INTERNET-DRAFT
draft-houri-sametime-community-client-00.txt
Expires February 28, 2000

Avshalom Houri
Ubique/Lotus
Ittai Golde
Ubique/Lotus

Sametime (TM) Community - Client Protocol

draft-houri-sametime-community-client-00.txt

This document and related documents are discussed on the impp mailing list. To join the list, send mail to impp-request@iastate.edu.
To contribute to the discussion, send mail to impp@iastate.edu.
The archives are at
http://lists.fsck.com/cgi-bin/wilma/pip.

The IMPP working group charter, including the current list of group documents, can be found at:
http://www.ietf.org/html.charters/impp-charter.html.

1. Table of Contents

2. Abstract

This document describes the protocol used by a client in a Sametime (TM) community. The protocol enables users connecting to a Sametime community to be aware each other and to send Instant Messages (IMs) to other users in the community.

While this protocol does not meet many of the requirements of the IMPP working group, it is provided as background information on existing Instant Messaging and Presence implementations. This protocol is provided 'as is' without warranty of any kind. In particular, commercial use of this protocol requires licensing.

The protocol described in this document should enable implementing a client program that can interoperate with Sametime communities, given the user has registered with the community.

3. Introduction

3.1. Background

The Sametime community architecture was designed and implemented by Ubique as part of the Sametime architecture of Lotus.

Ubique has been dealing with awareness since 1990, when it produced Virtual Places (tm). Virtual Places enable users viewing a web page to be aware of other users viewing the same page. In addition to Lotus Sametime Connect, Ubique produced an earlier buddy list client that is in use at well-known portals and community sites. At the time of writing, the largest such installation supports ???? users.

The largest number of concurrent users of a community running Ubique's software is 50K.

3.2. Terms & Definitions

- Community - A set of servers and service providers supplying presence management to a group of users that are authenticated by the same authentication authority.

- User - A person getting services from a community.

- Client - A program used by a user for interacting with the community.

- Service - A functionality supplied by the Community.

- Server - A community component to which the client connects for getting community services.

- Home Server - The server defined as the default server of the user. The persistent storage of a community defines the home server for each user.

- Service Provider - A community component providing services to the

community.

- Login - An active connection of a user or other community component to a server.

- Presence - A set of properties/attributes of a user that are managed by the community.

3.3. Sametime Community Architecture

A Sametime user can have multiple logins to the community. Each login is done via a different TCP connection. The community synchronizes between the various logins of the User to maintain a consistent presence. For example, when a user updates its details through one of her/his logins, the community reports to the other logins about the change.

The user can inform the community how that user's visibility should be limited. The user informs the system either by supplying a list of others to whom the user's presence should not be published, or by supplying a list of others who are the only ones to whom the user's presence should be published. This feature is called Privacy. Privacy is also synchronized between the multiple logins of the user if they exist.

Multiple logins are needed when the user runs two or more Sametime applications and the applications are not able (or were not written to) share the same TCP connection. We call these type of applications non-cooperative applications. The ability to have different logins enables using such applications without letting the user feel that they are using different logins.

Each login in a Sametime community can use many services in the community. Each service may be used by a different application that is run by the user. Upon connecting to the community, the login connects with the basic service of the community - the Community Service. Using the community service, it is possible to locate other services in the community and connect with them.

Traffic on a TCP connection is divided into channels. A channel is a virtual connection between two entities (login, server or service provider). Upon connecting to the community a default master channel is created between the client and the server. Using the master channel, other channels can be created. When a client connects to some service in the community or interacts with a login of another user, a channel is created for the interaction.

A channel may be encrypted. Upon creation of the channel, a method of
encryption is selected by its two endpoints. After selecting an encryption method, messages passing on the channel may be encrypted. This document does not describe how an encrypted channel is created and what encryption methods are supported.

This document does not cover the architecture of the server side of the Sametime community. In general, we can say that a Sametime community is

composed of multiple servers acting in concert. The Sametime community is scalable and distributed. Scalability means that the number of users that can be served by a community is larger than the number of the users that can be served by a single server. Distribution means that the community is composed of multiple servers, where each server is in a different location.

A user connected to one community may get services from other Sametime communities. This is achieved via a community-to-community connection. When a service from another community is requested, the community of the user can connect to the other community and get services for the user from there. Of course, the ability to get inter-community services is dependent on agreements between the communities.

4. User model

The Sametime user and login model supports multiple logins of the same user to the community. The user model is divided into persistent and runtime parts.

4.1 Persistent User Data

Each user in a Sametime community has the following basic properties. These properties are persistent across server shutdowns.

- Home server - The default server into which the user should usually login.

- User ID - A string representing an identifier of a user in the community. This string is unique among all the users in the community.

- Login Params - Set of fields used for the creation of a login into the community. Common fields will be a login name and a password. The login name is a string that has no further usage beyond the login phase.

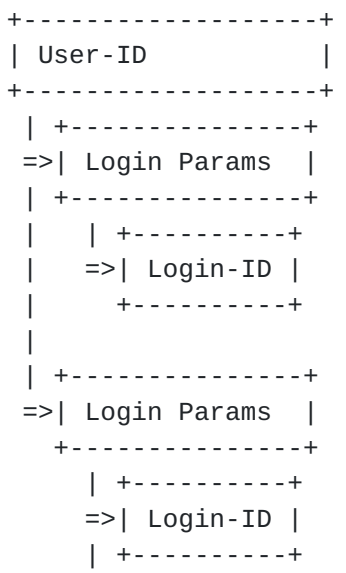
- User-Name - A name under which other users see the user. A user might have different user-names for different logins
(4.2)

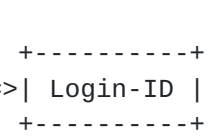
- Description - A descriptive text about the user. Again, a description may be different for different logins.

- Privacy List - A list of user IDs that may or may not know that the user is online. A privacy list is maintained per user. If a user logs into the community several times simultaneously, the server synchronizes the privacy list of the user between its different logins.

4.2 Runtime User Structures

The runtime user model of Sametime enables each user to create multiple concurrent logins to the community. As shown in the example below, a single User-ID may have multiple Login Params, each of which may have multiple Login-IDs.





Each TCP/IP connection of a user creates a single login of the user to the community. A login ID that is unique throughout the community is assigned to each login.

Sametime applications may be written by different vendors, or may be written by the same vendor using different programming environments (e.g. Java and OCX). Therefore it is not possible in all cases that the various Sametime applications run by the user will share the same TCP connection to the community.

For this reason a Sametime community supports multiple logins of a user. Multiple logins enable the various Sametime applications run by the user to be independent on each other. Even though there are multiple logins of the user, the Sametime community synchronizes various updates (e.g. privacy) between the logins.

In current implementations of the Sametime community, it is not possible to create different simultaneous logins from different user machines. The restriction is enforced by the Sametime community by logging out an "old" login.

5. Protocol Elements

This section defines the elements used in the Sametime protocol.

5.1. Community

A string uniquely identifying a Sametime community. Usually this string will be a domain name or some other identifier that is guaranteed to be unique.

5.2. User-ID

A string uniquely identifying a user in a community. user-ID is saved

in a persistent storage and is not affected by server restarts. At runtime the user-id is augmented to conatain also the community name.

5.3. Login-ID

Every entity logged into a community has a runtime login-ID. The login-ID is unique over all the Sametime communities. This uniqueness is achieved by having the login-ID be composed of the following:

- A string uniquely identifying the community

- A unique string constructed from the IP address of the server into which the user is logged and a number assigned by the server.

