a number of operations to be supported and criteria to be satisfied for interworking diverse instant messaging protocols. The intent is to allow a variety of different protocols interworking through gateways to support cross-protocol messaging that meets the requirements of RFC 2779 [15].

To adequately meet the security requirements of RFC 2779, a common message format is needed so that end-to-end signatures and encryption may be applied. This document describes a common canonical message format that must be used by any CPIM-compliant message transfer protocol, and over which signatures are calculated for end-to-end security.

1.2 Background

RFC 2779 requires that an instant message can carry a MIME payload [3,4]; thus some level of support for MIME will be a common element of any CPIM compliant protocol. Therefore it seems reasonable that a common message format should use a MIME/RFC822 syntax, as protocol implementations must already contain code to parse this.

Unfortunately, using pure RFC822/MIME [2] can be problematic:

- o Irregular lexical structure -- RFC822 allows a number of optional encodings and multiple ways to encode a particular value. For example RFC822 comments may be encoded in multiple ways. For security purposes, a single encoding method must be defined as a basis for computing message digest values. Protocols that transmit data in a different format would otherwise lose information needed to verify a signature.

- o Weak internationalization -- RFC822 requires header values to use 7-bit ASCII, which is problematic for encoding international character sets. Mechanisms for language tagging in RFC822 headers [16] are awkward to use and have limited applicability.

- o Mutability -- addition, modification or removal of header information. Because it is not explicitly forbidden, many applications that process MIME content (e.g. MIME gateways) rebuild or restructure messages in transit. This obliterates most attempt at achieving security (e.g. signatures), leaving receiving applications unable to verify the received data.

- o Message and payload separation -- there is not a clear syntactic distinction between message metadata and message content.

- o Limited extensibility (X-headers are problematic).

- o No support for structured information (text string values only).

- o Some processors impose line length limitations The message format defined by this memo overcomes some of these difficulties by having a syntax that is generally compatible with the format accepted by MIME/RFC822 parsers, but simplified, and having a stricter syntax. It also defines mechanisms to support some desired features not covered by the RFC822/MIME format specifications.

1.3 Goals

This specification aims to satisfy the following goals:

- o a securable end-to-end format for a message (a canonical message format for signature calculation)

- o independent of any specific application

- o capable of conveying a range of different address types

- o assumes an 8-bit clean message-transfer protocol

- o evolvable: extensible by multiple parties

- o to clearly separate message metadata from message content

- o a simple, regular, easily parsed syntax

- o a compact, low-overhead format for simple messages

1.4 Terminology and conventions

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 [1].

> NOTE: Comments like this provide additional nonessential information about the rationale behind this document. Such information is not needed for building a conformant implementation, but may help those who wish to understand the design in greater depth.

[[[Editorial comments and questions about outstanding issues are provided in triple brackets like this. These working comments should be resolved and removed prior to final publication.]]]

2. OVERALL MESSAGE STRUCTURE

The message/cpim format encapsulates an arbitrary MIME message content, together with message- and content-related metadata. This can optionally be signed or encrypted using MIME security multiparts in conjunction with an appropriate security scheme.

A message/cpim object is a multipart entity, where the first part contains the message metadata and the second part is the message content. The two parts are syntactically separated by a blank line, to keep the message header information (with its more stringent syntax rules) separate from the MIME message content headers.

Thus, the complete message looks something like this:

```
m: Content-type: message/cpim
s:
h: (message-metadata-headers)
s:
e: (encapsulated MIME message-body)
```

The end of the message body is defined by the framing mechanism of the protocol used. The tags 'm:', 's:', 'h:', 'e:', and 'x:' are not part of the message format and are used here to indicate the different parts of the message, thus:

```
m:  MIME headers for the overall message
s:  a blank separator line
h:  message headers
e:  encapsulated MIME object containing the message content
x:  MIME security multipart message wrapper
```

2.1 Message/cpim MIME headers

The message MIME headers identify the message as a CPIM-formatted message. The only required header is:

```
Content-type: message/cpim
```

Other MIME headers may be used as appropriate for the message transfer environment.

2.2 Message headers

Message headers carry information relevant to the end-to-end transfer of the message from sender to receiver. Message headers MUST NOT be modified, reformatted or reordered in transit, but in some circumstances they MAY be examined by a CPIM message transfer protocol.

The message headers serve a similar purpose to RFC822 message headers in email [2], and have a similar but restricted allowable syntax.

The basic header syntax is:

```
Key: Value
```

where "Key" is a header name and "Value" is the corresponding header value. The following considerations apply:

- o The entire header MUST be contained on a single line. The line terminator is not considered part of the header value.

- o Only one header per line. Multiple headers MUST NOT be included on a single line.

- o Processors SHOULD NOT impose any line-length limitations.

- o There MUST NOT be any whitespace at the beginning or end of a line.

- o UTF-8 character encoding [21] MUST be used throughout.

- o The character sequence CR,LF (13,10) MUST be used to terminate each line.

- o The header name contains only US-ASCII characters (see later for the specific syntax)

- The header MUST NOT contain any control characters (0-31). If a header value needs to represent control characters then the escape mechanism described below MUST be used.

- There MUST be a single space character (32) following the header name and colon.

- Multiple headers using the same key (header name) are allowed. (Specific header semantics may dictate only one occurrence of any particular header.)

- Headers names MUST match exactly (i.e. "From:" and "from:" are different headers).

- If a header name is not recognized or not understood, the header should be ignored. But see also the "Requires:" header.

- Interpretation (e.g. equivalence) of header values is dependent on the particular header definition. Message processors MUST preserve exactly all octets of all headers (both name and value).

- Message processors MUST NOT change the order of message headers.

Examples:

```
To: Pooh Bear <im:pooh@100akerwood.com>
From: <im:piglet@100akerwood.com>
Date: 2001-02-02T10:48:54-05:00
```

2.3 Character escape mechanism

This mechanism MUST be used to code control characters in a header, having Unicode code points in the range U+0000 to U+001f or U+007f. (The escape mechanism is as used by the Java programming language.) Note that the escape mechanism is applied to a UCS-2 character, NOT to the octets of its UTF-8 coding. Mapping from/to UTF-8 coding is performed without regard for escape sequences or character coding. (The header syntax is defined so that octets corresponding to control characters other than CR and LF do not appear in the output.)

An arbitrary UCS-2 character is escaped using the form:

```
\uxxxx
```

where:

```
\    is U+005c (backslash)
u    is U+0075 (lower case letter U)
xxxx is a sequence of exactly four hexadecimal digits
     (0-9, a-f or A-F) or
     (U+0030-U+0039, U+0041-U+0046, or U+0061-0066)
```

The hexadecimal number 'xxxx' is the UCS code-point value of the escaped character.

