INTERNET-DRAFT

H. Sugano
Fujitsu

F. Mazzoldi
Network Projects, Inc.

A. Diacakis
Network Projects, Inc.

S. Fujimoto
Fujitsu

G. Hudson
MIT

J. D. Ramsdell
The MITRE Corporation

Expires: September 2001

March 2001

Presence and Instant Messaging Protocol (PRIM)
<draft-mazzoldi-prim-impp-01.txt>

Status of this Memo

Please send comments to the authors or to the prim@ml.fujitsulabs.com discussion list.

Copyright Notice

Abstract

The architecture and specifications of the Presence and Instant Messaging protocols (PRIM) are described. PRIM defines a set of protocols for the Presence and Instant Messaging services which satisfy the IMPP requirements [RFC2779]. PRIM is also designed so as to conform with the Common Profile for Instant Messaging (CPIM) specification being developed in the IMPP WG.

Table of Contents

1. Introduction

On the Internet and elsewhere, a growing number of people would like to know when others are available to communicate with them. A system that provides this type of PRESENCE INFORMATION is known as Presence Service.

INSTANT MESSAGING allows text base communication to occur in a rapid, conversational fashion. An INSTANT MESSAGE is delivered to a recipient if the recipient is listening for messages, otherwise the message is dropped and the sender is informed of the delivery failure.

PRESENCE and INSTANT MESSAGING SERVICES are separate and can work independently of each other. However, by utilizing the Presence Service a user has a better idea as to whether a recipient is listening for INSTANT MESSAGES. Therefore, the two services are often used in tandem.

The PResence and Instant Messaging (PRIM) protocol is designed so that INSTANT MESSAGING and PRESENCE SERVICES can be provided by a set of servers distributed across a large number of administrative domains.

PRIM is also designed to conform to the Common Profile for Instant Messaging (CPIM) specification being developed by the IMPP WG. This enables that users of PRIM services exchange PRESENCE INFORMATION and INSTANT MESSAGES with the users of the services which use other CPIM compatible protocols.

1.1. Design Goals and Assumptions

Some of the design principles on which this protocol is based are:

o Transfer protocol directly atop of TCP

PRIM assumes TCP as the basic transport mechanism for INSTANT MESSAGES and PRESENCE INFORMATION. TCP provides a sufficiently reliable transport infrastructure which is required by both INSTANT MESSAGING and PRESENCE SERVICES.

o Long-lived Client/Server connections

PRIM uses long-lived client/server TCP connections in order to receive INSTANT MESSAGES and PRESENCE INFORMATION NOTIFICATIONS. Note that this is the prevailing model used by most Presence and IM systems today. It brings the following advantages:

- Overhead is reduced, because authentication is performed once, at the beginning of the connection. This is important, for example, when PRESENCE INFORMATION NOTIFICATIONS occur frequently.

- Connections are firewall friendly, because USER AGENTS initiate connections from inside a firewall that can carry NOTIFICATIONS or messages initiated from the outside.

o Selective Presence Publication

[RFC2779] stipulates various requirements for access control; 2.3.x and several in section 5. Among others, we consider the feature of "Polite Blocking" (5.1.15, 5.2.3) to be very important for PRESENCE SERVICES. This protocol contains a mechanism for such selective PRESENCE INFORMATION publication as well as in-band access control.

2.   Terminology

[RFC2778] and [RFC2779] define the terminology for the PRESENCE and INSTANT MESSAGING fields. Please refer to those documents for a complete glossary of the UPPER CASED terms.

The key words "MUST", "MUST NOT", "REQUIRED", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [34].

3.   Architecture

The PRIM architecture involves two components: Service Domains and USER AGENTS. Each Service Domain is a set of servers that are responsible for a set of PRINCIPALS. A PRINCIPAL’s Service Domain is called its Home Domain. A PRINCIPAL connects to its Home Domain via an USER AGENT to access PRESENCE and INSTANT MESSAGING SERVICES.

In particular, a Service Domain is composed by Presence and/or Instant Messaging Servers.

```
          +-------------------+          +-------------------+
          |  SERVICE DOMAIN   |<-------->|  SERVICE DOMAIN   |
          +-------------------+          +-------------------+
             ^            ^                 ^            ^
             |            |                 |            |
             |            |                 |            |
             v            v                 v            v
          +-------+    +-------+         +-------+    +-------+
          |  UA   |    |  UA   |         |  UA   |    |  UA   |
          +-------+    +-------+         +-------+    +-------+
```

Figure 1. PRIM Service Architecture

PRIM adopts a Client-Server-Server-Client architecture. A USER AGENT only communicates with servers in its HOME DOMAIN, and only servers can communicate with other servers. These servers may be located in different domains.

3.1. Service Domain Clustering

It may be necessary to have multiple Presence and/or IM Servers to handle PRINCIPALS of a given domain. It is beyond the scope of this document to describe how servers within a domain choose to locate each other and what protocol they choose to communicate with.

4. Connection Model

PRIM is a connection-based protocol. Every protocol command is exchanged through TCP connections established between clients and servers and servers and servers.

4.1. Client-server Connections

Both for the PRESENCE SERVICE and the INSTANT MESSAGING SERVICE, USER AGENTS need to open a TCP connection with each server. This connection will remain open while the USER AGENT wishes to send or receive information to/from the PRESENCE or IM SERVICES.

When a USER AGENT establishes a connection to a server, it authenticate its PRINCIPAL using SASL. If the authentication process succeeds, the server associates that connection with the specific PRINCIPAL. After that, the server MUST ensure each request it receives through that connection pertains to that PRINCIPAL. If a request pertains to an unauthorized principal the server returns an error message.

Details of the authentication process is described in section 13.1.

4.2. Server-server Connections

When a Presence or Instant Messaging Server needs to exchange information with another server, it will resolve the recipient’s name, and start a connection. The connection may be closed by either side at any time when there are no outstanding commands on the connection from that server’s point of view. Any commands sent to a server which closed the connection before sending a reply can safely be assumed to have gone unprocessed.

When a server establishes a connection to another server, that connection end-point can be authorized to communicate on behalf of multiple PRESENTITIES or INBOXES. This authorization can take place either at connection time, or throughout the duration of the connection.

For example, if server A receives a subscription request from server B, on behalf of user thanos@networkprojects.com, server A MUST verify that server B is one of the servers of the networkprojects.com domain. If so, it will then accept other requests from server B that pertain to users of the networkprojects.com domain.

PRIM provides several methods to authenticate and authorize servers, which are described in section 13.2.

4.3. Shared Connections for Both Services

Although the PRESENCE SERVICE and the INSTANT MESSAGING SERVICE are separate, there may be implementations that choose to implement both. Additionally those implementations may prefer to share a TCP connection for both services. To do so, a USER AGENT would open a single connection and authenticate itself twice using the LOGIN command, once to the PRESENCE SERVICE and once to the INSTANT MESSAGING SERVICE. This feature is OPTIONAL for the PRIM implementations.

The server can differentiate between the commands for either service by examining the version in the request line that indicates which service the command pertains to. If the connection is to use TLS, the STARTTLS connection will only be issued once. The version used in the STARTTLS command can be that of either service.

5. Presence Model

PRIM adopts the lease model for publishing PRESENCE INFORMATION. That is, a PRESENTITY MAY have two pieces of PRESENCE INFORMATION, a lease value and a permanent value, for each tuple of the PRESENCE INFORMATION. The USER AGENT publishes the lease value and specifies a duration for that lease. The lease needs to be renewed by the USER AGENT when the duration elapses, otherwise the permanent value is published automatically by the server. If no permanent value exists, that tuple will be removed and no longer published.

This feature provides a flexible solution to handle PRESENCE INFORMATION for different communication means. While availability of some devices is subject to unexpected failure or constantly changing communication environment, that of other communication means might always be acquirable from a particular entity. The latter do not have to use the lease value. Instead it can just change the permanent value of the PRESENCE INFORMATION.

5.1. Presence Subscriptions

PRINCIPALS can subscribe to a PRESENTITY in order to receive NOTIFICATIONS when the PRESENCE INFORMATION of that PRESENTITY changes.

SUBSCRIPTIONS have a duration under which they are in effect. This duration is specified at the time that the subscription is placed (or renewed). Once that period elapses, the SUBSCRIPTION has to be either renewed by the SUBSCRIBER, or else it MUST be removed by the PRESENTITY’s Presence Server.

This renewal may be either issued by the USER AGENT, or by the SUBSCRIBER’s Presence Server on behalf of the SUBSCRIBER.

5.2. PRESENCE Publication & Distribution

Every time the PRINCIPAL controlling a PRESENTITY publishes a PRESENCE TUPLE, NOTIFICATIONS will be issued to the SUBSCRIBERS of that PRESENTITY containing the updated PRESENCE INFORMATION.

PRESENTITIES need to be able to publish different PRESENCE INFORMATION to different WATCHERS. PRESENTITIES may also choose not to publish PRESENCE INFORMATION to designated WATCHERS ("polite blocking"). To do so, each PRESENTITY can classify WATCHERS in different Classes. A WATCHER MUST only exist in one Class. This classification takes place in the Class Table.

Every presence update request MUST contain Class information for which it is published. When the server receives the update request, it retrieves from the Class Table the WATCHERS that need to receive the update NOTIFCATIONS.

Note: PRINCIPALS can update one PRESENCE TUPLE at a time. However NOTIFICATIONS contain the whole PRESENCE INFORMATION for a PRESENTITY. The reason for this is that PRESENCE INFORMATION may be encrypted end-to-end and thus, if only one PRESENCE TUPLE was published the receiving remote server may not be able to determine which existing tuple the new one should replace.

6. Instant Messaging Model

INSTANT INBOXes are entities that receive INSTANT MESSAGES. When a USER AGENT wishes to start receiving INSTANT MESSAGES, it issues a LISTEN command to that INSTANT INBOX. Conversely, when it no longer wishes to receive INSTANT MESSAGES from that INSTANT INBOX, it issues a SILENCE command.

INSTANT INBOXes have two states, as described in RFC 2779: OPEN and CLOSED. An INBOX is OPEN when at least one PRINCIPAL is listening to that inbox. It is CLOSED when there are no PRINCIPALS listening to the INBOX.

If an INSTANT MESSAGE is sent to an INBOX that has multiple PRINCIPALS listening, the message is considered to be delivered successfully if at least one PRINCIPAL receives it.