The login-ID is the primary handle for referencing logins in the Community. When a channel to another user in the community is to be created, someone (either the client or some community component) first has to find a login-ID of the user and only then can the channel be created.

5.4. Channels

A Sametime community is composed from multiple servers and service providers - hence, the network path between two community entities (login or community component) may span several hops. Each hop is a TCP connection.

A Channel is a virtual connection between two community entities. The channel spans over the route of network connections that exist between the two sides of the channel. Hence the channel supplies the following functions:

- Defines the routing path between the two sides of the channel.
- Ensures order of messages.
- Supplies notifications to both sides of the channel when some network connection along the path is broken.

Channels are also used for efficient propagation of messages. When there is a need (e.g. in a N-way chat) to send a message to multiple recipients, the sender can send a single instance of the message with the list of recipients channels. Upon receiving the message, the server sends the message along the appropriate network connections where each network connection gets a single copy of the message and a list of channels that are contained in the network connection. This method of message propagation is called: multi-send on channel.

5.5 Master Channel

When a TCP connection from a client to a server is initiated, a default channel named Master-Channel is created. The master-channel serves the following purposes:

- Authentication (Handshake and Login messages)
- Creation of other Channels
- Sending Broadcast messages
- Sending One Time messages

- Sending utility messages
- Sending Privacy and Status messages

5.6. Channel-ID

Channel-ID numbers are unsigned words (32 bits). The numbering of channels is local to the TCP connection between each pair of community entities along the channel path.

The channel creator can be on either side of the TCP connection. In order to avoid using the same ID when creating a channel, the range of possible IDs is divided between the two sides of the TCP connection. The initiator of the TCP connection (the ACTIVE side) is allocated the numbers with the Most Significant Bit (MSB) at zero. The other side (the PASSIVE side) is allocated the numbers with the Most Significant Bit (MSB) at one.

Due to the locality of the channel numbering, channel-IDs can be reused immediately after the channel is closed.

5.7. Service Providers

Clients connected to a Sametime community are served by Service Providers. Each service provider is responsible for a certain functionality. We refer to these functionalities as Services.

The following services are among those offered by Sametime service providers, and are relevant to this protocol documentation:

* Who Is Online - Widely known as the buddy list service. This service supplies users with notifications about status & properties of other users in the community.

* Authenticate - Authenticates users against the database for its server.

* Resolve - Resolves a user-name into zero or more user-IDs

* N-Way Chat - Supplies chat-rooms, where several users can participate.

The Instant Messages (IM) are supplied directly by the Sametime server.

5.8. Encryption

Encryption is supported in the Sametime community. When a channel is created it can be specified as encrypted. The parameters specifying the encryption method to be used are sent in the channel creation phase. So immediately after the channel is created, the channel is either encrypted or not. There are no messages for converting a non-encrypted channel to an encrypted one or vice versa.

It is possible to send a non-encrypted message on an encrypted channel. A flag in the messages indicates whether the message is encrypted or not.

Encryption is also supported in a N-way chat. The channel of each chat participant is encrypted. The encryption method and keys used are local to each participant in the chat. Therefore, each message sent by the chat service provider has to be encrypted per each participant. The current chat service provider encrypts only the text sent by the chat participants. Messages indicating updates to the participant list of the chat are not encrypted.

This document describes only the parameters required for creating a non-encrypted channel.

6. Protocol Overview

The Sametime client protocol can be divided into several phases. Following is a short description of each phase. Subsequent sections will detail state transitions of each phase.

- Login - The user is authenticated with the community and is assigned a login-ID to be used in the login session.

- Creating a channel - A channel is created to another entity in the community.

- Subscribing and getting notifications on presences - Messages are sent to the awareness service, requesting notification on state changes of certain users.

- Changing attributes - The user (actually one of its logins) requests a change in an attribute of its presence.

- Changing privacy list - The user (actually one of its logins) requests a change in the content of its privacy list. Note that change in privacy list is separated from a change in an attribute since the privacy list is stored in a persistent storage in the server and the request for a change may be denied when the change can not be written.

- Initiating and responding to IMs - Creating or accepting a channel for the purpose of interchanging instant messages. This phase include also the messages for the actual IMs

- Resolving User-Names - Since the same user-name can be used by different users, user-names have to be resolved to user-IDs prior to usage. This phase describes how it is done.

- Sensing service - When a service provider can not be located, a client may request to be notified when the service is available. This method frees the client from polling on the availability of the service.

6.1 Login

This is the phase in which the client moves from being merely connected to the server via TCP to being a part of the community.

State Transitions (assumes TCP connection is already established):

Step	Action	Next Step
1.	**Client Sends Handshake**	2
2.	**Server Sends HandshakeAck**	3
3.	**Client Sends Login**	4 or 5 or 6
4.	**Server Sends LoginAck**	Finish
5.	**Server Sends AuthPassed**	1 or 7
6.	**Server Destroys Master Channel**	Finish
7.	**Client Sends LoginCont**	8
8.	**Server Sends HandshakeAck**	4

State Description

1. **The client (after establishing a TCP connection), sends a Handshake** message as described in 8.4.1.1 to the server

2. **The server sends a HandshakeAck message as described in** 8.4.1.2.

3. **The client sends a Login message as described in 8.4.1.3**

4. **If the user is permitted in the community and does not need to be** redirected, the server sends the LoginAck message as described in 8.4.1.4 .

5. **If the user needs to be redirected to another server, the server** sends the client an AuthPassed message as described in 8.4.1.6. The client can then either issue a LoginCont message (Step 7), or disconnect and attempt a direct connection to the other server.

6. **If the user fails to authenticate, the server destroys the Master** Channel with the appropriate error code (for values, see 8.3.1).

7. **The client chooses not to reconnect to the other server. It issues a** LoginCont message notifying the server to go on with the login procedure (though redirected to the other server). LoginCont message is described in 8.4.1.5.

8. **The server sends the client a HandshakeAck message, telling the** client it is being processed by the remote server; the protocol being provided to it by the local server will be the remote one's (see HandshakeAck in 8.4.1.2), and proceeds to step 4.

6.2 Creating a Channel

In this phase the client creates a channel (as described in CreateCnl on 8.4.1.7) to some other community entity.

From the point of view of the client, the entities that participate in a channel creation are the creator and the acceptor. The channel path may traverse additional community components but the server architecture and protocol are beyond the scope of this document.

State Transitions:

Step	Action	Next Step
1.	**The creator sends a CreateCnl** message to the acceptor.	2 or 3
2.	**The acceptor or some other** component sends a DestroyCnl message to the acceptor	Finish
3.	**The acceptor sends a AcceptCnl** message to the creator	Finish

State Description

1. The creator sends a CreateCnl message as described in 8.4.1.7.

2. The CreateCnl message is rejected either by some community component in the route to the acceptor or by the acceptor itself. The CreateCnl message may be rejected by some community components other then the acceptor due to various reasons as permissions or the unavailability of the acceptor.

3. The acceptor is willing to accept the channel, and sends an AcceptCnl back to the creator thus establishing a channel with the creator.

6.3 Subscribing and Getting Notifications on Presences

After a channel between the client and the awareness service provider is established, the client has to supply its awareness list to the awareness service in order to be notified on users in the list.