Further, the following special sequences introduced by "\" are used:

```
\\   for \ (backslash, U+005c)
\"   for " (double quote, U+0022)
\'   for ' (single quote, U+0027)
\b   for backspace (U+0008)
\t   for tab (U+0009)
\n   for linefeed (U+000a)
\r   for carriage return (U+000d)
```

2.3.1 Escape mechanism usage

When generating messages conformant with this specification:

o The special sequences listed above MUST be used to encode any occurrence of the following characters that appear anywhere in a header: backslash (U+005c), backspace (U+0008), tab (U+0009), linefeed (U+000a) or carriage return (U+000d).

o The special sequence \' MUST be used for any occurrence of a single quote (U+0027) that appears within a string delimited by single quotes.

o The special sequence \" MUST be used for any occurrence of a double quote (U+0022) that appears within a string delimited by double quotes.

+ Quote characters that delimit a string value MUST NOT be escaped.

o The general escape sequence \uxxxx MUST be used for any other control character (U+0000 to U+0007, U+000b to U+000c, U+000e to U+001f or u+007f) that appears anywhere in a header.

o All other characters MUST NOT be represented using an escape sequence.

When processing a message based on this specification, the escape sequence usage described above MUST be recognized.

Further, any other occurrence of any escape sequence described above SHOULD be recognized and treated as an occurrence of the corresponding Unicode character.

Any backslash ('\') character SHOULD be interpreted as introducing an escape sequence. Any unrecognized escape sequence SHOULD be treated as an instance of the character following the backslash character. An isolated backslash that is the last character of a header SHOULD be ignored.

2.4 Message content

The final section of a message/cpim is the MIME-encapsulated message content, which follows standard MIME formatting rules [3,4].

The MIME content headers MUST include at least a Content-Type header. The content may be any MIME type.

Example:

```
e: Content-Type: text/plain; charset=utf-8
e: Content-ID: <1234567890@foo.com>
e:
e: This is my encapsulated text message content
```

3. MESSAGE HEADER SYNTAX

A header is made of two parts, a name and a value, separated by a colon character (':') followed by a single space (32), and terminated by a sequence of CR,LF (13,10).

Headers use UTF-8 character encoding thoughout, per RFC 2279 [21].

3.1 Header names

The header name is a sequence of US-ASCII characters, excluding control characters, SPACE or separator characters. Use of the character "." in a header name is reserved for a namespace prefix separator.

Separator characters are:

```
SEPARATORS   = "(" / ")" / "<" / ">" / "@"
             / "," / ";" / ":" / "
             / "/" / "[" / "]" / "?" / "="
             / "{" / "}" / SP
```

NOTE: the range of allowed characters was determined by examination of HTTP and RFC822 header name formats and choosing the more resticted. The intent is to allow CPIM headers to follow a syntax that is compatible with the allowed syntax for both RFC 822 [2] and HTTP [18] (including HTTP-derived protocols such as SIP).

3.2 Header Value

A header value has a structure defined by the corresponding header specification. Implementations that use a particular header must adhere to the format and usage rules thus defined when creating or processing a message containing that header.

The other general constraints on header formats MUST also be followed (one line, UTF-8 character encoding, no control characters, etc.)

3.3 Language Tagging

Full internationalization of a protocol requires that a language can be indicated for any human-readable text [6,19].

A message header may indicate a language for its value by including ';lang=tag' after the header name and colon, where 'tag' is a language identifying token per RFC 3066 [7].

Example:

```
Subject:;lang=fr Objet de message
```

If the language parameter is not applied a header, any human-readable text is assumed to use the language identified as 'i-default' [19].

3.4 Namespaces for header name extensibility

NOTE: this section defines a framework for header extensibility whose use is optional. If no header extensions are allowed by an application then these structures may never be used.

An application that uses this message format is expected to define the set of headers that are required and allowed for that application. This section defines a header extensibility framework that can be used with any application.

The extensibility framework is based on that provided for XML [11] by XML namespaces [12]. All headers are associated with a "namespace", which is in turn associated with a globally unique URI.

Within a particular message instance, header names are associated with a particular namespace through the presence or absence of a namespace prefix, which is a leading part of the header name followed by a period ("."); e.g.

```
prefix.header-name: header-value
```

Here, 'prefix' is the header name prefix, 'header-name' is the header name within the namespace associated with 'prefix', and 'header-value' is the value for this header.

```
header-name: header-value
```

In this case, the header name prefix is absent, and the given 'header-name' is associated with a default namespace.

An application that uses this format designates a default namespace for any headers that are not more explicitly associated with any namespace. In many cases, the default namespace may be all that is needed.

A namespace is identified by a URI. In this usage, the URI is used simply as a globally unique identifier, and there is no requirement that it can be used for any other purpose. Any legal globally unique URI MAY be used to identify a namespace. (By "globally unique", we mean constructed according to some set of rules so that it is reasonable to expect that nobody else will use the same URI for a different purpose.) A URI used as an identifier MUST be a full absolute-URI, per RFC 2396 [10]. (Relative URIs and URI- references containing fragment identifiers MUST NOT be used for this purpose.)

Within a specific message, a 'NS' header is used to declare a namespace prefix and associate it with a URI that identifies a namespace. Following that declaration, within the scope of that message, the combination of namespace prefix and header name indicates a globally unique identifier for the header (consisting of the namespace URI and header name). For example:

```
NS: MyFeatures <mid:MessageFeatures@id.foo.com>
MyFeatures.WackyMessageOption: Use-silly-font
```

This defines a namespace prefix 'MyFeatures' associated with the namespace identifier 'mid:MessageFeatures@id.foo.com'. Subsequently the prefix indicates that the WackyMessageOption header name referenced is associated with the identified namespace.

A namespace prefix declaration MUST precede any use of that prefix.

With the exception of any application-specific predefined namespace prefixes (see section 6), a namespace prefix is strictly local to the message in which it occurs. The actual prefix used has no global significance. This means that the headers:

```
xxx.name: value
yyy.name: value
```

in two different messages may have exactly the same effect if namespace prefixes 'xxx' and 'yyy' are associated with the same namespace URI. Thus the following have exactly the same meaning:

```
NS: acme <http://id.acme.widgets/wily-headers/>
acme.runner-trap: set
```

and

```
NS: widget <http://id.acme.widgets/wily-headers/>
widget.runner-trap: set
```

A 'NS' header without a header prefix name specifies a default namespace for subsequent headers; that is a namespace that is associated with header names not having a prefix. For example:

```
NS: <http://id.acme.widgets/wily-headers/>
runner-trap: set
```

has the same meaning as the previous examples.

This framework allows different implementers to create extension headers without the worry of header name duplication; each defines headers within their own namespace.

3.5 Mandatory-to-recognize features

Sometimes it is necessary for the sender of a message to insist that some functionality is understood by the recipient. By using the mandatory-to-recognize indicator, a sender is notifying the recipient that it MUST understand the named header or feature in order to properly understand the message.

A header or feature is indicated as being mandatory-to-recognize by a 'Require:' header. For example:

```
Require: MyFeatures.VitalMessageOption
MyFeatures.VitalMessageOption: Confirmation-requested
```

Multiple required header names may be listed in a single 'Require' header, separated by commas.

> NOTE: indiscriminate use of 'Require:' headers could harm interoperability. It is suggested that any implementer who defines required headers also publish the header specifications so other implementations can succesfully interoperate.