7.   Namespace

In the following sections including this, the protocol specification of PRIM is described. The ABNF [RFC 2234] is used to define the syntax of the protocol elements.

7.1.   Identifiers

The next ABNF defines a Presence or IM identifiers, which are used to identify PRESENTITIES and INSTANT INBOXes respectively. It also defines IP address formats to be refered in some header definitions.

```
presence-id     = word-pres ":" local-part "@" domain
im-id           = word-im ":" local-part "@" domain
local-part      = 1*( unreserved / escaped )
unreserved      = ALPHA / DIGIT / "!" / "$" / "&" / "’" / "*"
                / "." / "+" / "-" / "/" / "=" / "?" / "_" / "~"
escaped         = "%" hex-char hex-char
hex-char        = DIGIT / "A" / "B" / "C" / "D" / "E" / "F"
                / "a" / "b" / "c" / "d" / "e" / "f"
domain          = 1*domain-label *("." 1*domain-label)
domain-label    = 1*( unreserved / escaped )
word-pres       = %x70.72.65.73     ; "pres"
word-im         = %x69.6D           ; "im"
decimal-byte    = 1*3DIGIT
ALPHA           = <defined by RFC 2234 -- ’A’-’Z’ / ’a’-’z’>
DIGIT           = <defined by RFC 2234 -- ’0’-’9’>

hex4            = 1*4hex-char
hexseq          = hex4 *(":" hex4)
ip6-address     = hexseq / hexseq "::" [ hexseq ] / "::" [ hexseq ]
ip4-address     = "::" 1*1decimal-byte 3*3("." 1*1decimal-byte)
```

The PRIM Presence and IM identifiers are defined so as to align with CPIM [CPIM]. They have the form of URI [RFC2396] and the same URI schemes are selected for Presence identifiers ("pres:") and IM identifiers ("im:").

The syntax for the "local-part" and "domain" of those identifiers are similar to that for email addresses, specified as addr-spec in [RFC822]. But, the characters defined in this specification is restricted so as to conform to the URI syntax [RFC2396]. The characters which are not allowed in this definition MUST be escaped. Also note that, unlike a mailto: URL [RFC 2368], a pres: or im: URL cannot contain multiple addresses.

Moreover, The syntax for "domain-label" here is so defined that it will be conformant to the prospective specification of the Internationalized Domain Name [IDN]. A string for "domain" MUST be a valid domain name according to the rules currently in existence.

Followings are some examples of valid Presence and IM identifiers:

```
pres:joe@example.net
im:%22Jane%20Smith%22@domain.com
```

A PRIM USER AGENT SHOULD recognize a PRESENTITY or INSTANT INBOX identifier without the scheme if it is entered in a PRESENCE or INSTANT MESSAGING context. Similarly, a USER AGENT SHOULD display a PRESENCE or INSTANT MESSAGING identifier without the scheme if it is displayed in a PRESENCE or INSTANT MESSAGING context.

A PRINCIPAL may or may not have the same IDENTIFIER for its PRESENTITY and its IM INBOX. However, for an integrated Presence and IM service, the service SHOULD NOT assign the IDENTIFIERS which are different only in the scheme part to different PRINCIPALS.

7.2. Name Resolution

Should two PRINCIPALS, each in a different SERVICE DOMAIN, need to communicate, their corresponding Servers will need to locate each other, given the IDENTIFIERS of the PRINCIPALS. Moreover, when a USER AGENT wishes to connect to the SERVICE DOMAIN, it also needs to locate the servers. PRIM reuses the existing Domain Name Services to achieve this.

If the domain of the two PRINCIPALS is the same, and they are handled by different servers, there needs to be a protocol to allow the servers to interact. This protocol is not in the scope of the current document.

7.2.1. Client-Server Connections

A USER AGENT MAY support site-specific means of server discovery, but it SHOULD support the following standard discovery algorithm: the USER AGENT performs a SRV [RFC 2782] lookup for its home domain using the protocol "tcp" and the service "presence-clients" for the PRESENCE SERVICE or "im-clients" for the INSTANT MESSAGING SERVICE.

If no SRV record is present, the USER AGENT performs an A-record look up on the domain and uses the resulting IP addresses with the allocated port [xxx] for the PRESENCE SERVICE or [xxx] for the INSTANT MESSAGING SERVICE.

7.2.2. Server-Server Connections

A server MUST discover a remote domain’s server using the following algorithm: the server performs a SRV lookup for the remote domain using the protocol "tcp" and the service "presence" (for PRESENCE) or "im" (for INSTANT MESSAGING). If no SRV record is present, the server performs an A lookup on the remote domain and uses the resulting IP addresses with the allocated port [xxx] for PRESENCE or [xxx] for INSTANT MESSAGING.

Note: The protocol is capable of using four different TCP ports: two for the PRESENCE SERVICE and two for the INSTANT MESSAGING SERVICE. Within each service, there may be different ports for client and server connections. However, the usage of one, two, three or four ports will be possible for different needs. The protocol ensures there is no ambiguity between commands received from different services, or from clients/servers.

8. Command Structure

A connection transports a sequence of commands. The underlying character set for commands is Unicode, represented in UTF-8 [RFC 2279]. Command bodies are an exception; they should be treated as unprocessed octets. An implementation MUST properly handle arbitrary binary data in the body. A command is either a request or a response.

```
        PRIM-command = request / response
```

Requests and responses use the generic command format of [RFC822] for transferring entities (the body of the command). Both types of command consist of a start-line, one or more header fields (also known as "headers"), an empty line (i.e., a line with nothing preceding the CRLF) indicating the end of the header fields, and an optional command-body.

```
        generic-command = start-line
                          *command-header
                          CRLF
                          [ command-body ]

             start-line = request-line / response-line
```

Receivers of commands SHOULD ignore any empty line(s) received where a start-line is expected.

8.1.   Requests

A request method includes the method to be applied to the resource, the protocol version, and the data needed for asynchronous requests.

```
          request-line = method
                         SP version
                         SP request-identifier
                         SP content-length
                         CRLF
```

8.1.1. Method

The method token indicates the method to be performed on the resource.

```
                method = "LOGIN"               ; Section 10.1
                       / "STARTTLS"            ; Section 10.2
                       / "LOGOUT"              ; Section 10.3
                       / "PING"                ; Section 10.4
                       / "VERIFYSERVER"        ; Section 10.5
                       / "SETACL"              ; Section 10.6.1
                       / "GETACL"              ; Section 10.6.2
                       / "SUBSCRIBE"           ; Section 11.1.1
                       / "UNSUBSCRIBE"         ; Section 11.1.2
                       / "CANCELSUBSCRIPTION"  ; Section 11.1.3
                       / "FETCH"               ; Section 11.1.4
                       / "PUBLISH"             ; Section 11.2.1
                       / "REMOVE"              ; Section 11.2.2
                       / "NOTIFY"              ; Section 11.3
                       / "SETCLASSTABLE"       ; Section 11.4.2
                       / "GETCLASSTABLE"       ; Section 11.4.3
                       / "STARTWATCHERNOTIFY"  ; Section 11.4.4
                       / "STOPWATCHERNOTIFY"   ; Section 11.4.5
                       / "WATCHERNOTIFY"       ; Section 11.4.6
                       / "LISTEN"              ; Section 12.1.1
```

```
                          / "SILENCE"            ; Section 12.1.2
                          / "SEND"               ; Section 12.2
```

8.1.2. Version

The version identifies the version of the protocol in use.

```
                  version = "PP/1.0" / "IMP/1.0"
```

PP is used to identify the Presence Protocol, and is used for all the requests and responses within the PRESENCE SERVICE. IMP is utilized by all requests and responses within the INSTANT MESSAGING SERVICE.

[Note that the definition of the "version" should be amended to be more generally. Also, the versioning policy and the semantics are necessary to be described. TBD.]

8.1.3. Request Identifier

Request identifiers are used to implement asynchronous requests.

```
          request-identifier = 1*[ALPHA / DIGIT] / "-"
```

An endpoint of a connection is responsible for generating request identifiers, and the request identifiers are used to match responses it receives with the requests it has sent. The other endpoint of a connection is responsible for labeling a response with the identifier it received in the request. An identifier may be reused after the endpoint receives the response to the request with the identifier.

The request identifier of a command is "-" if and only if the request expects no reply. If an endpoint receives a request with the request identifier "-", it MUST NOT send any response to the request.

8.1.4. Content Length

The content-length header contains the length of the command body in bytes.

```
           content-length = 1*DIGIT
```

8.2.   Responses

A response includes many of the same fields as a request with the addition of a status code and a response phrase.

```
            response-line = version
                            SP request-identifier
                            SP content-length
                            SP status-code
                            SP response-phrase
                            CRLF
```

The request identifier in the response MUST NOT be "-".

The status-code is a 3 digit code and the response-phrase is a short message description. The values are defined in Appendix A.

Some status codes are common to all commands, whereas others are only used by a subset of commands. Common status codes to all commands are:

```
        200 OK
        300 Redirect
        400 Bad Request
        401 Unauthorized (except for LOGIN)
        402 Forbidden (except for LOGIN, STARTTLS, PING)
        501 Internal Server Error
        503 Version Not Supported
```

9.   Command Headers

Command headers are defined as follows:

```
            command-header = (common-header
                              / presence-header
                              / im-header
                              )
                             CRLF

             common-header = (from-header
                              / to-header
                              / auth-state-header
                              / SASL-mechanism-header
                              / redirect-header
                              / content-type-header
                              / server-address-header
                              / astrength-header
                              / user-agent-id-header
                              / max-content-length-header
                              / date-header
                              )
```

```
           presence-header = (class-header
                             / tuple-id-header
                             / duration-header
                             / pi-type-header
                             / watcher-type-header
                             )

                 im-header = (message-id-header
                             / conversation-id-header
                             / reply-to-header
                             )
```

9.1.   Common Headers

9.1.1. From

Identifies the PRESENTITY or INBOX that issued this command, or that it was issued on behalf of.

```
             from-header = "From: " ( presence-id / im-id )
```

The receiving end of a command SHOULD always check that the sender is authorized to send commands on behalf of the identifier in the from-header, as described in Sections 13.