State Transitions:

Step	Action	Next Step
1.	**The client sends AddWatch with a**	2

	list of user-IDs to the service	
	provider	
2.	**The service provider sends a Snapshot**	**[3]**
	message to the client	
3.	**Upon any change in the state of**	
	of a presence, the service sends	...
	Update message to the client	
4.	**The client sends removeWatch with a**	...
	list of user-IDs	

State Description

1. The client sends AddWatch (as described in 8.4.2.2) to the Awareness service provider. The message contains the list of user-IDs the watcher is interested in their presence status.

2. On receiving the AddWatch message, the service provider synchronizes the client with the current status of the presences with a Snapshot message, as described in 8.4.2.4.

3. When a presence changes state, the Awareness service provider sends an Update Message (as described in 8.4.2.5) to the watcher client.

4. When the watcher stops "watching" the presence, it sends a RemoveWatch message (as described in 8.4.2.3) to the Awareness service provider.

6.4 Changing Privacy List

User privacy list is saved in a database at the server side for future logins. This is the reason the privacy change request might fail (if there's a problem in updating the privacy).

State Transitions:

Step	Action	Next Step
1.	**The client sends a setPrivacyList**	2 or 3
	to the server.	
2.	**If the database update fails, the**	Finish
	server replies with a setPrivacyDenied	
	message.	
3.	**If the database update succeeds, the**	Finish
	server sends a setPrivacyList message	
	to all the logins of the user.	

State Description:

1. Whenever a client wants to change its privacy list the client sends a setPrivacyList message to the server as described in **8.4.1.12. The server in turn updates the database with the** new mode.

2. If the database update fails, the server sends a setPrivacyDenied Message to the login, as described in

3. If the database update succeeds, the server sends setPrivactList message to all the logins of the user, with the new Privacy List.

6.5 Initiating and responding to IMs

IMs are actually messages being sent on a given channel.

State Transitions:

Step	Action	Next Step
1.	**Create an IM channel to the target** client	2 or 3
2.	**Target client destroys channel**	Finish
3.	**Target client accepts channel**	4
4.	**Either clients sends an IM message**	4 or 5
5.	**Either client destroys channel**	Finish

State Description

1. The clients creates an IM channel to the target client, as described in 8.4.1.7. This channel differs from regular channel by the fact that it has additional data (description of the additional data is in 8.4.3).

2. The target client can either destroy the channel as described in 8.4.1.10 , or...

3. The client accepts the channel, as described in 8.4.1.8.

4. Either clients can now send and receive messages on that channel, as described in 8.4.3. Note that there are two types of messages.

5. Either client destroys the channel, as described in 8.4.1.10.

6.6 Resolving User-Names

Since users may have non-unique user-names, it is necessary to resolve user-names to user-IDs which are unique. The client creates a channel

to the resolve service provider and sends a list of user-names in a resolve message (As described in 8.4.4.1). The resolver resolves the names to user-IDs (in a message described in **8.4.4.2). Note that each user-name may be resolved to** several user-IDs.

State Transitions:

It is assumed that the channel to the resolve service provider is already established.

Step	Action	Next Step
1.	**Client sends MultiResolve on** the channel to the resolver	2
2.	**The resolver returns a response** containing a list of answers per each resolved name. Each answer contains the name that was resolved, an error code and the list of matching names.	Finish

6.7 Sensing Services

Services are provided by service providers that connect to the server. When a user logs into a community it might be that not all service providers are present. Therefore as an optimization, instead of polling the server, the login may request to be notified when a certain service is available.

In this phase the login requests its server to be notified when a

certain service provider is active. It is done once per notification, meaning once a notification has been sent to the client about a service, the client has to re-issue the message in order to sense it again.

State Transitions :

Step	Action	Next Step
1.	**Client sends SenseService**	2 or Finish
2.	**[When the service is up] The** server sends SenseService to the client	Finish

State Description:

1. The client sends a SenseService message to the server as described in 8.4.1.14.

2. When (and if) the service being sensed is activated, the server sends the SenseService message back to the client, as described in 8.4.1.14

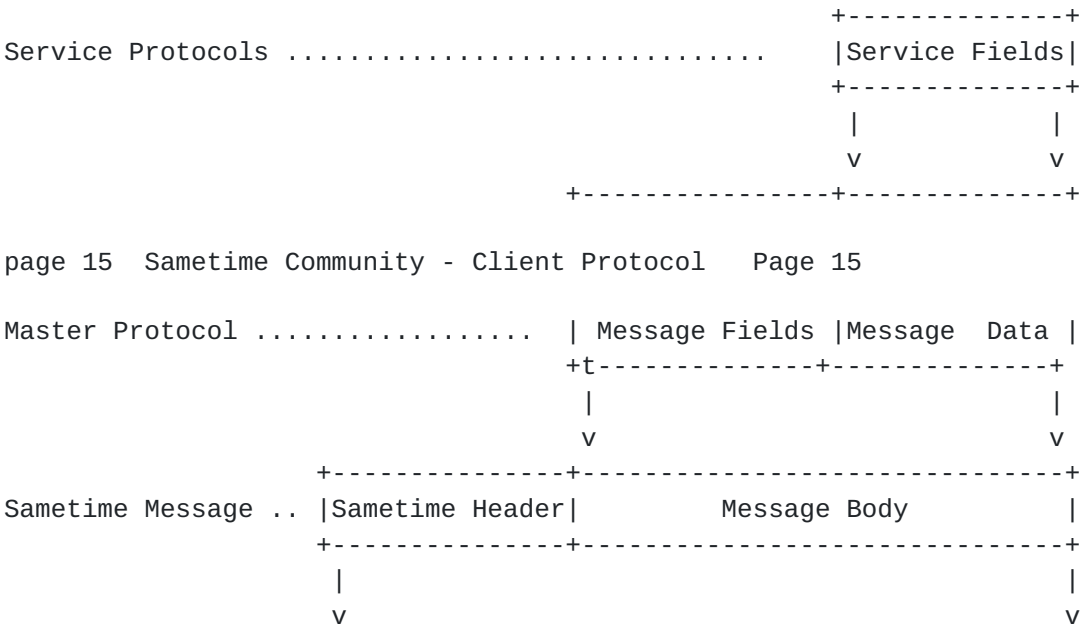
7. The Sametime Message format

Every Sametime message is preceded by an extra single byte. This byte is a counter, which is incremented by one for each message on a connection from 0x81-0xFF and then starting again at 0x81 (MSB is always 1). This extra byte is used as additional assurance of order preservation and reliable transfer of messages on TCP connections. A single byte with the MSB set to 1 and the other bits set to 0 (0x80) is used as a keep-alive byte. A Sametime system may be configured to send a keep-alive byte at regular intervals. This allows the system to close connections that are no longer in use and keep connections open where necessary. When an application receives a leading byte from 0x81-0xFF, it understands that a Sametime message follows. When an entity receives 0x80, it recognizes it as a single keep-alive byte.

7.1 Layering and message encapsulation

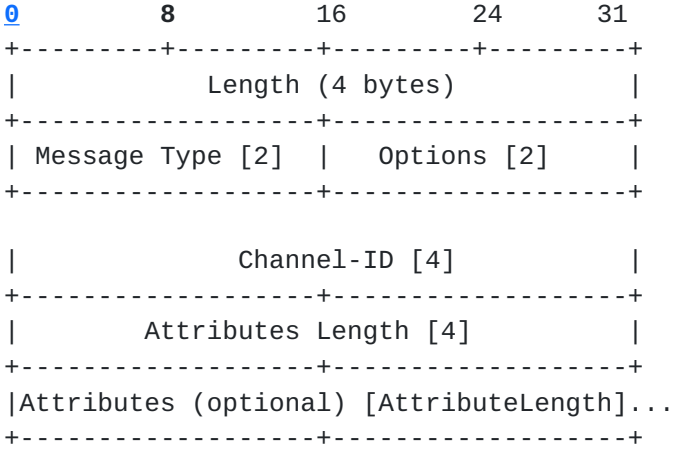
The Sametime communication layer handles the routing of messages between logins in a community. The communication layer is one layer above the TCP layer and it relies on underlying TCP connections.