The 'Require:' header MAY also be used to indicate that some non-header semantics must be implemented by the recipient, even when it does not appear as a header. For example:

```
Require: Locale.MustRenderKanji
```

might be used to indicate that message content includes characters from the Kanji repertoire, which must be rendered for proper understanding of the message. In this case, the header name is just a token (using header name syntax and namespace association) that indicates some desired behaviour.

3.6 Collected message header syntax

The following description of message header syntax uses ABNF, per RFC 2234 [17]. Most of this syntax can be interpreted as defining UCS character sequences or UTF-8 octet sequences. Alternate productions at the end allow for either interpretation.

```
      Header       = Header-name ":" *( ";" Parameter ) SP
                       Header-value
                       CRLF

      Header-name  = [ Name-prefix "." ] Name
      Name-prefix  = Name

      Parameter    = Lang-param / Ext-param
      Lang-param   = "lang=" Language-tag
      Ext-param    = Param-name "=" Param-value
      Param-name   = Name
      Param-value  = Token / Number / String

      Header-value = *HEADERCHAR

      Name         = 1*NAMECHAR
      Token        = 1*TOKENCHAR
      Number       = 1*DIGIT
      String       = DQUOTE *( Str-char / Escape ) DQUOTE
      Str-char     = %x20-21 / %x23-5B / %x5D-7E / UCS-high
      Escape       = "\" ( "u" 4(HEXDIG)    ; UCS codepoint
                         / "b"              ; Backspace
                         / "t"              ; Tab
                         / "n"              ; Linefeed
                         / "r"              ; Return
                         / DQUOTE           ; Double quote
                         / "'"              ; Single quote
                         / "\" )            ; Backslash

      Formal-name  = 1*( Token SP ) / String
      URI          = <defined as absolute-URI by RFC 2396>
      Language-tag = <defined by RFC 3066>

                   ; Any UCS character except CTLs, or escape
      HEADERCHAR   = UCS-no-CTL / Escape

                   ; Any US-ASCII char except ".", CTLs or SEPARATORS:
      NAMECHAR     = %21 / %23-26 / %2a-2b / %2d / %5e-60 / %7c / %7e
                   / ALPHA / DIGIT

                   ; Any UCS char except CTLs or SEPARATORS:
      TOKENCHAR    = NAMECHAR / "." / UCS-high
```

```
SEPARATORS   = "(" / ")" / "<" / ">" / "@"    ; 28/29/3c/3e/40
             / "," / ";" / ":" / "\" / <">    ; 2c/3b/3a/5c/22
             / "/" / "[" / "]" / "?" / "="    ; 2f/5b/5d/3f/3d
             / "{" / "}" / SP                 ; 7b/7d/20
CTL          = <Defined by RFC 2234 -- %x0-%x1f, %x7f>
CRLF         = <Defined by RFC 2234 -- CR, LF>
SP           = <defined by RFC 2234 -- %x20>
DIGIT        = <defined by RFC 2234 -- '0'-'9'>
HEXDIG       = <defined by RFC 2234 -- '0'-'9', 'A'-'F', 'a'-'f'>
ALPHA        = <defined by RFC 2234 -- 'A'-'Z', 'a'-'z'>
DQUOTE       = <defined by RFC 2234 -- %x22>
```

To interpret the syntax in a general UCS character environment, use the following productions:

```
UCS-no-CTL   = %x20-7e / UCS-high
UCS-high     = %x80-ffffffff
```

To interpret the syntax as defining UTF-8 coded octet sequences, use the following productions:

```
UCS-no-CTL   = UTF8-no-CTL
UCS-high     = UTF8-multi
UTF8-no-CTL  = %x20-7e / UTF8-multi
UTF8-multi   = %xC0-DF %x80-BF
             / %xE0-EF %x80-BF %x80-BF
             / %xF0-F7 %x80-BF %x80-BF %x80-BF
             / %xF8-FB %x80-BF %x80-BF %x80-BF %x80-BF
             / %xFC-FD %x80-BF %x80-BF %x80-BF %x80-BF %x80-BF
```

4. HEADER DEFINITIONS

This specification defines a core set of headers that are defined and available for use by applications: the application specification must indicate the headers that may be used, those that must be recognized and those that must appear in any message (see section 6).

The header definitions that follow fall into two categories:

(a) those that are part of the CPIM format extensibility framework, and

(b) some that have been based on similar headers in RFC 822, specified here with corresponding semantics.

Header names and syntax are given without a namespace qualification, and the associated namespace URI is listed as part of the header description. Any of the namespace associations already mentioned (implied default namespace, explicit default namespace or implied namespace prefix or explicit namespace prefix declaration) may be used to identify the namespace.

All headers defined here are associated with the namespace URI <[[[urn:iana:cpim-headers]]]>, which is defined according to [22].

4.1 The 'From' header

Indicates the sender of a message.

Header name: From

Namespace URI: <[[[urn:iana:cpim-headers]]]>

Syntax: (see also section 3.6)

```
From-header = "From" ": " [ Formal-name ] "<" URI ">"
```

Description:

Indicates the sender or originator of a message.

If present, the 'Formal-name' identifies the person or "real world" name for the originator.

The URI indicates an address for the originator.

Examples:

```
From: Winnie the Pooh <im:pooh@100akerwood.com>

From: <im:tigger@100akerwood.com>
```

4.2 The 'To' header

Specifies an intended recipient of a message.

Header name: To

Namespace URI: <[[[urn:iana:cpim-headers]]]>

Syntax: (see also section 3.6)

```
To-header = "To" ": " [ Formal-name ] "<" URI ">"
```

Description:

Indicates the recipient of a message.

If present, the 'Formal-name' identifies the person or "real world" name for the recipient.

The URI indicates an address for the recipient.

Multiple recipients may be indicated by including multiple 'To' headers.

Examples:

```
To: Winnie the Pooh <im:pooh@100akerwood.com>

To: <im:tigger@100akerwood.com>
```

4.3 The 'cc' header

Specifies a non-primary recipient ("courtesy copy") for a message.

Header name: cc

Namespace URI: <[[[urn:iana:cpim-headers]]]>

Syntax: (see also section 3.6)

```
Cc-header   = "cc" ": " [ Formal-name ] "<" URI ">"
```

Description:

Indicates a courtesy copy recipient of a message.

If present, the 'Formal-name', if present, identifies the person or "real world" name for the recipient.

The URI indicates an address for the recipient.

Multiple courtesy copy recipients may be indicated by including multiple 'cc' headers.

Examples:

cc: Winnie the Pooh <im:pooh@100akerwood.com>

cc: <im:tigger@100akerwood.com>

4.4 The 'DateTime' header

Specifies the date and time a message was sent.

Header name: Date

Namespace URI: <[[[urn:iana:cpim-headers]]]>

Syntax:

DateTime-header = "DateTime" ": " date-time

(where the syntax of 'date-time' is a profile of ISO8601, defined in "Date and Time on the Internet" [23])

Description:

The 'Date' header supplies the current date and time at which the sender sent the message.