9.1.2. To

Specifies the PRESENTITY or INBOX this command is intended to.

```
               to-header = "To: " ( presence-id / im-id )
```

9.1.3. Auth-State

Indicates the status in the authentication process in the LOGIN command.

```
             auth-state-header = "Auth-State: "
                               ( "init"
                               / "continue"
                               / "abort" )
```

9.1.4. SASL-Mechanism

Specifies the SASL mechanism in the LOGIN request or the response to the LOGIN request. In the request, the SASL mechanism the USER AGENT wants to use MUST be specified. When used in the response, one or more mechanisms which the server supports MAY be specified.

```
             SASL-mechanism-header = "SASL-Mech: " mechanisms
                        mechanisms = mechanism [ *(SP mechanism) ]
                         mechanism = 1*20(ALPHA / DIGIT / "-" / "_")
```

9.1.5. Redirect

When a server cannot handle requests from a USER AGENT or other server, it issues an error repsponse "300 Redirect" which includes the redirect-header. This lets the caller know that its request cannot be handled at this server and an alternative server address and port are provided.

```
           redirect-header = "Redirect: " address SP port
                   address = domain / ip4-address / ip6-address
                      port = 1*DIGIT
```

9.1.6. Content-Type

A command-body MUST NOT be included unless the description of the particular method allows it. If a command-body is included, the protocol command headers MAY include a Content-Type as specified in [RFC 2045]; if no Content-Type is provided, the default is "application/presence" for Presence commands, "text/plain; charset=UTF-8" for Instant Messaging commands, and "application/octet- stream" for the LOGIN command.

The Content-Transfer-Encoding header from [RFC 2045] is not necessary and MUST NOT be included in any command or response. An implementation which receives a Content-Transfer-Encoding header should reject the command with an error 400 Bad Request.

9.1.7. Server-Address

Indicates the IP address for the server that is initiating the connection. This header is used in the VERIFYSERVER method to show the address of the server that needs verification (see Sections 10.5 and 13.2).

```
     server-address-header = "Server-Address: "
                             ( ip4-address / ip6-address )
```

9.1.8. AStrength

When a server acts as a relay, it MUST communicate to the next node a rough indication of the authentication strength of the previous hops using the "Astrength" header. The syntax for the Astrength header is:

```
astrength-header = "AStrength: " astrength
       astrength = "strong" / "medium" / "weak" / "none"
```

The meanings of the astrength values are:

strong	Command authenticity and integrity cannot be compromised by an attacker who has full control of all network links, assuming no compromise of keying materials, installed software, or cryptographic algorithms.
medium	Command authenticity or integrity could be compromised by a packet substitution or DNS spoofing attack.
weak	Command could be forged by an attacker who has previously been a passive listener on one or more network links.
none	Command could be forged by an attacker with no special information.

Examples of medium protection include one-time passwords [RFC 2289] and HTTP digest authentication [RFC 2617 section 3]. Examples of weak protection include cleartext passwords or security protocols subject to replay attacks.

If a server or USER AGENT receives a command with no Astrength header, it should assume that the equivalent Astrength is "none", with one exception: If a server receives a command directly from a USER AGENT, it should determine the strength of that connection and use the appropriate AStrength.

A server relaying a command MUST communicate the weaker of the strength of the connection it received the command on and the Astrength value communicated from the last entity.

A server MAY choose to reject a command with a "410 AStrength Too Weak" error because it does not come with sufficient authentication strength (either as reported by the Astrength value or based on the connection from the immediate requestor). A relay MUST NOT reject a response on the basis of insufficient authentication strength.

Note that, separately from connection-level authentication, an operation may be authenticated using an end-to-end signature. The Astrength header does not bear any relation to this kind of authentication.

An example scenario: a PRIM USER AGENT connects to a server for example.net and authenticates using a weak mechanism. It then issues a "send" command from alice@example.net to bob@domain.com. The example.net server connects to domain.com, authenticates using DNSSEC- signed public keys and forwards the IM with "Astrength: weak" because the previous link was authenticated with a weak. The domain.com server sends the command to the clients receiving commands for bob@domain.com with "Astrength: weak" since that was the authentication value claimed by example.net, even though domain.com received the command over a strongly authenticated link.

Another example scenario: a PRIM client connects to a server for example.net and authenticates using some strong SASL mechanism as alice. It then issues a "send" command from alice@example.net to bob@domain.com. The example.net server connects to domain.com and authenticates, but example.net’s public key DNS record is not signed, so it could have been forged by a DNS spoofing attack. The example.net server sends the IM with "Astrength: strong" because it received the command from Alice over a strongly authenticated link; however, the domain.com server will weaken the Astrength to "Medium" when forwarding the command to Bob’s clients.

9.1.9. User-Agent-ID

For a single PRINCIPAL, mutliple USER AGENTS can open TCP connections to a server. A User Agent ID is used to distinguish these USER AGENTS of the same PRINCIPAL by the server. A User Agent ID is generated by the server and contained in a response to a LOGIN command.

```
     user-agent-id-header = "User-Agent-ID: " 1*(unreserved / escaped)
```

[This header might be unnecesary once we discarded multiple connections.]

9.1.10. Max-Content-Length

Used by the USER AGENT to indicate to the server that it MUST NOT send commands with length greater than the value supplied.

```
    max-content-length-header = "Max-Content-Length: " 1*DIGIT
```

9.1.11. Date

Specifies the date and time this command was originally issued. PRIM adopts the date syntax as defined in Section 15.5, i.e. specified in [RFC1123].

```
               date-header = "Date: " date-time
                                     ; as defined in Section 15.5
```

[It will be affected by the CPIM specification because it would be preferable to have the same format with it. Need more discussions.]

9.2.   Presence Headers

9.2.1. Class

Identifies the class(es) to which the PRESENCE TUPLE should be published (See Section 14.1).

```
                class-header = "Class: " class [ SP class ]
                       class = 1*(unreserved / escaped)
```

9.2.2. Tuple-ID

Identifies the PRESENCE TUPLE that is being published. This can be any string that uniquely identifies the tuple or it MAY be the CONTACT ADDRESS for the communication means that corresponds to the PRESENCE TUPLE.

```
             tuple-id-header = "Tuple-ID: " 1*(unreserved / escaped)
```

9.2.3. Duration

Specifies the amount of seconds this command should remain in effect. Used for the leased operations.

```
             duration-header = "Duration: " 1*DIGIT
```

9.2.4. PI-Type

Indicates whether new leased PRESENCE INFORMATION is being published, an existing lease is being renewed, a permanent value is being published, or a leased value is being replaced with a permanent value.

```
               pi-type-header = "PI-Type: "
                              ( "leased"
                              / "permanent"
                              / "renew"
                              / "revert")
```

9.2.5. Watcher-Type

Used by the NOTIFYWATCHER command to specify whether a FETCH or SUBSCRIBE operation has occured.

```
     watcher-type-header = "Watcher-Type: " ("fetch" / "subscribe")
```

9.3. IM Headers

9.3.1. Message-ID

The message-id-header specifies the identifier of each IM, which distinguishes the message from others. The sender must generate a unique message-id for each IM sent.

```
     message-id-header = "Message-ID" 1*(DIGIT / ALPHA) ": " im-id
```

9.3.2. Conversation-ID

The conversation-id is used in the SEND command to identify the conversation channel shared by the participants of an IM exchange. A "conversation channel" means a virtual channel which consists of a thread of IMs. When a PRINCIPAL replies to an IM, the reply MUST have the same conversation-id header.

```
   conversation-id-header = "Conversation-ID: " 1*(unreserved / escaped)
```

9.3.3. Reply-To

The reply-to-header is optionally specified in a SEND command. It indicates an INSTANT INBOX identifier where the sender would prefer to receive any replies. The recipient SHOULD use the "reply-to" header, instead of the "from" header, if the former exists.

```
     reply-to-header = "Reply-To: " im-id
```

10. Common Commands

The commands described in this section apply to both the PRESENCE and INSTANT MESSAGING services.

10.1. Connection Setup - LOGIN

```
   Direction: C->S
   Required Headers: from-header,
                     auth-state-header,
                     SASL-mechanism-header,
                     max-content-length
   Optional Headers: none
   Command Body: Required
```

The initiating client MUST issue a LOGIN request to the server in order to start the authentication process. As described in Section 13.2, server-server connections are not authenticated at connection time.

If the authentication process is not successful the TCP connection MUST be dropped. The LOGIN request MAY be preceded by the STARTTLS request when the implementations support TLS for a secure connection. Any other requests that are received before the authentication completed MUST receive an "Unauthorized" response.

The authentication process is not necessarily completed in a single request/response pair, but it can be fulfilled in a sequence of the request/response pairs. The auth-state-header MUST be used to indicate the state of the authentication process.

The command-body in the LOGIN request carries the challenge information for the respective SASL mechanism.

Return Codes:

100 Authentication Continued: This response may possibly carry a command-body with information pertaining to the SASL challenge, and a SASL-mechanism-header specifying the SASL mechanism supported by the server. The originator needs to send other LOGIN command, with auth-state-header as "continue", and the response to the challenge in the command-body.

200 OK: The sender is authenticated and the connection may be used to transport further commands. The server MUST include the user-agent-id-header in its response. If this is the first connection from a USER AGENT, the server will assign a new value, unique to the connecting PRESENTITY or INSTANT INBOX. If this is an additional connection, the server will simply return the header as sent by the client.

406 Authorization Failed: The operation failed to authenticate the connection. No further commands are allowed and the receiver MUST terminate the connection.

409 Already Authenticated: This is returned if a LOGIN command has already succeeded.

10.2.   Connection Setup - STARTTLS

      Direction: C->S, S->S
      Required Headers: none
      Optional Headers: none
      Command Body: none

A client or server MAY issue STARTTLS request to upgrade a TCP connection to a TLS [TLS] enabled one. Implementations that support TLS MAY issue a STARTTLS request prior to issuing any other requests.

Once the client credentials are successfully exchanged using TLS negotiation, the "EXTERNAL" SASL mechanism MAY be used in the subsequent LOGIN process. The "PLAIN" SASL mechanism SHOULD NOT be used if the STARTTLS upgrading process fails to establish a fully strong encryption layer.

The Response MUST NOT carry a command-body.

Return Codes:

200 OK: The TLS negotiation should start. Once a STARTTLS command issued, the initiator MUST NOT issue further requests until a server response is received and the TLS negotiation is completed.