The following figure illustrates the use of layering and encapsulation in Sametime:



7.1.1 The Sametime Message Header

The message header has the following format:



Fields Description:

Length
  A 4-byte field containing the size of the message in bytes.

Message Type
  A 2-byte field containing the type of the message. This field
  specifies the Master protocol (8) message type to be
  used to interpret the message body.

Options
  A 2-byte field containing options. Various options are specified
  by setting bits to zero or one. The following bits are currently in
  use (LSB is numbered as zero):
    * Bit 0 - If set to 1 indicates the presence of an attribute
      section in the message.
    * Bit 1 - Indicates encryption of the message.

Channel-ID
  A 4-byte field that specifies the channel on which the message is
  sent. Depending on the type of message, the ID represents either a
  master channel or a regular channel.

Attributes Length
  A 4-byte field that specifies the length of the attributes part of
  the message. This field exists only if the Options field indicates

that the message contains attributes.

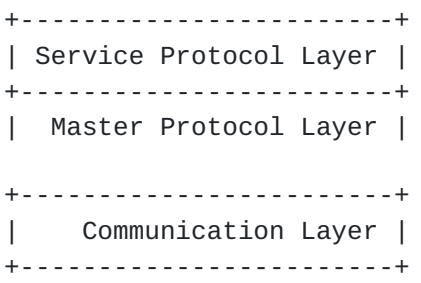
Attributes
Varies in content according to message type. The length of this field is specified by the Attributes Length field.

7.1.2 The Message Body

Message body format is dependent on the message type. See "Master Protocol Reference" (8) for a description of each message type and its body format. A message type may also have subtypes. The message subtype may determine how the message data is interpreted.

7.1.3 Protocol Layers

The Sametime protocol is built on a model of message encapsulation and protocol layering. The following figure shows the order of the Sametime protocol layers in relation to the TCP layer:



- Communication Layer - Responsible for routing messages between community elements through the use of channels on top of TCP. Messages at the Communication layer have a Sametime header and a message body.

Master Protocol Layer - Interprets the data in the message body according to the message type specified in the Sametime header. The Master Protocol is used to interpret all Sametime messages. The following activities are handled at the Master protocol level:
    * Handshake
    * Login and authentication
    * Channel creation and destruction
    * Sending messages on other channels. These messages are not interpreted at the master protocol. They are interpreted by the particular service protocol associated with the channel on which the message was sent.
    * Sending broadcast messages
    * Sending one time messages
    * Sending utility messages (such as, "service up" and "service down")
    * Sending privacy and status messages

- Service Protocol Layer - Interprets the message data that is sent within certain master protocol messages on established channels. The

particular service protocol used is determined for all communication on

that channel when the channel is created.

8. Master Protocol Reference

Each message type is described separately in this section. For each message type, there is a description of its function and a diagram of the message body. All messages have the Sametime header, as described above (see "Sametime Header" on 7.1.1).

Note: Unless explicitly specified, fields do not necessarily start at a word boundry as can be understood from the following diagrams.

8.1 Basic Data Types

The following types are used in field descriptions:

```
Flag (Boolean) - 1 byte
char           - 1 byte
Ulong          - 4 bytes
Ushort         - 2 bytes
Opaque         - A large binary object, with its length in the header,
                 the length is an unsigned long ( = 4 bytes)
String         - A null terminated string, with its length in the
                 header, the length is short (= 2 bytes), The length
                 does not include the null character terminating the
                 string.
```

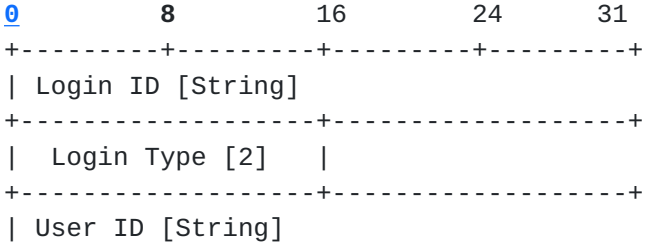
8.2 Common Structures

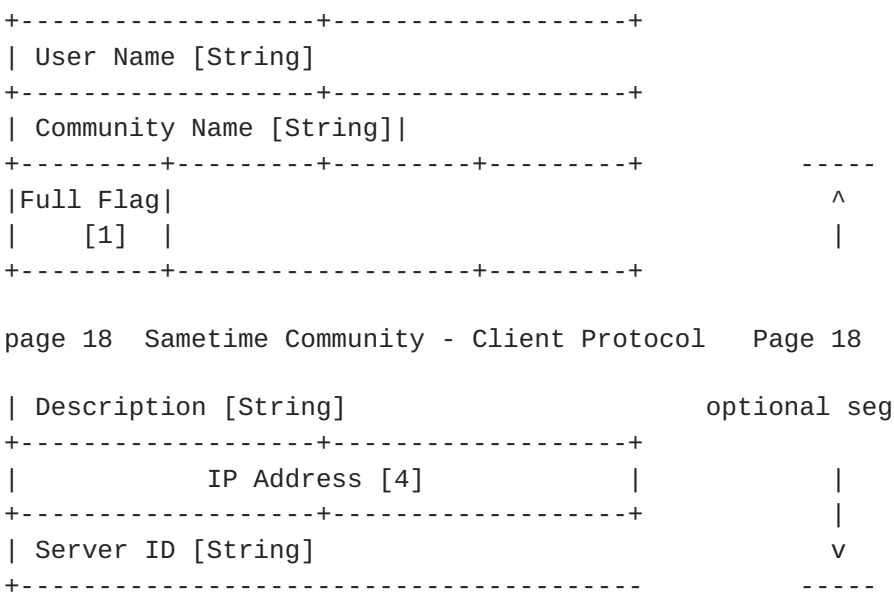
The following blocks of fields appear in several message types. These blocks are described separately for reasons of clarity, conciseness, and improved understanding of their purpose.

8.2.1 Login Info block

The Login Info block appears in messages when a complete description of an entity is passed. The portion of the block following the Full Flag is optionally present depending on the message type.

Message Body





Description of Fields:

Login ID
A string containing the unique (within the community) ID of the login described.

Login Type
A 2-byte field. Specifies the type of Sametime login described. See "User Model" on 8.3.6 for a list of established constants.

User ID
A string containing the unique (within the community) ID of the user.

User Name
A string containing the name of the user. This is the string that appears as the user's name on the screen. It is stored in a database and is looked up according to the login name.

Community-Name
A string containing the name of the community. The community name can be any string, but the Internet domain name is usually used.

Full Flag
A single byte. If the value equals 1, the rest of the Login Info block (Description, IP Address, and Server ID fields) is transmitted. If the value equals 0, the rest of the block is not present.

Description
A string field. The description can be any string data. The current implementation does not place any information in this field.

IP Address

A 4-byte field containing the IP address of the login described.

Server ID
A string containing the unique ID of the login's server (where the login is held).