One purpose of the this header is to provide for protection against a replay attack, by allowing the recipient to know when the message was intended to be sent. The value of the date header is the current time at the sender when the message was transmitted, using ISO 8601 date and time format as profiles in "Date and Time on the Internet: Timestamps" [23].

Example:

Date: 2001-02-01T12:16:49-05:00

4.5 The 'Subject' header

Contains a description of the topic of the message.

Header name: Subject

Namespace URI: <[[[urn:iana:cpim-headers]]]>

Syntax: (see also section 3.6)

```
Subject-header = "Subject" ":" [ lang-param ] SP *HEADERCHAR
```

Description:

The 'Subject' header supplies the sender's description of the topic or content of the message.

The sending agent should specify the language parameter if it has any reasonable knowledge of the language used by the sender to describe the message.

Example:

```
Subject:;lang=en Eeyore's feeling very depressed today
```

4.6 The 'NS' header

The "NS" header is used to declare a local namespace prefix.

Header name: NS

Namespace URI: <[[[urn:iana:cpim-headers]]]>

Syntax: (see also section 3.6)

```
NS-header = "NS" ": " [ Name-prefix ] "<" URI ">"
```

Description:

Declares a namespace prefix that may be used in subsequent header names. See section 3.4 for more details.

Example:

```
NS: MyAlias <mid:MessageFeatures@id.foo.com>
MyAlias.MyHeader: private-extension-data
```

4.7 The 'Require' header

Specify a header or feature that must be implemented by the receiver for correct message processing.

Header name: NS

Namespace URI: <[[[urn:iana:cpim-headers]]]>

Syntax: (see also section 3.6)

```
Require-header = "Require" ": " Header-name *( "," Header-name )
```

Description:

Declares a namespace prefix that may be used in subsequent header names. See section 3.5 for more details.

Note that there is no requirement that the required header actually be used, but for brevity it is recommended that an implemention not use issue require header for unused headers.

Example:

```
Require: MyAlias.VitalHeader
```

5. EXAMPLES

The examples in the following sections use the following per-line tags to indicate different parts of the overall message format:

- m: MIME headers for the overall message
- s: a blank separator line
- h: message headers
- e: encapsulated MIME object containing the message content
- x: MIME security multipart message wrapper

The following examples also assume that <[[[urn:iana:cpim-headers]]]> is the implied default namespace for the application concerned.

5.1 An example message/cpim message

The following example shows a message/cpim message:

```
m: Content-type: message/cpim
s:
h: From: MR SANDERS <im:piglet@100akerwood.com>
h: To: Depressed Donkey <im:eeyore@100akerwood.com>
h: Date: 2000-12-13T13:40:00-08:00
h: Subject: the weather will be fine today
h: Subject:;lang=fr beau temps prevu pour aujourd'hui
h: NS: MyFeatures <mid:MessageFeatures@id.foo.com>
h: Require: MyFeatures.VitalMessageOption
h: MyFeatures.VitalMessageOption: Confirmation-requested
h: MyFeatures.WackyMessageOption: Use-silly-font
s:
e: Content-type: text/xml; charset=utf-8
e: Content-ID: <1234567890@foo.com>
e:
e: <body>
e: Here is the text of my message.
e: </body>
```

5.2 An example using MIME multipart/signed

In order to secure a message/cpim, an application or implementation should use RFC 1847 and some appropriate cryptographic scheme.

Using S/MIME and pkcs7, the above message would look like this:

```
x: Content-Type: multipart/signed; boundary=next;
                 MDALG=SHA-1; type=application/pkcs
x:
x: --next
m: Content-Type: message/cpim
s:
h: From: MR SANDERS <im:piglet@100akerwood.com>
h: To: Dopey Donkey <im:eeyore@100akerwood.com>
h: Date: 2000-12-13T13:40:00-08:00
h: Subject: the weather will be fine today
h: Subject:;lang=fr beau temps prevu pour aujourd'hui
h: NS: MyFeatures <mid:MessageFeatures@id.foo.com>
h: Require: MyFeatures.VitalMessageOption
h: MyFeatures.VitalMessageOption: Confirmation-requested
h: MyFeatures.WackyMessageOption: Use-silly-font
s:
```

```
e: Content-type: text/xml; charset=utf-8
e: Content-ID: <1234567890@foo.com>
e:
e: <body>
e: Here is the text of my message.
e: </body>
x: --next
x: Content-Type: application/pkcs7
x:
x: (signature stuff)
    :
x: --next--
```

6. APPLICATION DESIGN CONSIDERATIONS

Applications using this specification must specify:

- o a default namespace URI for messages created and processed by that application
- o any namespace prefixes that are implicitly defined for messages created and processed by that application
- o all headers that must be recognized by implementations of the application
- o any headers that must be present in messages created by that application.
- o any headers that may appear more than once in a message, and how they are to be interpreted (e.g. how to interpret multiple 'subject:' headers with different language parameter values).

Within a network of message transfer agents, an intermediate gateway MUST NOT change the message/cpim content in any way. This implies that headers cannot be changed or reordered, transfer encoding cannot be changed, languages cannot be changed, etc.

Because message/cpim messages are immutable, any transfer agent that wants to modify the message should create a new message/cpim message with the modified header and containing the original message as its content. (This approach is similar to real-world bill-of-lading handling, where each person in the chain attaches a new sheet to the message. Then anyone can validate the original message and see what was changed and who changed it by following the trail of amendments. Another metaphor is including the old message in a new envelope.)

7. IANA CONSIDERATIONS

[[[Registration template for message/cpim content type]]]

[[[Registration of namespace URN for CPIM headers]]]

8. INTERNATIONALIZATION CONSIDERATIONS

Message headers use UTF-8 character encoding throughout, so can convey the full UCS-4 (Unicode, ISO/IEC 10646) character repertoire.

Language tagging is provided for message headers using the "Language" parameter.

Message content is any MIME-encapsulated content, and normal MIME content internationalization considerations apply.

9. SECURITY CONSIDERATIONS

The message/cpim format is designed with security in mind. In particular it is designed to be used with MIME security multiparts for signatures and encryption. To this end, message/cpim messages must be considered immutable once created.

Because message/cpim messages are binary messages (due to UTF-8 encoding), if they are transmitted across non-8-bit-clean transports then the transfer agent must tunnel the entire message. Changing the message data encoding is not an allowable option. This implies that the message/cpim must be encapsulated by the message tranfer system and unencapsulated at the receiving end of the tunnel.

The resulting message must have no data loss due to the encoding and unencoding of the message. For example, an application may choose to apply the MIME base64 content-transfer-encoding to the message/cpim object to meet this requirement.

10. ACKNOWLEDGEMENTS

The authors thank the following for their helpful comments: Harald Alvestrand, Walter Houser, Leslie Daigle, [[[....]]]

11. REFERENCES

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

[2] Crocker, D., "Standard for the format of ARPA Internet text messages", RFC 822, STD 11, August 1982.

[3] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, November 1996.

[4] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046 November 1996.