501 Not Implemented: TLS is not implemented and thus the client must authenticate itself using the LOGIN method.

10.3.   Connection Shutdown - LOGOUT

      Direction: C->S, S->S
      Required Headers: none
      Optional Headers: none
      Command Body: none

The receiver of the LOGOUT command MUST NOT send any response.

10.4.   Testing a connection - PING

      Direction: C->S, S->S, C->S
      Required Headers: none
      Optional Headers: none
      Command Body: none

When a peer in a connection wants to verify if the connection is alive, it may send a PING command. No response is expected from the other peer.

A successful transmission of a PING does not guarantee its reception at the other end, nor does it verify that all is well with its peer. However the transmission of the PING may provoke an error, and thereby causing the sending peer to realize there is a problem with the connection. If this happens the USER AGENT or server assumes an implicit LOGOUT command.

10.5. Verifying a server's authority - VERIFYSERVER

Direction: S->S
Required Headers: server-address-header
Optional Headers: none
Command Body: none

As described in section 13.2, when a server needs to verify whether another server (known through its IP address) belongs to a given domain, it performs one or more DNS lookups. Large domains with a significant amount of servers might not be able to publish every entry for every server, due to DNS limitations. Thus a DNS lookup might not be sufficient to determine whether a given server belongs to a given domain.

If it is not possible to verify the domain of a server through a DNS lookup, a VERIFYSERVER command can be issued.

The VERIFYSERVER MAY be issued in a new TCP connection, without previous LOGIN. The verifying server will issue the command to any of the addresses returned in the DNS lookup.

The server-address-header specifies the IP address of the server that needs verification.

The response MUST NOT have a command body.

Return Codes:

200 OK: the server does belong to that domain.

403 Resource Not Found: the server does not belong to this domain.

10.6. Access Control

The PRESENCE and INSTANT MESSAGING SERVICES use the Access Control Lists to determine what operations PRESENTITIES and INSTANT INBOXES are permitted to perform on protected resources. The operations that can be performed on PRESENTITIES are different than those for INSTANT INBOXES. However, the access control mechanisms are the same and thus we will describe them together.

10.6.1. SETACL

Direction: C->S
Required Headers: from-header,
Optional Headers: none
Command Body: Required

This operation is used by the USER AGENT to send an access control list to the server.

The command-body MUST carry a valid XML document according to the DTD given in Section 14.

The from-header specifies the PRESENTITY or INBOX this ACL pertains to.

The Response MUST NOT have a command-body.

Return Codes:

200 OK: The access control list sent replaced the one currently active in the server.

400 Bad Request: The command was malformed or the command-body did not carry a valid XML document. The access control list did not replace the current one.

402 Forbidden: The PRINCIPAL authenticated in the current connection does not own the requested resource, thus cannot set the ACL for it. The access control list did not replace the current one.

403 Resource Not Found: The resource (PRESENTITY or IM INBOX) does not exist. The access control list did not replace the current one.

10.6.2. GETACL

Direction: C->S
Required Headers: from-header,
Optional Headers: none
Command Body: Required

This method is issued to retrieve the current access control list for a specific resource from a Presence or IM server.

The from-header specifies the identifier of the resource (PRESENTITY or IM INBOX) for which the ACL is required.

The Return Codes are:

200 OK: The command-body of the Response has a valid XML document according to the DTD presented in Section 14, representing the current ACL for the resource requested.

402 Forbidden: The PRINCIPAL authenticated in the current connection does not own the requested resource, thus cannot get the ACL for it. No command-body is present.

403 Resource Not Found: The resource (PRESENTITY or IM INBOX) does not exist. No command-body is present.

11. Presence Service Commands

This section describes the details of the protocol for the PRESENCE SERVICE.

11.1. Placement, Renewal and Removal of SUBSCRIPTIONS

11.1.1. SUBSCRIBE

```
Direction: C->S, S->S
Required Headers: from-header,
                  to-header,
                  duration-header,
                  astrength-header (optional on C->S)
Optional Headers: none
Command Body: none
```

The SUBSCRIBE method is used to express a WATCHER's interest on the PRESENCE INFORMATION of a PRESENTITY. There are two scenarios where the method is issued: when a

- o WATCHER wishes to establish a new SUBSCRIPTION to a PRESENTITY, or

- o Presence Server or USER AGENT needs to renew a SUBSCRIPTION on behalf of a WATCHER

The from-header identifies the PRESENTITY requesting the SUBSCRIPTION.

The to-header specifies the PRESENTITY to subscribe to.

The duration-header specifies the amount of seconds that this subscription is valid for.

The Return Codes are:

200 OK: The SUBSCRIPTION was placed successfully. The command-body carries the current PRESENCE INFORMATION for the PRESENTITY.

201 Duration Adjusted: The SUBSRIPTION was placed successfully, yet a different duration was set and this is indicated in the duration-header of the response.

402 Forbidden: The PRESENTITY authenticated in the current connection does not have rights (through the current ACL) to SUBSCRIBE to the PRESENTITY requested. No command-body is present.

403 Resource Not Found: The PRESENTITY does not exist. No command-body is present.

505 Too Many Subscriptions: The maximum amount of SUBSCRIPTIONS for that PRESENTITY has been reached. No command-body is present.

If a SUBSCRIPTION already exists between a WATCHER and a PRESENTITY, then a successful SUBSCRIBE request from the WATCHER updates the duration of the SUBSCRIPTION to the value carried in the request.

11.1.2. UNSUBSCRIBE

```
Direction: C->S, S->S
Required Headers: from-header,
                  to-header,
                  astrength-header (optional on C->S)
Optional Headers: none
Command Body: none
```

The UNSUBSCRIBE method indicates that a WATCHER is no longer interested in receiving NOTIFICATIONS for changes in PRESENCE INFORMATION of a PRESENTITY.

It may either be issued by a USER AGENT or a Presence Server on behalf of the WATCHER.

The from-header identifies the WATCHER requesting the SUBSCRIPTION cancellation.

The to-header specifies the PRESENTITY to unsubscribe from.

The Response MUST NOT carry a command-body. The Return Codes in the Response are:

200 OK: The SUBSCRIPTION was removed.

404 Subscription Not Found: there is no SUBSCRIPTION from the specified WATCHER to the specified PRESENTITY.

Note: When the duration of a SUBSCRIPTION elapses, without the reception of a renewal, the Presence Server MUST assume an implicit UNSUBSCRIBE method has been received.

11.1.3. CANCELSUBSCRIPTION

```
Direction: S->S, S->C
Required Headers: from-header,
                  to-header,
                  astrength-header
Optional Headers: none
Command Body: none
```

The CANCELSUBSCRIPTION method is used when a PRINCIPAL controlling a PRESENTITY denies SUBSCRIPTION access (through a SETACL) to a current WATCHER.

The Presence Server, realizing that the WATCHER already had a subscription, will send a CANCELSUBSCRIPTION command to the WATCHER, letting it know that its SUBSCRIPTION has been cancelled. No future NOTIFICATIONS will be sent to this WATCHER.

The from-header specifies the PRESENTITY that is cancelling the subscription.

The to-header specifies the WATCHER who’s SUBSCRIPTION has been cancelled.

No response is needed for this command.

11.1.4. FETCH

```
Direction: C->S, S->S
Required Headers: from-header,
                  to-header,
                  astrength-header (optional on C->S)
Optional Headers: none
Command Body: none
```

The FETCH method is used when a WATCHER wishes to retrieve the present value of the PRESENCE INFORMATION for a PRESENTITY.

The from-header identifies the WATCHER requesting the PRESENCE

INFORMATION

The to-header specifies the PRESENTITY that the WATCHER is interested in.

The Return Codes are:

200 OK: The FETCH was successful. The command-body carries the current PRESENCE INFORMATION for the PRESENTITY.

402 Forbidden: The PRESENTITY authenticated in the current connection does not have rights (through the current ACL) to FETCH the PRESENCE INFO. No command-body is present.

403 Resource Not Found: The PRESENTITY does not exist. No command-body is present.

11.2. Publication & Removal of PRESENCE INFORMATION

11.2.1. PUBLISH

```
Direction: C->S
Required Headers: from-header,
                  pi-type-header,
                  duration-header,
                  class-header,
                  tuple-id-header
Optional Headers: none
Command Body: Required
```

This method is used to publish one TUPLE of PRESENCE INFORMATION for a PRESENTITY. It is used to set the leased and permanent values of the PRESENCE INFORMATION.

The command-body MUST carry one XML document as described in Section 15, corresponding to the PRESENCE TUPLE the USER AGENT wishes to publish.

The USER AGENT SHOULD ensure that at most one PRESENCE TUPLE is published for a given address at any moment for a specific class.

All successful PUBLISH operations will cause NOTIFICATIONS to be sent to all the SUBSCRIBERS of that PRESENTITY, except:

- o Renewal of an existing lease
- o Publishing of permanent PI, while a valid lease exists

In addition, the expiration of a lease will also cause NOTIFICATIONS to be sent to all the SUBSCRIBERS of the PRESENTITY.

The headers used for this method are:

from-header: identifies the PRESENTITY publishing the PRESENCE INFORMATION

pi-type-header: if this header carries the value "leased", the PRESENCE INFORMATION will be valid only for the period of time specified in the duration-header (in seconds). After that time elapses and the lease is not renewed the PRESENCE INFORMATION reverts to its permanent value.

The leased value can be renewed if the USER AGENT issues a PUBLISH operation with a "renew" pi-type-header.

The leased value can be removed if the USER AGENT issues a PUBLISH operation with a "revert" pi-type-header.

If "permanent" is specified, this determines that value that the PRESENCE INFORMATION will revert to when the lease expires.

duration-header: only available for the "leased" value in the type-header, it represents the amount of seconds for which the PRESENCE TUPLE sent will be valid. After that period, if no PUBLISH method is received, the server MUST publish the permanent value of the PRESENCE TUPLE or, if no permanent value exists it MUST remove the PRESENCE TUPLE.

class-header: the Classes the PRESENCE TUPLE should be published to. This value is used to determine which WATCHERS should receive this information.

tuple-id-header: identifies the PRESENCE TUPLE that is being published. The server uses this value to determine which existing PRESENCE TUPLE it must remove or replace with the current one.

The Response for this command MUST NOT carry any command-body. The Return codes for this command are:

200 OK: the PRESENCE TUPLE has been accepted. If it is leased or if it is permanent and no lease exists, the PRESENTITY’S PRESENCE INFO will be distributed to all SUBSCRIBERS in the classes specified in the class-header.