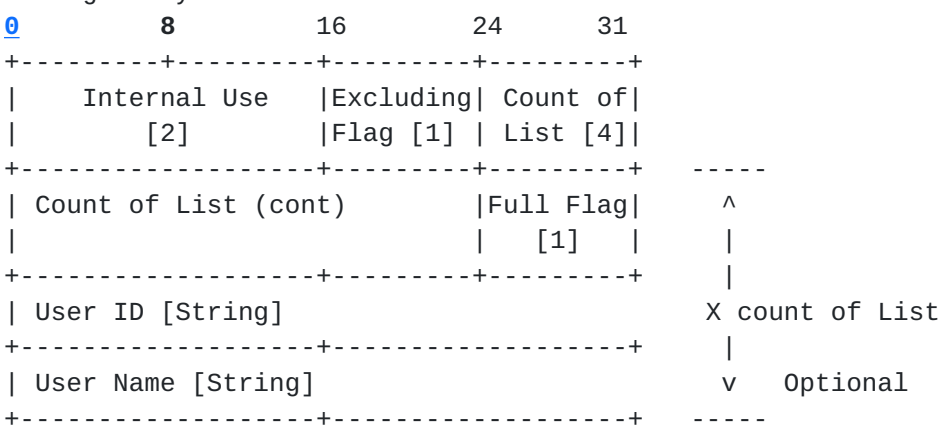
8.2.2 Privacy Info Block

The Privacy Info block consists of a list of user-IDs. The portion following the "Count of List" represents one item (user item) in the user-ID list. This portion is repeated the number of times equal to "Count."

The User Name portion of each User Item is an optional segment, which

appears depending on the Full flag.

Message Body



Description of Fields

Internal Use Byte
Reserved for internal use

Excluding Flag
A single byte. If the value equals 1, the list of users that follows represents the users who are excluded from seeing the user whose privacy is described (all other users are permitted).
If the value equals 0, the list contains the users who may see the user whose privacy is described (all others are excluded).

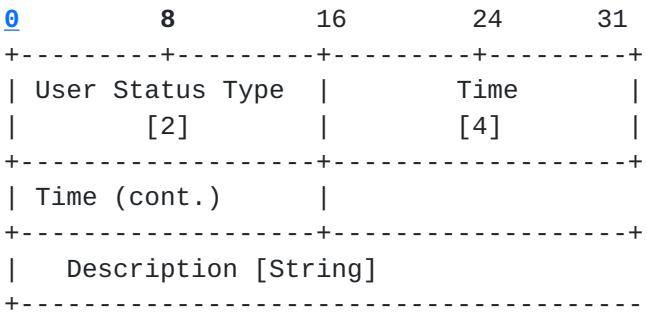
User List
A list of users representing either the users who are excluded from seeing the user whose privacy is described or the users who are permitted to see the user whose privacy is described. The list begins with a 4-byte field specifying the number of items in the list. Each item in the list is a User Item. Each User Item contains

a Full Flag, which indicates whether the item is full or not, a User ID string, and if the item is full, a User Name string.
The User ID is a string containing the unique (within the community) ID of the user. The User Name is a string containing the name of the user, which is the name displayed on the screen in the user interface.

8.2.3 User Status Block

This block contains information about the user's status. It is used in messages that contain user status information.

Message Body



Description of Fields

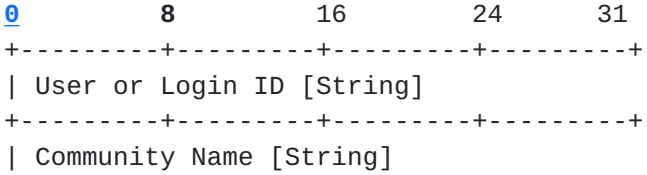
User Status Type
A 2-byte field indicating the status of the user. See 8.3.5 for a list of established constants.

Time
A 4-byte field containing the time of the last change in the user's status in minutes. Not set by Sametime 1.0 and 1.5 servers.

Description
A string containing the message associated with the status type for this user. The user's server saves the message customized by the user for certain status types.

8.2.4 ID Block

This block is used for addressing messages. This combination of two strings uniquely identifies a user or login in a multi-community environment.



```
+---------------------------------------
```

Description of Fields:

Target ID
  A string that can be either a User ID or a Login ID or an empty string.
    * If it is a User ID, the ID block specifies a user. A user may have more than one associated login. The server may decide which login will be the ultimate target.
    * If it is an Login ID, the ID block specifies a single login.
    * If it is an empty string, the message target will be decided on the basis of alternative criteria - service type.

Community Name
  A string specifying the community name. This may be an empty string if the target is on the same community.

8.2.5 Encryption Block

This document does not contain description of how to create encrypted channels. Following are the values that are sent on non-encrypted channel creation.

```
0         8         16        24      31
+---------+---------+---------+---------+
| No-Encryption [Opaque]
+---------+---------+---------+---------+
```


The No-Encryption opaque contains only a short (2-byte) of value zero.

8.3 Constants

8.3.1 Error Codes

8.3.1.1 General error/success codes

operation succeeded --- 0x00000000

operation failed --- 0x80000000

request accepted but will be served later --- 0x00000001

request is invalid due to invalid state or parameters --- 0x80000001

not logged in to community --- 0x80000002

unauthorized to perform an action or access --- 0x80000003

a resource

operation has been aborted --- 0x80000004

the element is non-existent --- 0x80000005

the user is non-existent --- 0x80000006

the data are invalid or corrupted --- 0x80000007

the requested feature is not implemented --- 0x80000008

not enough resources to perform the operation --- 0x8000000A

the requested channel is not supported --- 0x8000000B

the requested channel already exists --- 0x8000000C

the requested service is not supported --- 0x8000000D

the requested protocol is not supported --- 0x8000000E

the requested protocol is not supported --- 0x8000000F

the version is not supported --- 0x80000010

user is invalid or not trusted --- 0x80000011

already initialized --- 0x80000013

not an owner of the requested resource --- 0x80000014

invalid token --- 0x80000015

token has expired --- 0x80000016

token IP mismatch --- 0x80000017

WK port is in use --- 0x80000018

low-level network error occurred --- 0x80000019

no master channel exists --- 0x8000001A

already subscribed to object(s) or event(s) --- 0x8000001B

not subscribed to object(s) or event(s) --- 0x8000001C

encryption is not supported or failed unexpectedly --- 0x8000001D

encryption mechanism has not been initialized yet --- 0x8000001E

the requested encryption level is unacceptably low --- 0x8000001F

the encryption data passed are invalid or or corrupted --- 0x80000020

there is no common encryption method --- 0x80000021

the channel is destroyed after a recommendation is made connect elsewhere --- 0x80000022

the channel has been redirected to another destination --- 0x00000023

8.3.1.2 Connection/disconnection errors

versions don't match --- 0x80000200

not enough resources for connection (buffers) --- 0x80000201

not in use --- 0x80000202

not enough resources for connection (socket id) --- 0x80000203

hardware error occurred --- 0x80000204

network down --- 0x80000205

host down --- 0x80000206

host unreachable --- 0x80000207

TCP/IP protocol error --- 0x80000208

the message is too large --- 0x80000209

proxy error --- 0x8000020A

server is full --- 0x8000020B

server is not responding --- 0x8000020C

cannot connect --- 0x8000020D

user has been removed from the server --- 0x8000020E

VP protocol error --- 0x8000020F

cannot connect because user has been restricted --- 0x80000210

incorrect login --- 0x80000211

encryption mismatch --- 0x80000212

user is unregistered --- 0x80000213

verification service down --- 0x80000214

user has been idle for too long --- 0x80000216

the guest name is currently being used --- 0x80000217

the user is already signed on --- 0x80000218

the user has signed on again --- 0x80000219

the name cannot be used --- 0x8000021A

the registration mode is not supported --- 0x8000021B

user does not have appropriate privilege level --- 0x8000021C

email address must be used --- 0x8000021D

error in DNS --- 0x8000021E

fatal error in DNS --- 0x8000021F

server name not found --- 0x80000220

the connection has been broken --- 0x80000221

an established connection was aborted by the software in the host machine --- 0x80000222