[5] Freed, N., Klensin, J., and J. Postel, "Multipurpose Internet Mail Extensions (MIME) Part Four: Registration Procedures", RFC 2048, BCP 13, November 1996.

[6] Weider, C., Preston, C., Simonsen, K., Alvestrand, H., Atkinson, R., Crispin, M., Svanberg, P., "Report from the IAB Character Set Workshop", RFC 2130, April 1997.

[7] Alvestrand, H., "Tags for the Identification of Languages", RFC 3066, January 2001. (Defines Content-language header.)

[8] Ramsdell, B., "S/MIME Version 3 Message Specification", RFC 2633, June 1999.

[9] Callas, J., Donnerhacke, L., Finney, H. and R. Thayer, "OpenPGP Message Format", RFC 2440, November 1998.

[10] Berners-Lee, T., Fielding, R.T. and L. Masinter, "Uniform Resource Identifiers (URI): Generic Syntax", RFC 2396, August 1998.

[11] Tim Bray, Jean Paoli, and C. M. Sperberg-McQueen, "Extensible Markup Language (XML) 1.0", W3C recommendation: <http://www.w3.org/TR/REC-xml>, 10 February 1998.

[12] Tim Bray, Dave Hollander, and Andrew Layman "Namespaces in XML", W3C recommendation: <http://www.w3.org/TR/REC-xml-names>, 14 January 1999.

[13] "Data elements and interchange formats - Information interchange - Representation of dates and times" ISO 8601:1988(E) International Organization for Standardization June 1988.

[14] Crocker, D.H., Diacakis, A., Mazzoldi, F., Huitema, C., Klyne, G., Rose, M.T., Rosenberg, J., Sparks, R. and H. Sugano, "A Common Profile for Instant Messaging (CPIM)", draft-thenine-im-common-00 (work in progress), August 2000.

[15] Day, M., Aggarwal, S., Mohr, G., and J. Vincent "Instant Messaging / Presence Protocol Requirements" RFC 2779 February 2000.

[16] N. Freed, K. Moore "MIME Parameter Value and Encoded Word Extensions: Character Sets, Languages, and Continuations" RFC 2231 November 1997.

[17] D. Crocker, P. Overell "Augmented BNF for Syntax Specifications: ABNF" RFC 2234 November 1997.

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

[19] Alvestrand, H, "IETF Policy on Character Sets and Languages", RFC 2277, BCP 18, January 1998.

[20] Freed, N., and J. Postel, "IANA Charset Registration Procedures", BCP 19, RFC 2278, January 1998.

[21] F. Yergeau "UTF-8, a transformation format of ISO 10646" RFC 2279 January 1998.

[22] M. Mealling "A URN Namespace for IANA Registered Protocol Elements" draft-mealling-iana-urn-00.txt (work in progress) November 2000

[23] C. Newman, G. Klyne "Date and Time on the Internet: Timestamps" draft-ietf-impp-datetime-03.txt (work in progress) May 2001.

12. AUTHORS' ADDRESSES

Derek Atkins
Telcordia Technologies
6 Farragut Ave
Somerville, MA 02144
USA.
Telephone: +1 617 623 3745
E-mail: warlord@research.telcordia.com
E-mail: warlord@alum.mit.edu

Graham Klyne
Baltimore Technologies - Content Security Group,
1310 Waterside,
Arlington Business Park
Theale
Reading, RG7 4SA
United Kingdom.
Telephone: +44 118 903 8000
Facsimile: +44 118 903 9000
E-mail: GK@ACM.ORG

Appendix A: Amendment history

00a 01-Feb-2001 Memo initially created.

00b 06-Feb-2001 Editorial review. Reworked namespace framework description. Deferred specification of mandatory headers to the application specification, allowing this document to be less application-dependent. Expanded references. Replaced some text with ABNF syntax descriptions. Reordered some major sections.

00c 07-Feb-2001 Folded in some review comments. Fix up some syntax problems. Other small editorial changes. Add some references.

01a 29-Mar-2001 Incorporate review comments. State (simply) that this is a canonical end-to-end format for the purpose of signature calculation. Defined escape mechanism for control characters. Header name parameters placed after the ":". Changed name of Date: header to DateTime:. Revised syntax to separate character-level syntax from UTF-8 octet- level syntax.

01b 30-Mar-2001 State explicitly that unrecognized header names should be ignored. Remove text about (non)significance of header order: simply say that order must be preserved.

02a 30-May-2001 Updated reference to date/time draft. Editorial changes.

03a 13-Jun-2001 Tighten up application of escape sequences.

TODO:

- o confirm urn namespace for headers (currently depends on a work-in-progress).

- o Complete IANA considerations

REVIEW CHECKLIST:

(Points to be checked or considered more widely on or before final review.)

- o The desirability of a completely rigid syntax.

- o Escape mechanism details.

Full copyright statement

Network Working Group G. Klyne, Baltimore Technologies
Internet Draft C. Newman, Sun Microsystems
3 July 2001
Expires: January 2002

Date and Time on the Internet: Timestamps

<draft-ietf-impp-datetime-04.txt>

Status of this memo

Copyright Notice

Abstract

This document defines a date and time format for use in Internet protocols that is a profile of the ISO 8601 [ISO8601] standard for representation of dates and times using the Gregorian calendar.

Table of Contents

1. Introduction

Date and time formats cause a lot of confusion and interoperability problems on the Internet. This document addresses many of the problems encountered and makes recommendations to improve consistency and interoperability when representing and using date and time in Internet protocols.

This document includes an Internet profile of the ISO 8601 [ISO8601] standard for representation of dates and times using the Gregorian calendar.

There are many ways in which date and time values might appear in Internet protocols: this document focuses on just one common usage, viz. timestamps for Internet protocol events. This limited consideration has the following consequences:

- All dates and times are assumed to be in the "current era", somewhere between 0000AD and 9999AD.

- All times expressed have a stated relationship (offset) to Coordinated Universal Time (UTC). (This is distinct from some usage in scheduling applications where a local time and location may be known, but the actual relationship to UTC may be dependent on the unknown or unknowable actions of politicians or administrators. The UTC time corresponding to 17:00 on 23rd March 2005 in New York may depend on administrative decisions about daylight savings time. This specification steers well clear of such considerations.)

- Timestamps can express times that occurred before the introduction of UTC. Such timestamps are expressed relative to universal time, using the best available practice at the stated time.

- Date and time expressions indicate an instant in time. Description of time periods, or intervals, is not covered here.

2. Definitions

UTC — Coordinated Universal Time as maintained by the Bureau International des Poids et Mesures (BIPM).

second — A basic unit of measurement of time in the International System of Units. It is defined as the duration of 9,192,631,770 cycles of microwave light absorbed or emitted by the hyperfine transition of cesium-133 atoms in their ground state undisturbed by external fields.

minute — A period of time of 60 seconds. However, see also the restrictions in section 5.7 and Appendix D for how leap seconds are denoted within minutes.

hour — A period of time of 60 minutes.

day — A period of time of 24 hours.

leap year In the Gregorian calendar, a year which has 366 days. A leap year is a year whose number is divisible by four an integral number of times, except that if it is a centennial year (i.e. divisible by one hundred) it shall also be divisible by four hundred an integral number of times.