400 Bad Request: The command was malformed or the command-body did not carry valid PRESENCE INFORMATION as defined in section 15. The PRESENCE TUPLE was not accepted.

402 Forbidden: The PRESENTITY authenticated in the current connection does not have rights (through the current ACL) to PUBLISH PRESENCE INFORMATION for this PRESENTITY.

403 Resource Not Found: The PRESENTITY does not exist.

11.2.2. REMOVE

```
Direction: C->S
Required Headers: from-header,
                  class-header,
                  tuple-id-header
Optional Headers: none
Command Body: none
```

This method is used to remove one TUPLE of PRESENCE INFORMATION for a PRESENTITY. Once removed the PRESENCE TUPLE will no longer be published to WATCHERS.

This command will remove both the leased and permanent values of the tuple, and also trigger the appropriate NOTIFICATIONS.

The headers used by this method are:

from-header: identifies the PRESENTITY publishing the PRESENCE INFORMATION

class-header: the Classes from which the PRESENCE TUPLE should be removed from. This value is used to determine which WATCHERS should receive this information.

tuple-id-header: identifies the PRESENCE TUPLE that is being removed.

The Response for this command MUST NOT carry any command-body. The Return codes for this command are:

200 OK: the PRESENCE TUPLE has been removed

402 Forbidden: The PRESENTITY authenticated in the current connection does not have rights (through the current ACL) to REMOVE any PRESENCE TUPLES.

403 Resource Not Found: The PRESENTITY does not exist or the tuple does not exist.

The successful removal of a PRESENCE TUPLE will trigger the server to send NOTIFICATION commands to all the SUBSCRIBERS for that

PRESENTITY.

11.3. Propagation of PRESENCE INFORMATION - NOTIFY

```
Direction: S->S, S->C
Required Headers: from-header,
                  to-header,
                  astrength-header (optional on C->S)
Optional Headers: none
Command Body: Required
```

The NOTIFY command informs WATCHERS when the PRESENCE INFORMATION of the PRESENTITY they have SUBSCRIPTIONS to has changed. This command is always issued by Presence Servers.

When a successful PUBLISH method with "lease" type is processed by the Presence Server, it will go through the list of SUBSCRIPTIONS, filtered by the Class Table, and identify all the WATCHERS that need to be notified of the change. It will then send one NOTIFY command for each of these WATCHERS.

When some tuple of leased PRESENCE INFORMATION expires (the duration time elapses) without receiving another PUBLISH command, that PRESENCE INFORMATION needs to be reverted to the permanent values. The Presence Server, then, with the same algorithm as before, sends new NOTIFY commands to the WATCHERS. The NOTIFY will carry the whole PRESENCE INFORMATION and not just the modified tuple.

The command-body MUST carry a valid XML document as described in Section 15, corresponding to the PRESENCE INFORMATION for the PRESENTITY.

The headers for this command are:

from-header: the PRESENTITY this NOTIFICATION is about

to-header: the WATCHER that needs to receive this information

The Response MUST NOT carry any command-body. The Return Codes are:

200 OK: the PRESENCE INFORMATION was received and processed correctly.

400 Bad Request: The command was malformed or the command-body did not carry a valid XML document. The PRESENCE INFORMATION was not accepted.

403 Resource Not Found: no such WATCHER exists.

11.4. Presence Privacy Management

11.4.1. Access Control List

For the purposes of the PRESENCE SERVICE the operations available for the SETACL and GETACL methods defined in Section 10.6 are restricted to FETCH, SUBSCRIBE, PUBLISH and REMOVE.

11.4.2. SETCLASSTABLE

Direction: C->S
Required Headers: from-header
Optional Headers: none
Command Body: Required

Through the SETCLASSTABLE method, the USER AGENT sets the list of WATCHERS belonging to a given Class.

The command-body of the SETCLASSTABLE command MUST carry a CLASSTABLE XML Element, as described in Section 14.1.

The from-header specifies the PRESENTITY for which this Class Table should be set.

The Response MUST NOT carry a command-body. The possible Return Codes are:

200 OK: The class table sent replaced the one currently active in the server.

400 Bad Request: The command was malformed or the command-body did not carry a valid XML document. The Class table did not replace the current one.

402 Forbidden: The PRESENTITY authenticated in the current connection does not own the PRESENTITY specified in the from-header. The Class table did not replace the current one.

403 Resource Not Found: The PRESENTITY does not exist. The Class table did not replace the current one.

11.4.3. GETCLASSTABLE

Direction: C->S
Required Headers: from-header
Optional Headers: none
Command Body: none

The GETCLASSTABLE method is used by the USER AGENT to retrieve the Class Table for a given PRESENTITY. The response to GETCLASSTABLE command contains a CLASSTABLE XML element in the command-body.

The from-header specifies the identifier of the PRESENTITY for which the Class Table is required.

The Return Codes are:

200 OK: The command-body of the Response carries a CLASSTABLE XML Element, as described in Section 14.1., representing the Class Table for the PRESENTITY requested.

402 Forbidden: The PRESENTITY authenticated in the current connection does not own the PRESENTITY in the from-header. No command-body is present.

403 Resource Not Found: The PRESENTITY does not exist. No command-body is present.

11.4.4. STARTWATCHERNOTIFY

Direction: C->S
Required Headers: from-header
Optional Headers: none
Command Body: none

This command instructs a server to start sending watcher information to a USER AGENT.

The from-header identifies the PRESENTITY requesting the NOTIFICATIONS.

The Response to STARTWATCHERNOTIFY may carry a command-body, depending on the return Code:

200 OK: The server will start sending NOTIFICATIONS when new SUBSCRIPTIONS are placed or when the a FETCH is perfored on the PRESENTITY. The command-body carries an XML document SUBSCRIBERS with the information about the current SUBSCRIBERS of the specified presentity.

402 Forbidden: The PRESENTITY authenticated in the current connection does not have rights to perform the requested operation. No command-body is present.

403 Resource Not Found: The PRESENTITY does not exist. No command-

body is present.

The XML document carried in the command-body if the operation is successful MUST be valid under the following DTD:

```
<!ELEMENT SUBSCRIBERS (subscriber)*>
<!ELEMENT subscriber (#PCDATA)>
```

Each subscriber element carries the identifier of one WATCHER SUBSCRIBED to the requested PRESENTITY.

11.4.5. STOPWATCHERNOTIFY

```
Direction: C->S
Required Headers: from-header
Optional Headers: none
Command Body: none
```

This command instructs a server to stop sending watcher information to a USER AGENT.

The from-header identifies the PRESENTITY requesting the termination of the NOTIFICATIONS.

The Response to STARTWATCHERNOTIFY MUST NOT carry a command-body.

200 OK: The server will stop sending watcher information NOTIFICATIONS.

402 Forbidden: The PRESENTITY authenticated in the current connection does not have rights to perform the requested operation.

403 Resource Not Found: The PRESENTITY does not exist.

11.4.6. WATCHERNOTIFY

```
Direction: S->C
Required Headers: from-header,
                  to-header,
                  watcher-type-header
Optional Headers: none
Command Body: none
```

Notifies the PRINCIPAL that a FETCH or SUBSCRIBE operation was performed on the PRESENTITY indicated in the to-header, by the PRESENTITY indicated in the from-header.

The headers for this method are:

from-header: identifies the PRESENTITY performing the FETCH or SUBSCRIBE

to-header: specifies the PRESENTITY that the FETCH or SUBSCRIBE was performed upon.

watcher-type-header: specifies whether a FETCH or SUBSCRIBE occurred.

12. Instant Messaging Service Commands

12.1. Listening to INSTANT INBOXes

12.1.1. LISTEN

```
Direction: C->S
Required Headers: from-header
Optional Headers: none
Command Body: none
```

Whenever a USER AGENT needs to receive messages targeted to an INSTANT INBOX, it MUST issue a LISTEN command to its IM Server.

The from-header specifies the INSTANT INBOX identifier that the PRINCIPAL wishes to listen to.

The Response MUST NOT carry a command-body either, and the possible Return Codes are:

200 OK: the PRINCIPAL is now listening to the INSTANT INBOX, and will receive any messages sent to it.

402 Forbidden: The PRINCIPAL authenticated in the current connection does not have rights (through the current ACL) to LISTEN to the INSTANT INBOX requested.

403 Resource Not Found: The INSTANT INBOX does not exist.

12.1.2. SILENCE

```
Direction: C->S
Required Headers: from-header
Optional Headers: none
Command Body: none
```

When a PRINCIPAL is not interested to receive any more messages from an INSTANT INBOX, it issues a SILENCE method, through which it instructs the IM Server not to forward any messages arriving to the specified INBOX through its connection.

The from-header specifies the INSTANT INBOX identifier the PRINCIPAL does not want to listen to any more.

The response to this command MUST NOT have any command-body either, and the return codes are:

200 OK: the PRINCIPAL will not longer receive any messages for the specified INBOX. This does not imply that the INBOX is closed, since there may be other PRINCIPALS listening to the same INBOX.

403 Resource Not Found: The INBOX does not exist.

408 Inbox Is Closed: the PRINCIPAL is not currently listening to that INBOX.

12.2. Sending Messages - SEND

```
Direction: C->S, S->S, S->C
Required Headers: from-header,
                  to-header,
                  message-id-header,
                  conversation-id-header,
                  astrength-header (optional on C->S)
Optional Headers: reply-to-header
Command Body: Required
```

[Note. It would be necessary to make the "SEND" command syntax compatible with the CPIM specification. We need more discussion.]

The SEND command is used to transport an INSTANT MESSAGE from the SENDER’s USER AGENT to the RECEIVER’s USER AGENT.

The command-body carries an INSTANT MESSAGE, as described in section 16.

The from-header identifies the SENDER of the message.

The to-header identifies the INSTANT INBOX the message is sent to.

The astrength-header indicates the lowest authentication strength for previous hops of the command.

The message-id-header specifies a unique identifier for each INSTANT MESSAGE.

The conversation-id-header specifies a unique identifier to distinguish a given conversation thread between multiple participants.

The reply-to-header indicates an INSTANT INBOX identifier where the sender would prefer to receive any replies.