the connection has been refused --- 0x80000223

the connection has been reset --- 0x80000224

the connection has timed out --- 0x80000225

the connection has been closed --- 0x80000226

disconnected due to login in two Sametime servers concurrently --- 0x80000227

maps to 0x80000227 retained for compatibility with --- 0x80000228

disconnected due to login from another computer. --- 0x80000229

unable to log in because you are already logged on from another computer --- 0x8000022A

unable to log in because the server is eother unreachable, or not configured properly. --- 0x8000022B

unable to log in to home Sametime server through the requested server, since your home server needs to be upgraded. --- 0x8000022C

the applet was logged out with this reason. Perform relogin and you will return to the former state. --- 0x8000022D

8.3.1.3 Client error codes

the user is not online --- 0x80002000

the user is in do not disturb mode --- 0x80002001

can not login because already logged in with a different user name (Java only) --- 0x80002002

8.3.1.4 IM error codes

cannot register a reserved type --- 0x80002003

the requested type is already registered --- 0x80002004

the requested type is not registered --- 0x80002005

8.3.1.5 Resolve error codes

the resolve process was not completed, but a partial response is available --- 0x00010000

the name was found, but is not unique (request was for unique only) --- 0x80020000

the name is not resolvable due to its format, for example an Internet email address --- 0x80030000

8.3.2 Service Types

The service type of the buddylist server application --- 0x00000011

The service type of the resolver server application --- 0x00000015

The service type of the IM channel --- 0x00001000

8.3.3 Protocol Types

The protocol type of the buddylist server application --- 0x00000011

The protocol type of the resolver server application --- 0x00000015

The protocol type of the IM channel --- 0x00001000

8.3.4 Version Constants

Major Version value --- 0x001E

Minor Version value --- 0x0018

Buddy List Protocol Version --- 0x00030005

Resolve Protocol Version --- 0x0

8.3.5 User Status Types

User is active --- 0x0020

User is idle --- 0x0040

User is away --- 0x0060

User request not to be desturbed --- 0x0080

8.3.6 Login Types

Login is using a C++ Component (i.e. ActiveX) --- 0x1000

Login is using a Java Applet --- 0x1001

Login is using a binary executable --- 0x1002

Login is using a Java Application --- 0x1003

8.3.7 Authentication Types

Plain Password Authentication --- 0x0000

Notes Token Authentication --- 0x0001

Encrpyted Password Authentication --- 0x0002

8.3.8 Awareness Constants

8.3.8.1 Awareness Context Constants

The following are Awareness context constants (all are 2 byte unsigned short) :

ADD 0x0068

REMOVE 0x0069

SNAPSHOT 0x01F4

UPDATE 0x01F5

UPDATE_ID 0x01F7

8.3.8.2 Awareness Presence Types

The following are Awareness presence types (all are 2 byte unsigned short) :

User 0x0002

8.4 Messages

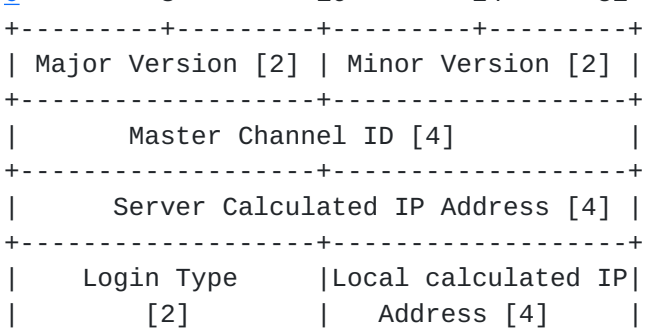
8.4.1 Basic Community Messages

8.4.1.1 Handshake

This message is sent by the initiator of a TCP connection when the TCP connection is created. This message creates a new master channel from the initiator to the recipient (a server).

The message structure includes two fields for the client IP address: one is calculated locally and one is calculated at the server end of the TCP connection. This allows for differences that may result when, for example, the client uses a proxy.

Message Body



```
+------------------+------------------+
|Local calculated IP|
|Address (cont.)    |
+------------------+
```

Description of Fields

Major/Minor Version
  Two 2-byte fields containing the originator's Sametime major and minor version numbers. The version number of the handshake requestor is checked by the server to determine whether the protocols are compatible, for values used in Sametime 1.5 see 8.3.4.

Master Channel ID
  A 4-byte field containing the Master Channel ID (MCID) of the new channel. When the message is sent from the originator, the MCID is 0.

server-calculated Client IP Address
  A 4-byte field containing the IP address of the originator as calculated at the server end of the TCP connection. When the message is sent from the originator, this field is 0. This number is replaced by the server before the message is routed to the next station. The IP address seen by the server may be different than that calculated locally by the originator (for example, when the client uses a proxy).

Login Type
  A 2-byte field representing the type of login originating the TCP connection. See 8.3.6 for values.

Local-calculated Client IP Address
  A 4-byte field containing the IP address of the originator as calculated locally.

8.4.1.2 HandshakeAck

This message is sent from the server to the initiator of the connection. It is an acknowledgement of a successful handshake and the creation of a master channel between the initiator and the server. This message also informs the initiator of the version number of the server and the initiator's IP address as calculated by the server.

Note: The Master Channel ID is included in the message header.

Message Body

```
0         8         16        24      31
+---------+---------+---------+---------+
```

```
| Major Version [2] | Minor Version [2] |
+-------------------+-------------------+
| server calculated client IP Address[4]|
+-------------------+-------------------+
```

Description of Fields

Major/Minor Version
  Two 2-byte fields containing the server's Sametime major and minor version numbers. The version number of the handshake requestor enables the server to determine if it can support the protocol version. The server also might use different protocols according to the version sent by the requestor. For values used in Sametime 1.5 see 8.3.4.

server-calculated Client IP Address
  A 4-byte field containing the IP address of the originator as calculated at the server end of the TCP connection. The IP address seen by the server may be different than that calculated locally by

  the originator (for example, when the client uses a proxy).

8.4.1.3 Login

This message provides login data to the server. The Login Type field determines the format and interpretation of the remainder of the message body. For a client, the message contains the Login Name and authentication data (see "Login" on 8.4.1.3).

Note: The Master Channel ID is included in the message header.

Message Body

```
0         8         16        24      31
+---------+---------+---------+---------+
| Login Type [2]    |
+-------------------+-------------------+
| Login Name [String]
+-------------------+-------------------+
|Authentication Type|
|        [2]        |
+-------------------+-------------------+
| Authentication Data [Opaque]
+---------------------------------------+
```

Description of Fields:

Login Type
  A 2-byte field. Specifies the type of Sametime login sending the

message. See 8.3.6.

Login Name
A string containing the user's login name. The login name is used in conjunction with the authentication data to identify and authenticate the user. The first two bytes specify the total length of the string.