ABNF Augmented Backus-Naur Form, a format used to represent permissible strings in a protocol or language, as defined in [ABNF].

Email Date/Time Format
The date/time format used by Internet Mail as defined by RFC 2822 [IMAIL-UPDATE].

Internet Date/Time Format
The date format defined in section 5 of this document.

For more information about time scales, see Appendix E of [NTP], Section 3 of [ISO8601], and the appropriate ITU documents [ITU-R-TF].

3. Two Digit Years

The following requirements are to address the problems of ambiguity of 2-digit years:

- Internet Protocols MUST generate four digit years in dates.

- The use of 2-digit years is deprecated. If a 2-digit year is received, it should be accepted ONLY if an incorrect interpretation will not cause a protocol or processing failure (e.g. if used only for logging or tracing purposes).

- It is possible that a program using two digit years will represent years after 1999 as three digits. This occurs if the program simply subtracts 1900 from the year and doesn't check the number of digits. Programs wishing to robustly deal with dates generated by such broken software may add 1900 to three digit years.

- It is possible that a program using two digit years will represent years after 1999 as ":0", ":1", ... ":9", ";0", ... This occurs if the program simply subtracts 1900 from the year and adds the decade to the US-ASCII character zero. Programs wishing to robustly deal with dates generated by such broken software should detect non-numeric decades and interpret appropriately.

The problems with two digit years amply demonstrate why all dates and times used in Internet protocols MUST be fully qualified.

4. Local Time

4.1. Coordinated Universal Time (UTC)

Because the daylight saving rules for local time zones are so convoluted and can change based on local law at unpredictable times, true interoperability is best achieved by using Coordinated Universal Time (UTC). This specification does not cater to local time zone rules.

4.2. Local Offsets

The offset between local time and UTC is often useful information. For example, in electronic mail (RFC2822, [IMAIL-UPDATE]) the local offset provides a useful heuristic to determine the probability of a prompt response. Attempts to label local offsets with alphabetic strings have resulted in poor interoperability in the past [IMAIL], [HOST-REQ]. As a result, RFC2822 [IMAIL-UPDATE] has made numeric offsets mandatory.

Numeric offsets are calculated as "local time minus UTC". So the equivalent time in UTC can be determined by subtracting the offset from the local time. For example, 18:50:00-04:00 is the same time as 22:50:00Z.

NOTE: Following ISO 8601, numeric offsets represent only time zones that differ from UTC by an integral number of minutes. However, many historical time zones differ from UTC by a non-integral number of minutes. To represent such historical time stamps exactly, applications must convert them to a representable time zone.

4.3. Unknown Local Offset Convention

If the time in UTC is known, but the offset to local time is unknown, this can be represented with an offset of "-00:00". This differs semantically from an offset of "Z" or "+00:00", which imply that UTC is the preferred reference point for the specified time. RFC2822 [IMAIL-UPDATE] describes a similar convention for email.

4.4. Unqualified Local Time

A number of devices currently connected to the Internet run their internal clocks in local time and are unaware of UTC. While the Internet does have a tradition of accepting reality when creating specifications, this should not be done at the expense of interoperability. Since interpretation of an unqualified local time zone will fail in approximately 23/24 of the globe, the interoperability problems of unqualified local time are deemed unacceptable for the Internet. Systems that are configured with a local time, are unaware of the corresponding UTC offset, and depend on time synchronization with other Internet systems, MUST use a mechanism that ensures correct synchronization with UTC. Some suitable mechanisms are:

- o Use Network Time Protocol [NTP] to obtain the time in UTC.

- o Use another host in the same local time zone as a gateway to the Internet. This host MUST correct unqualified local times they are transmitted to other hosts.

- o Prompt the user for the local time zone and daylight saving rule settings.

5. Date and Time format

This section discusses desirable qualities of date and time formats and defines a profile of ISO 8601 for use in Internet protocols.

5.1. Ordering

If date and time components are ordered from least precise to most precise, then a useful property is achieved. Assuming that the time zones of the dates and times are the same (e.g. all in UTC), expressed using the same string (e.g. all "Z" or all "+00:00"), and all times have the same number of fractional second digits, then the date and time strings may be sorted as strings (e.g. using the strcmp() function in C) and a time-ordered sequence will result. The presence of optional punctuation would violate this characteristic.

5.2. Human Readability

Human readability has proved to be a valuable feature of Internet protocols. Human readable protocols greatly reduce the costs of debugging since telnet often suffices as a test client and network analyzers need not be modified with knowledge of the protocol. On the other hand, human readability sometimes results in interoperability problems. For example, the date format "10/11/1996" is completely unsuitable for global interchange because it is interpreted differently in different countries. In addition, the date format in [IMAIL] has resulted in interoperability problems when people assumed any text string was permitted and translated the three letter abbreviations to other languages or substituted date formats which were easier to generate (e.g. the format used by the C function ctime). For this reason, a balance must be struck between human readability and interoperability.

Because no date and time format is readable according to the conventions of all countries, Internet clients SHOULD be prepared to transform dates into a display format suitable for the locality. This may include translating UTC to local time.

5.3. Rarely Used Options

A format which includes rarely used options is likely to cause interoperability problems. This is because rarely used options are less likely to be used in alpha or beta testing, so bugs in parsing are less likely to be discovered. Rarely used options should be made mandatory or omitted for the sake of interoperability whenever possible.

The format defined below includes only one rarely used option: fractions of a second. It is expected that this will be used only by applications which require strict ordering of date/time stamps or which have an unusual precision requirement.

5.4. Redundant Information

If a date/time format includes redundant information, that introduces the possibility that the redundant information will not correlate. For example, including the day of the week in a date/time format introduces the possibility that the day of week is incorrect but the date is correct, or vice versa. Since it is not difficult to compute the day of week from a date (see Appendix B), the day of week should not be included in a date/time format.

5.5. Simplicity

The complete set of date and time formats specified in ISO 8601 [ISO8601] is quite complex in an attempt to provide multiple representations and partial representations. Appendix A contains an attempt to translate the complete syntax of ISO 8601 into ABNF. Internet protocols have somewhat different requirements and simplicity has proved to be an important characteristic. In addition, Internet protocols usually need complete specification of data in order to achieve true interoperability. Therefore, the complete grammar for ISO 8601 is deemed too complex for most Internet protocols.

The following section defines a profile of ISO 8601 for use on the Internet. It is a conformant subset of the ISO 8601 extended format. Simplicity is achieved by making most fields and punctuation mandatory.