The response to this method MUST NOT carry any command-body, and MAY have the following return codes:

101 Unknown Delivery Status: The IM Service cannot assure that the message was delivered.

200 OK: the INSTANT MESSAGE was delivered at least to one PRINCIPAL that was listening to the recipient INSTANT INBOX.

402 Forbidden: The PRINCIPAL authenticated in the current connection does not have rights (through the current ACL) to send messages to the recipient INSTANT INBOX.

408 Inbox Is Closed: the INSTANT MESSAGE could not be delivered because the recipient INSTANT INBOX was closed. This may be issued by either the IM Server if there is no-one listening to that inbox, or by a USER AGENT if it decides to block the sender (see explanation below).

The response code sent by the IM Server hosting the recipient INSTANT INBOX is always the most positive response from all the connections listening to that INBOX. Thus, if at least one USER AGENT acknowledges the message, then its server will acknowledge it too.

Note: It is important to remember that PRESENCE INFORMATION may be revealed through the responses to INSTANT MESSAGES. For example, it may be possible for someone to "ping" an INSTANT INBOX by sending messages to it, in order to deduce PRESENCE INFORMATION from the state of that INBOX. USER AGENT implementations can prevent that if necessary by returning 408 Inbox Is Closed if the sender of an INSTANT MESSAGE should not know that the INBOX is OPEN.

12.3. Access Control Lists

For the purposes of the INSTANT MESSAGING SERVICE the operations available for the SETACL and GETACL methods defined in Section 10.6 are restricted to LISTEN, SILENCE and SEND.

13. Authentication

PRIM implements security on a per-connection basis: each connection end-point is authenticated in order to establish the PRESENTITIES and INBOXES on behalf of which that end-point can communicate.

After authentication succeeds, the other end-point will accept requests pertaining to those PRESENTITIES or INBOXes, and direct requests to them over that connection.

13.1. Client-Server Authentication

When a USER AGENT establishes a connection to a server it issues a LOGIN command to authenticate its PRINCIPAL. The authentication procedure in PRIM uses SASL [SASL]. The LOGIN request and response MUST include a SASL-Mechanism header field so that the USER AGENT and the server could negotiate the SASL mechanism to be used. As SASL mechanisms, every PRIM implementation MUST implement PLAIN [SASL-PLAIN] and is RECOMMENDED to implement CRAM-MD5 [CRAM-MD5], and EXTERNAL [SASL]. It MAY also implement other mechanisms as needed.

If the authentication process succeeds, the server associates that connection with the specific PRINCIPAL. After that, the server MUST ensure each request it receives through that connection pertains to that PRINCIPAL. If a request pertains to an unauthorized principal the server returns an error message.

The LOGIN authentication procedure is sketched as follows;

(1) The initial LOGIN request

A USER AGENT issues a LOGIN request including the Auth-State header with the value "init". It MUST also contain the SASL-Mechanism header whose value is a comma-separated list of SASL mechanisms the USER AGENT is capable to use in the descending order of preference.

[Example]

```
LOGIN PP/1.0 0224 0
From: pres:suga@prim.fujitsu.com
Auth-State: init
SASL-Mech: CRAM-MD5 PLAIN
```

If the LOGIN request is acceptable, the server SHOULD respond with ’100 Authentication Continued’ response. It MUST contains the SASL-Mechanism header with the value of at least one selected SASL mechanism by the server. If a challenge-response mechanism is selected, the response MUST contain a challenge data in the body.

```
PP/1.0 0224 48 100 Authentication Continued
```

```
From: pres:suga@prim.fujitsu.com
Auth-State: init
SASL-Mech: CRAM-MD5

<20010226095208.1018677043.foo1.bar.fujitsu.com>
```

(2) The subsequent LOGIN requests

If a USER AGENT receives a '100 Authentication Continued' response to the initial LOGIN request, it SHOULD try another LOGIN request with the header 'Auth-State: continue'. This LOGIN request MUST contain the SASL-Mechanism header with the single value of selected SASL machanism.

The LOGIN request MAY contain the PRINCIPAL's authentication information in the body required by the selected mechanism. Details in the case of CRAM-MD5, PLAIN, and EXTERNAL are described in the following subsections.

If the LOGIN request is validated, the server respond with a '200 OK' response. If the same PRINCIPAL is already authenticated by a preceding LOGIN procedure, the server MAY respond with a '409 Already Authenticated'. Otherwise, a '406 Authentication Failed' response SHOULD be returned to the USER AGENT. In this case, the USER AGENT MUST NOT send any other request commands in this connection.

(2-a) CRAM-MD5

The USER AGENT calculates the response data using the keyed MD5 algorithm [KEYED-MD5] where the key is the shared pass phrase and the text is the challenge data. Then, it sends the hexadecimal string of the response octets prepended by the user name and CRLF in the body of the LOGIN request.

[Example]

```
LOGIN PP/1.0 0225 55
From: pres:suga@prim.fujitsu.com
Auth-State: continue
SASL-Mech: CRAM-MD5
Content-Type: text/plain

suga@prim.fujitsu.com
106d12b16fc323dc2f3d19b587f8d0ff
```

(2-b) PLAIN

The PLAIN mechanism is intended to be used only on a fully secured connection, such as one already encrypted using a prior STARTTLS command. In this case, the body part of the LOGIN request contains the user name and the pass phrase in a plain text separated by CRLF.

[Example]

```
LOGIN PP/1.0 84505230 32
From: pres:suga@prim.fujitsu.com
Auth-State: continue
SASL-Mech: PLAIN
Content-Type: text/plain

suga@prim.fujitsu.com
hi there!
```

(2-c) EXTERNAL

The EXTERNAL mechanism is intended to be used in the case that the PRINCIPAL has been already authenticated with some external authentication method, such as TLS client authentication. The LOGIN command using this mechanism contains nothing in the body.

13.2. Server-Server Authentication

When a server establishes a connection to another server, that connection end-point can be authorized to communicate on behalf of multiple PRESENTITIES or INBOXES. This authorization can take place either at connection time, or throughout the duration of the connection.

If the connection uses TLS, then the domains served by each end-point are established in the beginning, through the certificates provided.

If the connection does not use TLS, then each end-point will establish that a domain is being served by the other end-point when the first request pertaining to that domain is received. This can happen more than once per connection.

For example, if server A receives a subscription request from server B, on behalf of user thanos@networkprojects.com, server A MUST verify that server B is one of the servers of the networkprojects.com domain. If so, it will then accept other requests from server B that pertain to users of the networkprojects.com domain.

Verification that a given server is responsible for a given domain is done by performing a name resolution (as described in section 7.2). It is possible that due to DNS limitations, in the case of a domain with a large number of servers, only partial DNS records are advertised. Thus, the address of the server initiating the connection may not be in the records received. In this case a VERIFYSERVER method is performed to establish whether the initiating server has authority over the corresponding domain. This is described in Section 10.5.

14. Privacy Management

14.1. Presence Publication Control

When a USER AGENT publishes a PRESENCE TUPLE, it issues a PUBLISH request to the server. It MUST contain a Class header which specifies the class of WATCHERS that will receive that PRESENCE TUPLE. The server retrieves the relevant WATCHERS from the Class Table when it receives the PUBLISH request.

14.1.1. Class Table

Class Tables are stored at the server and manipulated by the USER AGENTS through the GETCLASSTABLE and SETCLASSTABLE commands. A SETCLASSTABLE request and GETCLASSTABLE response contain an XML encoded Class Table in its body. The DTD of the Class Table is defined as follows:

```
<!ELEMENT CLASSTABLE (class)*>
<!ELEMENT class (watcher)*>
<!ATTLIST class name CDATA #REQUIRED>
<!ELEMENT watcher (#PCDATA)>
```

The matching of WATCHERS to entries in the Class Table goes from

14.1.2. Example

As an example, consider the following Class Table document for bob@workdomain.com:

```
<classtable>
  <class name="important_people">
```

```
          <watcher>wife@example.com</watcher>
          <watcher>@workdomain.com</watcher>
        </class>
        <class name="not_so_important_people">
          <watcher>friend@otherexample.com</watcher>
          <watcher>uncle@otherdomain.com</watcher>
          <watcher>slacker@workdomain.com</watcher>
        </class>
        <class name="everyone">
          <watcher>.</watcher>
        </class>
      </classtable>
```

Every publication of a PRESENCE TUPLE for "important_people" will only be distributed to "wife@example.com" and all WATCHERS from

"workdomain.com" (except "slacker@workdomain.com").

Every publication of a PRESENCE TUPLE for "not_so_important_people" will only be distributed to "friend@otherexample.com", "uncle@otherdomain.com" and "slacker@workdomain.com".

Finally, every publication of a PRESENCE TUPLE for "everyone" will go to all WATCHERS that their identifier doesn’t match any of the above classes.

14.2.   Access Control

14.2.1. Overview

There are two kinds of protected resources: PRESENTITIES and INSTANT INBOXES. Certain requests pertaining to those resources are subject to access control and may only be invoked on behalf of a restricted set of PRINCIPALS. Thus each protected resource has an access control list.

In order to decide if a particular request for an INBOX or a PRESENTITY is permitted, an access control list is used. Abstractly, the decision depends on three parameters, the originator of the request, the operation requested, and the parameters of the operation. The originator of the request is identified in the "From" field of a request. There are several operations that pertain to PRESENTITY ACLs, some that pertain to INSTANT INBOX ACLs, and some that pertain to both.

Access control decisions on behalf of the PRINCIPALS are made at their home servers. The USER AGENT sets the access control list on the Presence and Instant Messaging servers.

More specific ACL entries are evaluated before less specific ones:

- o If the ACL object has an entry with the requestor’s identifier as a key, the value of that entry is the list of permitted operations.

- o Next, if the ACL object has an entry with domain name of the requestor’s identifier, the value of that entry is the list of permitted operations.

- o Next, if there is an entry with the key ".", the value of that entry is the list of permitted operations.

- o Finally, no operations are permitted if there is no corresponding key.

In representing an access control list as an ACL object, each key is either a PRIM identifier, a Domain identifier (preceded by "@") or "." for everybody. The value associated with a key is a set of elements representing permitted operations.

Note that the Server may distinguish if each ACL object is referencing a PRIM identifier, a domain or "everybody" by comparing the first character with "@" and ".".

SUBSCRIBE, FETCH, PUBLISH and REMOVE give the target the right to perform that operation to the ACL owner’s PRESENCE INFORMATION.

SEND, LISTEN and SILENCE give the target the right to perform that operation to the ACL owner’s INSTANT INBOX.