Authentication Type
A two-byte field indicating the type of authentication. See 8.3.7. The authentication type determines the interpretation of the authentication data.

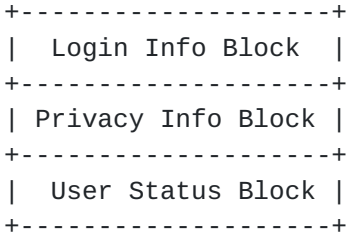
Authentication Data
An opaque containing data used for authentication of the user (e.g., a password or a token). The meaning of this data is determined by the authentication type.

8.4.1.4 LoginAck

This message is sent by the server to the login that requested a login after a successful login has been accomplished. It returns information about the requestor stored in the server to the requestor.

Note: The Master Channel ID is included in the message header.

Message Body



Description of Fields:

Login Info Block
The login described in this block (see "Login Info Block" in 8.2.1) is the login requestor. The Full flag is always TRUE.
The IP address
In this block is the IP address as seen from the server side. Therefore, if the client is using a proxy, the IP address will be the proxy's IP address.

Privacy Info Block
This block (see "Privacy Info Block" on 8.2.2) is used to send the user's privacy information, which the server retrieves, to the client.

User Status Block
  This block (see "Status Info Block" on 8.2.3) is used to
  send the user's status information. This information is based on
  other (current) logins of the same user.

8.4.1.5 LoginCont

This message has no body. It just notifies the server that the client wishes to be redirected (by the current server) to the remote server. This message is a response to a AuthPassed message sent to the client by the server, as described in 8.4.1.6.

8.4.1.6 AuthPassed

When a client logs into a server, and the server decides to redirect the client to another server, for any reason (the home server of the user is different or the user already has a login in another server), the server sends an AuthPassed message to the client, notifying it about a redirection possibility. The message indicates that the client can disconnect and connect to the specified server, or continue and have its connection redirected to the remote server by the current server. Should the client decide to continue with the current server, a LoginCont message should be sent to the server (as described in 8.4.1.5).

Message Body

```
0         8         16        24      31
+---------+---------+---------+---------+
| Home Server ID [String]
+---------------------------------------+
```



```
|        Home Server Version [4]        |
+---------------------------------------+
```

Description of Fields:

Home Server ID
  The server the client should connect to (or will be redirected to).

Home Server Version
  A 4-byte long describing the version of the remote server.

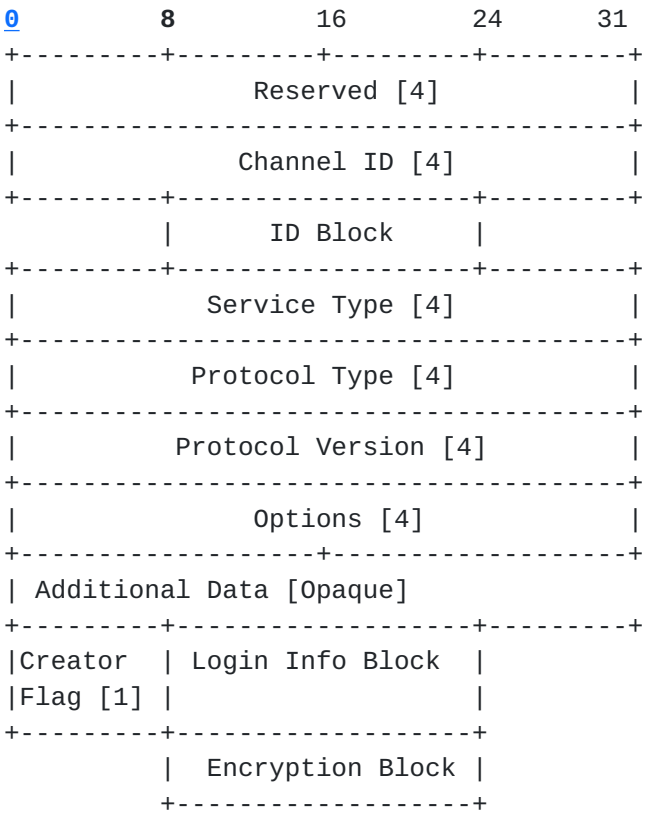
8.4.1.7 CreateCnl

All interactions between community entities take place over channels (that is, additional channels on top of the master channel). When a login needs to create a new channel to another entity, the channel creator sends a CreateCnl message to the target. A new channel is

created for every interaction. E.g. for every IM session and for every service. A new channel will be created for every N-way chat even though all the N-way chats that the login is participating in, are supplied by the same service provider.

Note: The Master Channel ID is included in the message header.

Message Body



Description of Fields:

Reserved
  A 4-byte field. Reserved for future use. Must be set to zero in this version of the protocol.

Channel ID

  The channel-ID (4 bytes long) of the channel.

ID Block
  A block of two strings that describes the target entity (single login) or user of the channel. If the ID string is empty, the channel is targeted based on the Service Type. For more details about this block, see "ID Block" on 8.2.4.

Service Type

A 4-byte field indicating the type of service that the channel creator is requesting from the target. For a list of all service types, see 8.3.2.

Protocol Type
A 4-byte field indicating the type of service protocol that will be used on the channel. For a list of all protocol types, see 8.3.3.

Protocol Version
A 4-byte field indicating the version number of the specified service protocol. This number is used for synchronizing both sides the channel on the same version of the protocol

Options

A 4-byte field. Reserved for future use. Must be set to zero in this version of the protocol.

Additional Data
Opaque data that further specifies the purpose of the channel. This information may be used by the target entity for various purposes.

Creator Flag
A single byte field, which indicates the presence of and Login Info block in the message. This allows the message size to be adjusted as necessary. For example, when the message originator is a client, the Login Info block is 0 (since the client does not send the information) and the Creator Flag is FALSE. When the message is routed through the client's server, the server adds the login Info block and the Creator Flag becomes TRUE.

Login Info Block
An optional block. Describes the channel creator (message originator). The Full flag is always TRUE. See "Login Info Block" in 8.2.1.

Encryption Block
See "Encryption Block" in 5.8.

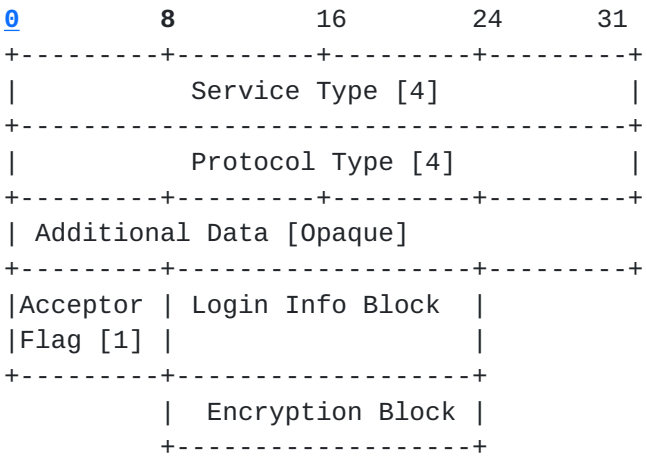
8.4.1.8 AcceptCnl

This message is sent from the channel acceptor to the channel creator as an acknowledgement of the channel creation. The message may also provide information about the acceptor, which is stored on the server. In the Login Info block in this message type, the Full flag is always

TRUE, and the login described is the channel acceptor.

Note: The Local Channel ID is included in the message header.