5.6. Internet Date/Time Format

The following profile of ISO 8601 [ISO8601] dates SHOULD be used in new protocols on the Internet. This is specified using the syntax description notation defined in [ABNF].

```
date-fullyear   = 4DIGIT
date-month      = 2DIGIT  ; 01-12
date-mday       = 2DIGIT  ; 01-28, 01-29, 01-30, 01-31 based on month/year
time-hour       = 2DIGIT  ; 00-23
time-minute     = 2DIGIT  ; 00-59
time-second     = 2DIGIT  ; 00-58, 00-59, 00-60 based on leap second rules
time-secfrac    = "." 1*DIGIT
time-numoffset  = ("+" / "-") time-hour ":" time-minute
time-offset     = "Z" / time-numoffset

partial-time    = time-hour ":" time-minute ":" time-second
                  [time-secfrac]
full-date       = date-fullyear "-" date-month "-" date-mday
full-time       = partial-time time-offset

date-time       = full-date "T" full-time
```

NOTE: Per [ABNF] and ISO8601, the "T" and "Z" characters in this syntax may alternatively be lower case "t" or "z" respectively.

NOTE: ISO 8601 defines date and time separated by "T". Applications using this syntax may choose, for the sake of readability, to specify a full-date and full-time separated by (say) a space character.

5.7. Restrictions

The grammar element date-mday represents the day number within the current month. The maximum value varies based on the month and year as follows:

Month Number	Month/Year	Maximum value of date-mday
01	January	31
02	February, normal	28
02	February, leap year	29
03	March	31
04	April	30
05	May	31
06	June	30
07	July	31
08	August	31
09	September	30
10	October	31
11	November	30
12	December	31

Appendix C contains sample C code to determine if a year is a leap year.

The grammar element time-second may have the value "60" at the end of months in which a leap second occurs -- to date: June (XXXX-06-30T23:59:60Z) or December (XXXX-12-31T23:59:60Z); see Appendix D for a table of leap seconds. It is also possible for a leap second to be subtracted, at which times the maximum value of time-second is "58". At all other times the maximum value of time-second is "59". Further, in time zones other than "Z", the leap second point is shifted by the zone offset (so it happens at the same instant around the globe).

Leap seconds cannot be predicted far into the future. The International Earth Rotation Service publishes bulletins [IERS] that announce leap seconds with a few weeks' warning. Applications should not generate timestamps involving inserted leap seconds until after the leap seconds are announced.

Although ISO 8601 permits the hour to be "24", this profile of ISO 8601 only allows values between "00" and "23" for the hour in order to reduce confusion.

5.8. Examples

Here are some examples of Internet date/time format.

```
1985-04-12T23:20:50.52Z
```

This represents 20 minutes and 50.52 seconds after the 23rd hour of April 12th, 1985 in UTC.

```
1996-12-19T16:39:57-08:00
```

This represents 39 minutes and 57 seconds after the 16th hour of December 19th, 1996 with an offset of -08:00 from UTC (Pacific Standard Time). Note that this is equivalent to 1996-12-20T00:39:57Z in UTC.

```
1990-12-31T23:59:60Z
```

This represents the leap second inserted at the end of 1990.

```
1990-12-31T15:59:60-08:00
```

This represents the same leap second in Pacific Standard Time, 8 hours behind UTC.

```
1937-01-01T12:00:27.87+00:20
```

This represents the same instant of time as noon, January 1, 1937, Netherlands time. Standard time in the Netherlands was exactly 19 minutes and 32.13 seconds ahead of UTC by law from 1909-05-01 through 1937-06-30. This time zone cannot be represented exactly using the HH:MM format, and this timestamp uses the closest representable UTC offset.

6. Acknowledgements

The following people provided helpful advice for an earlier incarnation of this document: Ned Freed, Neal McBurnett, David Keegel, Markus Kuhn, Paul Eggert and Robert Elz. Thanks are also due to participants of the IETF Calendaring/Scheduling working group mailing list, and participants of the time zone mailing list.

The following reviewers contributed helpful suggestions for the present revision: Tom Harsch, Markus Kuhn, Pete Resnick, Dan Kohn. Paul Eggert provided many careful observations regarding the subtleties of leap seconds and time zone offsets.

7. References

[Zeller] Chr. Zeller, "Kalender-Formeln", Acta Mathematica, Vol. 9, Nov 1886.

[IMAIL] Crocker, D., "Standard for the Format of Arpa Internet Text Messages", RFC 822, August 1982.

[IMAIL-UPDATE] Resnick, P., "Internet Message Format", RFC 2822, April 2001.

[ABNF] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 2234, November 1997.

[ISO8601] "Data elements and interchange formats -- Information interchange -- Representation of dates and times", ISO 8601:1988(E), International Organization for Standardization, June, 1988.

[ISO8601:2000] "Data elements and interchange formats -- Information interchange -- Representation of dates and times", ISO 8601:2000, International Organization for Standardization, December, 2000.

[HOST-REQ] Braden, R., "Requirements for Internet Hosts -- Application and Support", RFC 1123, Internet Engineering Task Force, October 1989.

[IERS] International Earth Rotation Service Bulletins, <http://hpiers.obspm.fr/eop-pc/products/bulletins.html>.

[NTP] Mills, D., "Network Time Protocol (Version 3) Specification, Implementation and Analysis", RFC 1305, University of Delaware, March 1992.

[ITU-R-TF] International Telecommunication Union Recommendations for Time Signals and Frequency Standards Emissions. <http://www.itu.ch/publications/itu-r/iturtf.htm>

8. Security Considerations

Since the local time zone of a site may be useful for determining a time when systems are less likely to be monitored and might be more susceptible to a security probe, some sites may wish to emit times in

UTC only. Others might consider this to be loss of useful functionality at the hands of paranoia.

9. Authors' Addresses

Chris Newman
Sun Microsystems
1050 Lakes Drive, Suite 250
West Covina, CA 91790 USA

Email: cnewman@iplanet.com

Graham Klyne (editor, this revision)
Baltimore Technologies - Content Security Group
1310 Waterside
Arlington Business Park
Theale
Reading, RG7 4SA
United Kingdom.
Telephone: +44 118 903 8000
Facsimile: +44 118 903 9000
E-mail: GK@ACM.ORG

Appendix A. ISO 8601 Collected ABNF

This information is based on the 1988 version of ISO 8601. There may be some changes in the 2000 revision.

ISO 8601 does not specify a formal grammar for the date and time formats it defines. The following is an attempt to create a formal grammar from ISO 8601. This is informational only and may contain errors. ISO 8601 remains the authoritative reference.

Note that due to ambiguities in ISO 8601, some interpretations had to be made. First, ISO 8601 is not clear if mixtures of basic and extended format are permissible. This grammar permits mixtures. ISO 8601 is not clear on whether an hour of 24 is permissible only if minutes and seconds are 0. This assumes that an hour of 24 is permissible in any context. Restrictions on date-mday in section 5.7 apply. ISO 8601 states that the "T" may be omitted under some circumstances. This grammar requires the "T" to avoid ambiguity.