14.2.2. PRESENTITY ACL

The ACL for a PRESENTITY conforms to the following DTD. This is used by the GETACL and SETACL commands, described in Sections 10.6.1 and 10.6.2.

```
<!ELEMENT ACL (entry)*>
<!ELEMENT entry (target, allow)>
<!ELEMENT target (address+)>
<!ELEMENT address #PCDATA>
<!ELEMENT allow (fetch | subscribe | publish | remove)*>

<!ELEMENT fetch EMPTY>
```

```
              <!ELEMENT subscribe EMPTY>
              <!ELEMENT publish EMPTY>
              <!ELEMENT remove EMPTY>
```

14.2.3. INSTANT INBOX ACL

The ACL for an INSTANT INBOX conforms to the following DTD. This is used by the GETACL and SETACL commands, described in Section 10.6.1 and 10.6.2.

```
              <!ELEMENT ACL (entry)*>
              <!ELEMENT entry (target, allow)>
              <!ELEMENT target (address+)>
              <!ELEMENT address #PCDATA>
              <!ELEMENT allow (send | listen | silence)*>

              <!ELEMENT send EMPTY>
              <!ELEMENT listen EMPTY>
              <!ELEMENT silence EMPTY>
```

14.2.4. Example

A sample access control list represented as an ACL object follows. In this example:

- The user permits all operations from "secretary@mycompany.com". Note that she can even PUBLISH my PRESENCE INFORMATION.

- Forbids all operations from any user at "badguys.com", except "goodfriend@badguys.com" who is permitted to FETCH and SUBSCRIBE.

- Permits FETCH and SUBSCRIBE from all users from "@mycompany.com".

- Allows FETCH operations from all other users.

```
            <ACL>

              <entry>
                <target>
                  <address>secretary@mycompany.com</address>
                </target>
                <allow>
                  <FETCH/>
                  <SUBSCRIBE/>
                  <PUBLISH/>
                  <REMOVE/>
                </allow>
              </entry>


              <entry>
                <target>
                  <address>@mycompany.com</address>
                  <address>goodfriend@badguys.com</address>
                </target>
                <allow>
                  <FETCH/>
                  <SUBSCRIBE/>
                </allow>
              </entry>

              <entry>
                <target>
                  <address>@badguys.com</address>
                </target>
                <allow>
                </allow>
              </entry>

              <entry>
                <target>
                  <address>.</address>
                </target>
                <allow>
                  <FETCH/>
                </allow>
              </entry>

            </ACL>
```

15.   Presence Information Data Format (PIDF)

[Note: This section should be rewritten in order to conform to CPIM when the presence format of CPIM is defined.]

15.1.   General Design

Presence information in PRIM is encoded as a MIME-encapsulated XML object. XML is adopted for a base format because of its expressiveness for structured data and its broad acceptance on the Internet. MIME is used to wrap presence XML objects to suit with the [RFC822] style command syntax of PRIM. MIME would also be preferable when MIME based security mechanism (S/MIME or PGP/MIME) is needed for end-to-end security.

We have designed PIDF so that the XML part can be transfered end-to-end without requiring any interpretation or modification of the contents at the Home Servers and any relaying servers. Furthermore, as we assume a case that different devices (e.g. PC, PDA, and cell phone) can update parts of PRESENCE INFORMATION of a single PRESENTITY, PIDF is designed so that the presence servers can handle each part independently of others. This is important to solve the race-condition induced by this use case.

The smallest unit of transporting PRESENCE INFORMATION is a PRESENCE TUPLE. Thus, a tuple is a single XML document encapsulated by MIME. The MIME type for a tuple is "application/presence". In a case that multiple tuples are to be transferred in a single PRIM command, those tuples are combined in a single MIME object by a MIME multipart mechanim. The MIME type used for such a composite object is "multipart/mixed".

15.2. Required Headers for PIDF

All PRIM commands transporting PIDF formatted presence documents MUST have a Content-Type header, whose value is the MIME type of the presence document.

A tuple, a MIME object whose MIME type is "application/presence", MUST have a non-MIME-related header: tuple-id-header. The tuple-id-header header is used to identify the TUPLE by the USER AGENTs and the presence server storing the PRESENCE INFORMATION.

Following the MIME specification, the topmost MIME entity MUST have the MIME-Version header.

The Content-Transfer-Encoding header MUST NOT appear in any PRIM commands. Other MIME related headers, such as Content-ID or Content-Description, MAY be appear in a presence document, but they MUST NOT affect the server behavior at all.

15.3. XML Format Definition

The XML part of a presence document MUST be a well-formed XML document [XML]. The presence XML language is defined as a DTD in section 15.7. However, clients and servers are not required to validate the presence documents according to that DTD.

A presence document SHOULD NOT include any processor instructions or CDATA sections. Client implementations MAY discard them silently.

This is because PRIM assumes the existence of resource-limited terminals, that may have only a very simple parser.

Elements not defined in this document MAY appear in a presence document. But, such elements do not have any particular meaning, and SHOULD be silently discarded by clients.

15.4. XML tags and attributes definitions

15.4.1. The <presence> element

The root element of the presence document is defined as <presence>. This element contains one optional <presentity> element and zero or more <contact> elements.

The root element has two attributes as defined below:

- o address: this attributes indicates the unique identifier of the PRESENTITY publishing this presence document.

- o date: this attribute contains an indication of the date and time when this presence document was published. The format for expressing date and time is defined in 15.5.

15.4.2. The <presentity> element

The <presentity> element contains one <display-name> element, an optional <note> element, and an optional <other-markup> element.

The intention of this element is to contain information not related to any <contact> information but related to the PRESENTITY itself. Examples for such information may include the user’s mental state (contained in <note>) or some profile of the user like vcard info (maybe contained in <other-markup>).

15.4.3. The <contact> element

The <contact> element contains the PRESENCE INFORMATION related to one PRESENCE TUPLE. It MUST contain <display-name> and <status> elements, and MAY be followed by <preference>, <note> and <other-markup> elements.

The <contact> element also MUST contain one "address" attribute, that contains the URL form [RFC1738] of the communication address.

15.4.4. The <status> element

The <status> element has no children elements, but it has the "value" attribute containing the status value. In order to simplify client implementations, the same values will be used for all types of contact addresses (IM, e-mail, phone, ...).

The value of the "value" attribute MUST be one of the following strings:

```
     open
     closed
```

"open" implies that the contact address is in normal working order and messages will be read in a timely fashion. "closed" implies that the contact address will temporarily not work.

The <status> element MAY have a optional attribute "details", which describes the detailed information about status. Currently, the only value specified for this attribute is "deferred", which implies that a greater than normal delay will be experienced before the principal receives communications sent to the contact address.

15.4.5. The <display-name> element

The <display-name> element contains a string for the purpose of the display by the USER AGENT.

15.4.6. The <note> element

The <note> element of the presence document contains some comments related to one PRESENTITY or address. This element SHOULD only contain text, without any tags.

Note that, in future versions, the specification may support XML namespaces, what would allow more advanced formatting of these notes very easily (e.g. by including some XHTML markup)

The content of the note SHOULD be sufficiently small to be rendered in several user interfaces, including a small GSM screen, a "tooltip" on a PC screen and other space-limited situations.

15.4.7. The <preference> element

The <preference> element is an empty element which has only the "value" attribute. The value of this attribute is an unsigned integer, represented in decimal, specifying the preference of a contact relative to the other contacts. Preference values MUST be less than 2^32. A lower preference value indicates a more desirable contact address. A contact without a preference value is less desirable than any contact address with a preference value.

15.5. Date Format

As the format for expressing "date", PRIM adopts the syntax from [RFC822] section 5 as amended by [RFC1123] section 5.2.14. Time zones MUST be expressed numerically. For reference, the ABNF resulting from these references and restrictions is (note that ABNF strings are case-insensitive, so ASCII case-folding MUST be performed when matching day and month strings):

```
date-time = [day ","] date time
      day = "Mon" / "Tue" / "Wed" / "Thu" /  "Fri"
          / "Sat" / "Sun"
     date = 1*2DIGIT month 4DIGIT
    month = "Jan" / "Feb" / "Mar" / "Apr" / "May"
          / "Jun" / "Jul" / "Aug" / "Sep" / "Oct"
          / "Nov" / "Dec"
     time = hour zone
     hour = 2DIGIT ":" 2DIGIT [":" 2DIGIT]
     zone = ("+" / "-") 4DIGIT ; hours+min (HHMM)
```

```
Example: date = "Thu, 30 Mar 2000 09:59:34 -0500"
```

15.6. Examples

The following example is a simple, but complete PRESENCE INFORMATION record that could be sent from the Presence Service to a WATCHER, including INSTANT MESSAGING and telephone addresses.

```
Content-Length: xxx
Content-Type: multipart/mixed; boundary="0123456789"
MIME-Version: 1.0

--0123456789
Content-Type: application/presence
Tuple-ID: im:foo@bar.com

<presence identifier="pres:foo@bar.com"
          date="Thu, 5 Sep 2000 15:45:45 -0500">
   <presentity>
      <display-name>FOO (^^)v</display-name>
      <note>I’m Hungry.</note>
   </presentity>
   <contact address="im:foo@bar.com">
      <display-name>IM</display-name>
      <status value="open"/>
      <preference value="1"/>
   </contact>
```

```
      </presence>

      --0123456789
      Content-Type: application/presence
      Tuple-ID: tel:+1-800-IGOTYOU

      <presence identifier="pres:foo@bar.com"
                date="Thu, 5 Sep 2000 16:30:28 -0500">
         <contact address="tel:+1-800-IGOTYOU">
            <display-name>Phone(Office)</display-name>
            <status value="open" detail="deferred"/>
         </contact>
      </presence>

      --0123456789
```

15.7.   Presence Document DTD

```
      <!ENTITY % URI "CDATA">
      <!ENTITY % statuses "open|closed">
      <!-- The entity "details" is defined as CDATA for
           extensibility. But, the only distiuguished value
           "deferred" has a meaning at present. -->
      <!ENTITY % details "CDATA">

      <!-- the presence tag is the top-level tag for presence. -->
      <!ELEMENT presence (presentity?,contact)>
      <!ATTLIST presence identifier %URI; #REQUIRED>
      <!ATTLIST presence date CDATA #REQUIRED>

      <!ELEMENT presentity (display-name, note?, other-markup?)>

      <!ELEMENT display-name (#PCDATA)>

      <!ELEMENT note (#PCDATA)>

      <!ELEMENT other-markup (#PCDATA)>

      <!-- the contact tag is for each tuple. -->
      <!ELEMENT contact (display-name, status,
                         preference?, note?,
                         other-markup?)>

      <!ATTLIST contact address %URI; #IMPLIED>

      <!ELEMENT status EMPTY>
      <!ATTLIST status value (%statuses;) #REQUIRED>
      <!ATTLIST status detail %details; #IMPLIED>
      <!ELEMENT preference EMPTY>
      <!ATTLIST preference value CDATA #REQUIRED>
```

16. IM Format

INSTANT MESSAGES are opaque payloads transferred by SEND commands tagged by a MIME [MIME] content type.