ISO 8601 also requires (in section 5.3.1.3) that a decimal fraction be proceeded by a "0" if less than unity. Annex B.2 of ISO 8601 gives examples where the decimal fractions are not preceded by a "0". This grammar assumes section 5.3.1.3 is correct and that Annex B.2 is in error.

```
   date-century    = 2DIGIT  ; 00-99
   date-decade     =  DIGIT  ; 0-9
   date-subdecade  =  DIGIT  ; 0-9
   date-year       = date-decade date-subdecade
   date-fullyear   = date-century date-year
   date-month      = 2DIGIT  ; 01-12
   date-wday       =  DIGIT  ; 1-7  ; 1 is Monday, 7 is Sunday
   date-mday       = 2DIGIT  ; 01-28, 01-29, 01-30, 01-31 based on month/year
   date-yday       = 3DIGIT  ; 001-365, 001-366 based on year
   date-week       = 2DIGIT  ; 01-52, 01-53 based on year

   datepart-fullyear = [date-century] date-year ["-"]
   datepart-ptyear   = "-" [date-subdecade ["-"]]
   datepart-wkyear   = datepart-ptyear / datepart-fullyear

   dateopt-century   = "-" / date-century
   dateopt-fullyear  = "-" / datepart-fullyear
   dateopt-year      = "-" / (date-year ["-"])
   dateopt-month     = "-" / (date-month ["-"])
   dateopt-week      = "-" / (date-week ["-"])

   datespec-full     = datepart-fullyear date-month ["-"] date-mday
   datespec-year     = date-century / dateopt-century date-year
   datespec-month    = "-" dateopt-year date-month [["-"] date-mday]
   datespec-mday     = "--" dateopt-month date-mday
   datespec-week     = datepart-wkyear "W"
                       (date-week / dateopt-week date-wday)
   datespec-wday     = "---" date-wday
   datespec-yday     = dateopt-fullyear date-yday

   date              = datespec-full / datespec-year / datespec-month /
       datespec-mday / datespec-week / datespec-wday / datespec-yday
```

Time:

```
time-hour         = 2DIGIT ; 00-24
time-minute       = 2DIGIT ; 00-59
time-second       = 2DIGIT ; 00-58, 00-59, 00-60 based on leap-second rules
time-fraction     = ("," / ".") 1*DIGIT
time-numoffset    = ("+" / "-") time-hour [[":"] time-minute]
time-zone         = "Z" / time-numoffset

timeopt-hour      = "-" / (time-hour [":"])
timeopt-minute    = "-" / (time-minute [":"])

timespec-hour     = time-hour [[":"] time-minute [[":"] time-second]]
timespec-minute   = timeopt-hour time-minute [[":"] time-second]
timespec-second   = "-" timeopt-minute time-second
timespec-base     = timespec-hour / timespec-minute / timespec-second

time              = timespec-base [time-fraction] [time-zone]

iso-date-time     = date "T" time
```

Durations:

```
dur-second        = 1*DIGIT "S"
dur-minute        = 1*DIGIT "M" [dur-second]
dur-hour          = 1*DIGIT "H" [dur-minute]
dur-time          = "T" (dur-hour / dur-minute / dur-second)
dur-day           = 1*DIGIT "D"
dur-week          = 1*DIGIT "W"
dur-month         = 1*DIGIT "M" [dur-day]
dur-year          = 1*DIGIT "Y" [dur-month]
dur-date          = (dur-day / dur-month / dur-year) [dur-time]

duration          = "P" (dur-date / dur-time / dur-week)
```

Periods:

```
period-explicit   = date-time "/" date-time
period-start      = date-time "/" duration
period-end        = duration "/" date-time

period            = period-explicit / period-start / period-end
```

Appendix B. Day of the Week

The following is a sample C subroutine loosely based on Zeller's Congruence [Zeller] which may be used to obtain the day of the week for dates on or after 0000-02-01:

```
char *day_of_week(int day, int month, int year)
{
    int cent;
    char *dayofweek[] = {
        "Sunday", "Monday", "Tuesday", "Wednesday",
        "Thursday", "Friday", "Saturday"
    };

    /* adjust months so February is the last one */
    month -= 2;
    if (month < 1) {
        month += 12;
        --year;
    }
    /* split by century */
    cent = year / 100;
    year %= 100;
    return (dayofweek[((26 * month - 2) / 10 + day + year
                       + year / 4 + cent / 4 - 2 * cent) % 7]);
}
```

Appendix C. Leap Years

Here is a sample C subroutine to calculate if a year is a leap year:

```
/* This returns non-zero if year is a leap year.  Must use 4 digit year.
 */
int leap_year(int year)
{
    return (year % 4 == 0 && (year % 100 != 0 || year % 400 == 0));
}
```

Appendix D. Leap Seconds

Information about leap seconds can be found at: <http://tycho.usno.navy.mil/leapsec.html>. In particular, it notes that:

> The decision to introduce a leap second in UTC is the responsibility of the International Earth Rotation Service (IERS). According to the CCIR Recommendation, first preference is given to the opportunities at the end of December and June, and second preference to those at the end of March and September.

When required, insertion of a leap second occurs as an extra second at the end of a day in UTC, represented by a timestamp of the form YYYY-MM-DDT23:59:60Z. A leap second occurs simultaneously in all time zones, so that time zone relationships are not affected. See section 5.8 for some examples of leap second times.

The following table is an excerpt from the table maintained by the United States Naval Observatory. The source data is located at:

<ftp://maia.usno.navy.mil/ser7/tai-utc.dat>

This table shows the date of the leap second, and the difference between the time standard TAI (which isn't adjusted by leap seconds) and UTC after that leap second.

UTC Date	TAI - UTC After Leap Second
1972-06-30	11
1972-12-31	12
1973-12-31	13
1974-12-31	14
1975-12-31	15
1976-12-31	16
1977-12-31	17
1978-12-31	18
1979-12-31	19
1981-06-30	20
1982-06-30	21
1983-06-30	22
1985-06-30	23
1987-12-31	24
1989-12-31	25
1990-12-31	26
1992-06-30	27
1993-06-30	28
1994-06-30	29
1995-12-31	30
1997-06-30	31
1998-12-31	32

Appendix E. Amendment history

00a 30-Mar-2001 This document version created from Chris Newman's original 'draft-ietf-impp-datetime-00.txt'. Material relating to future times (schedule events) and time zone names has been removed. Added introductory text setting the scope for this document. Various small editorial changes.

00b 03-Apr-2001 Added reference [ABNF], and updated citations. Added comment about possible use of space-separated date/time fields. Added comment about possible use of lower case "t" and "z" in syntax. Corrected leap-second examples and noted that leap second point is offset by time zone.

01a 06-Apr-2001 Updated author affiliation and contact details. Udated leap-second table.

01b 10-May-2001 Clarified provenance of (non-normative) information in appendix A.

02a 11-May-2001 Reference updated email specification (RFC2822).

02b 14-May-2001 Fix up some detailed information concerning leap seconds. Include text describing timestamps for times before introduction of UTC. Caution against the use of future timestamps using leap seconds. Correction to day-of-week sample code, and note restriction on applicability. Various editorial corrections.

03a 23-May-2001 Editorial fixes. Minor clarification of leap seconds.

03b 24-May-2001 More clarification of leap seconds and time zones.

03c 25-May-2001 More minor editorial fixes.

04a 03-Jul-2001 Fix off-by-one error in Netherlands example.