A SEND command MUST contain a Content-Type header which specifies the content type of the payload. It MAY contain any proper MIME header which may not be defined here.

For the CPIM conformance, A USER AGENT MUST understand and generate messages of the content type ’message/cpim’[CPIM-MSG]. In particular, a USER AGENT MUST generate an INSTANT MESSAGE of the type ’message/cpim’ if it sends the message to other domains which do not or may not understand PRIM. The correspondence between the PRIM and CPIM message format is described in Section 17.

The PRIM servers MUST forward the message as is when the message is relayed to the clients or other servers. That is, the servers MUST NOT delete or modify any header which appears in the command.

17. CPIM/PRIM Mapping

17.1. Presence Protocol

CPIM defines a slightly different subscription model as a common protocol framework for the presence service. In CPIM, a "subscribe" operation will invoke a response reporting only the result of the operation and another immediate "notify" operation conveying the PRESENCE INFORMATION, while the response of the SUBSCRIBE command in PRIM contains the PRESENCE INFORMATION at the same time.

A CPIM gateway SHOULD be implemented to absorb this difference when the "subscribe" operation is gatewayed.

[[Note: More investigation needed. Another choice would be to modify the PRIM spec to align the CPIM framework.]]

When the CPIM presence format is finished, the mapping between the elements of PRIM presence format and those of CPIM will be described.

17.2. Instant Messaging Protocol

The CPIM message format document [CPIM-MSG] defines the common format for INSTANT MESSAGES for the sake of interoperability and the capability to achieve end-to-end security.

A gateway to other domains which does not or may not understand PRIM MUST send INSTANT MESSAGES in the CPIM message format of the content type 'message/cpim'. The gateway MUST convert the SEND request to the corresponding request to the "message" operation [CPIM] of the protocol on the other side. The incoming response MUST be converted to the corresponding PRIM response message. [Details will be described in the next revision.]

If a USER AGENT send a signed or encrypted INSTANT MESSAGES, it MUST compose them in the CPIM message format.

For mapping the PRIM command format to the CPIM message format, the following rules SHOULD be applied. [Note: Obviously, more investigation is needed.]

- o To, From, and Date headers in a PRIM command are copied to the message header part in the CPIM message. [Reply-to, too?]

- o Message-ID and Conversation-ID headers in a PRIM command is converted using the CPIM namespace mechanism and moved to the message header part in the CPIM message as follows;

```
NS: PRIM <http://www.fujitsulabs.com/prim/>
PRIM.Message-ID: [[a unique id for this message]]
PRIM.Conversation-ID: [[a unique id for this conversation]]
```

- o Content-type header in a PRIM command are moved to the content header part in the CPIM message.

18. Security Considerations

There exists many kind of security threats in the Presence / Instant Messaging services and applications as described in the IMPP Requirements [RFC 1778] and the CPIM document [CPIM].

PRIM specifies mechanisms to achieve connection security and to realize access control including presence publication control.

The future PRIM specifications will conform to the expected CPIM data formats for secure and interoperable Presence information and IM exchanges [CPIM,CPIM-MSG]. It will acquire the message level security such as end-to-end confidentiality and integrity.

19. Appendix A: Response Codes

The policy for assigning response codes follows the convention used in HTTP/1.1 [HTTP1.1].

- 1xx: Informational - Request received, continuing process

- 2xx: Success - The action was successfully received, understood, and accepted

- 3xx: Redirection - Further action must be taken in order to complete the request

- 4xx: Request Error - The request contains bad syntax or cannot be fulfilled

- 5xx: Server Error - The server failed to fulfill an apparently valid request

100 Authentication Continued

The request for authentication has been accepted and the authentication process is continued.

101 Unknown Delivery Status

The server was unable to determine that the message was successfully delivered to an INSTANT INBOX or that the transmission failed. This could be because the message was delivered on a best-effort basis, or it was delivered to an "offline" message store.

200 OK

Everything is dandy!

201 Duration Adjusted

The SUBSRIPTION was placed successfully, yet its duration was not acceptable to the server. A new duration was set and this was indicated in the duration-header of the response.

300 Redirect

The server was unable to deal with the request and instructs the caller to reconnect to a different server and reissue the operation there.

400 Bad Request

The request could not be understood by the server due to malformed syntax of the headers or malformed content. The client SHOULD NOT repeat the request without modifications.

401 Unauthorized

The request requires user authentication. The client MUST authenticate itself through the LOGIN request.

402 Forbidden

The server understood the request, but it has not been authorized.

403 Resource Not Found

The specified resource was not found at the server.

404 Subscription Not Found

The SUBSCRIPTION specified in the Subscription-ID header was not found at the resource. This status code MAY appear in the response to the UNSUBSCRIBE and CANCELSUBSCRIPTION requests, and the SUBSCRIBE request in "renew" mode.

406 Authentication Failed

The authentication process has failed. The reason for it is one of the following:

o The authentication process using the specified SASL-Mechanism failed.

o The LOGIN request specifies an inappropriate SASL Mechanism.

o In the midst of the authentication process, the client tries to start another authentication process by specifying ’Auth-State: init’.

407 Timeout

The server timed-out after waiting for a response from another client or server.

408 Inbox Is Closed

The INSTANT INBOX is not currently accepting messages.

409 Already Authenticated

The connection was authenticated previously through a LOGIN command.

410 Astrength Too Weak

The command was rejected because the server requires a higher level of security and this could not be provided.

500 Internal Server Error

The request has not been fulfilled because of the error internal to the server.

501 Not Implemented

The server does not support the functionality required to fulfill the request.

503 Version Not Supported

The server or client does not support the specified protocol version used for the request.

505 Too Many Subscriptions

The SUBSCRIBE request has not been fulfilled because the request exceeds the specified maximum number of SUBSCRIPTIONS at the resource. When this status code is received, the client SHOULD NOT retry the SUBSCRIPTION immediately.

20. References

[CPIM] D. Crocker et al., "A Common Profile for Instant Messaging (CPIM)", draft-ietf-impp-cpim-01.txt, Work in Progress.

[CPIM-MSG] D. Atkins and G. Klyne, "Common Presence and Instant Messaging Message Format", draft-ietf-impp-cpim-msgfmt-00.txt, Work in Progress.

[CRAM-MD5] J.Klensin, R.Catoe and P. Krumviede, "IMAP/POP AUTHorize Extension for Simple Challenge/Response", RFC 2195, September 997.

[HTTP1.1] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999

[MIME] Multipurpose Internet Mail Extensions. See RFC 822, RFC 2045, RFC 2046, RFC 2047, RFC 2048, and RFC 2049.

[OpenPGP] J. Callas, etc., "OpenPGP Message Format", RFC2440, November 1998.

[RFC822] Crocker, D., "Standard for the format of ARPA Internet text messages", RFC 822, STD 11, Aug 1982.

[RFC1123] R. Braden, "Requirements for Internet Hosts -- Application and Support", RFC 1123, October 1989

[RFC1738] T. Berners-Lee, L. Masinter, M. McCahill, "Uniform Resource Locators", RFC 1738, December 1994.

[RFC2778] M. Day, J. Rosenberg, H. Sugano, "A Model for Presence and Instant Messaging", RFC 2778, February 2000.

[RFC2779] M.Day, S.Aggarwal, G.Mohr, and J.Vincent, "Instant Messaging / Presence Protocol Requirements", RFC 2779, February 2000.

[SASL] J. Myers, "Simple Authentication and Security Layer (SASL)", RFC2222, October 1997.

[SASL-PLAIN] C. Newman, "Using TLS with IMAP, POP3 and ACAP", RFC2595, June 1999.

[SMIME] P. Hoffman, Ed, "S/MIME Version 3 Message Specification", RFC2633, June 1999.

[TLS] T.Dierks, and C. Allen, "The TLS Protocol Version 1.0", RFC2246, January 1999.

[URI] T. Berners-Lee, R. Fielding, L. Masinter, "Uniform Resource Identifiers (URI): Generic Syntax", RFC2396, August 1998.

[XML] Extensible Mark Up Language. A W3C recommendation. See http://www.w3.org/TR/1998/REC-xml-19980210 for the 10 February 1998 version.

21. Acknowledgements

The authors greatly appreciate helpful comments from John Stracke and Harald Alvestrand.

This document is based upon several Internet Drafts submitted to the IMPP-WG:

IMTP/PITP:	G. Hudson, MIT
OneIM:	A. Diacakis, F. Mazzoldi, Network Projects, Inc.
PePP:	H. Sugano et al, Fujitsu
SIMP:	J. Ramsdell, MITRE Corporation

22. Author’s Addresses

F. Mazzoldi
flo@networkprojects.com
Network Projects, Inc.
4516 Henry Street, Suite 113
Pittsburgh PA 15213
USA

A. Diacakis
thanos@networkprojects.com
Network Projects, Inc.
4516 Henry Street, Suite 113
Pittsburgh, PA 15213
USA

S. Fujimoto
shingo@fla.fujitsu.com
Fujitsu Laboratories of America, Inc.
595 Lawrence Expressway
Sunnyvale, CA 94085
USA

G. Hudson
ghudson@mit.edu
Massachusetts Institue of Technology
Cambridge, MA 02139
USA

J. D. Ramsdell
ramsdell@mitre.org
The MITRE Corporation
202 Burlington Road
Bedford, MA 01730-1420
USA

H. Sugano
suga@flab.fujitsu.co.jp
Fujitsu Laboratories Ltd.
64, Nishiwaki
Ohkubo-cho
Akashi 674

Japan

23. Full Copyright Statement

Acknowledgement

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