Message Body



Description of Fields

Service Type
  A 4-byte field indicating the type of service that the channel acceptor is providing. For a list of all service types, see 8.3.2.

Protocol Type
  A 4-byte field indicating the type of service protocol that will be used on the channel. For a list of all protocol types, see 8.3.3.

Protocol Version
  A 4-byte field indicating the version number of the specified service protocol. This number is used to synchronize both ends of the channel to use a compatible service protocol version. See 8.3.4.

Additional Data
  Opaque data that further specifies the purpose of the channel.

Acceptor Flag
  A single byte field, which indicates the presence of a Login Info Block in the message. This allows the message size to be adjusted as necessary. For example, when the message originator is a client, the Login Info block is 0 (since the client does not send the information) and the Acceptor Flag is FALSE. When the message is routed through the client's server, the server adds the Login Info Block and the Acceptor Flag becomes TRUE.

Login Info Block
  An optional block. Describes the channel acceptor (message originator). The Full Flag is always TRUE. See "Login Info Block" on 8.2.1.

Encryption Block
  See "Encryption Block" on 5.8.

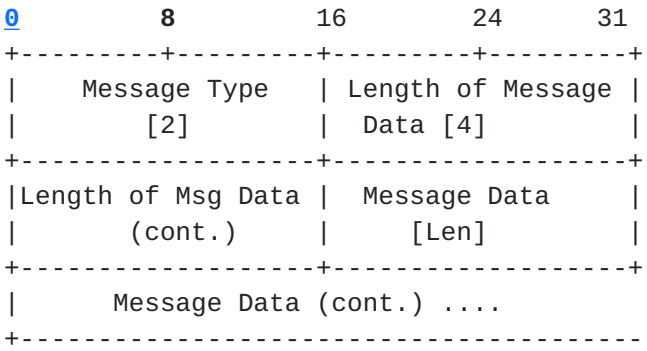
8.4.1.9 SendOnCnl

This message sends enclosed message data from one login to another on an existing channel. The Message Data field is interpreted using the Service Protocol that is specified for the channel.

Note that on encrypted channels the Message Data may be encrypted, however encrypted channels are not documented in this document.

Note: The Local Channel ID is included in the message header.

Message Body



Description of Fields:

Message Type
  A two-byte field specifying the message type within the established service protocol. For definitions, refer to the individual protocol chapter.

Message Data
  Opaque data (preceded by a 4-byte length field), which is interpreted by the service protocol established for the channel.

8.4.1.10 DestroyCnl

This is the last message on a channel. It contains enclosed message data.

Note: The Channel ID is included in message header.

Message Body

```
0         8         16        24      31
+---------+---------+---------+---------+
|            Reason Code [4]            |
+-------------------+-------------------+
| Message Data [Opaque]
+-------------------+-------------------+
```

Description of Fields

Reason Code

A 4-byte field specifying the reason for the destruction of the channel. This reason code is understood at the Master protocol level. For code reference, see "Constants" in 8.3.

Message Data
Opaque data (preceded by a 4-byte length field), which is interpreted by the service protocol established for the channel.

8.4.1.11 setUserStatus

This message informs the server of any changes in the user's status. This same message is also used by the server to inform other logins belonging to the same user of the status change.

Note: The Master Channel ID is included in the message header.

Message Body

```
+-------------------+
| User Status Block |
+-------------------+
```

Description of Fields

User Status Block
Describes the new user status. For details about this block, see "User Status Block" on 8.2.3.

8.4.1.12 SetPrivacyList

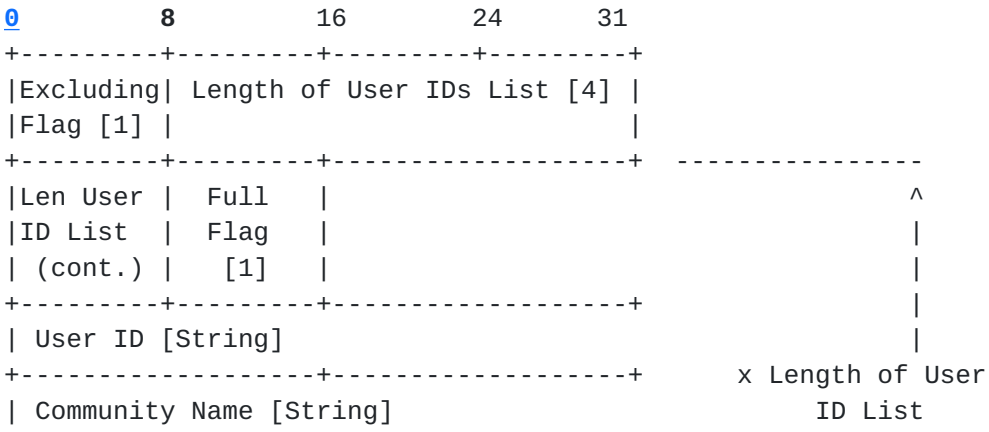
This message is sent from a client to request that the server set new privacy list options for the user. The privacy list is a list of users who are either permitted to, or prohibited from, seeing presence information about the requesting user ("Who Can See Me" options). The server is responsible for storing the change in a database by means outside the scope of this document.

This same message is also used by the server to inform other logins, belonging to the same user, of the privacy list options change. The server sends the notification message after the change has been

entered successfully in the database.

Note: The Master Channel ID is included in the message header.

Message Body



Description of Fields:

Excluding Flag
A single byte. If the value equals 1, the list of users that follows represents the users who are excluded from seeing the user whose privacy is described (all other users are permitted). If the value equals 0, the list contains the users who may see the user whose privacy is described (all others are excluded).

User List
A list of users representing either the users who are excluded from seeing the user whose privacy is described, or the users who are permitted to see the user whose privacy is described. The list begins with a 4-byte field specifying the number of items in the list. Each item in the list is a User Item. Each User Item contains a Full Flag, which indicates whether the item is full or not, a User ID string, and if the item is full, a User Name string. The User ID is a string containing the unique (within the community) ID of the user. The User Name is a string containing the name of the user, which is the name displayed on the screen in the user interface.

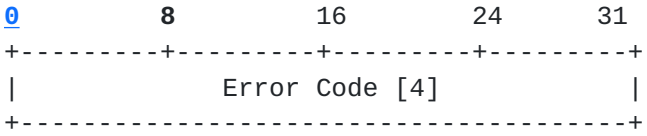
8.4.1.13 SetPrivacyDenied

This message is sent by the server to notify the client that a previous SetPrivacyList action (see 8.4.1.12) was not successful
(that is, the server attempted to save the changes using a server

application, and the changes were not saved successfully).

Note: The Master Channel ID is included in the message header.

Message Body



Description of Fields

Error Code
A 4-byte field indicating the reason for the denial of the request. This error code is understood at the Master protocol level. For code reference, see 8.3.1.

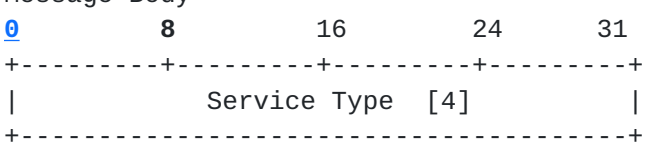
8.4.1.14 SenseService

This message is sent by a client to a server to request a notification when a specified service becomes available. This same message is sent by the server to the client as a notification when the service is available. The notification on availability of a service provider is

possible only within a single community.

Note: The Master Channel ID is included in the message header.

Message Body



Description of Fields

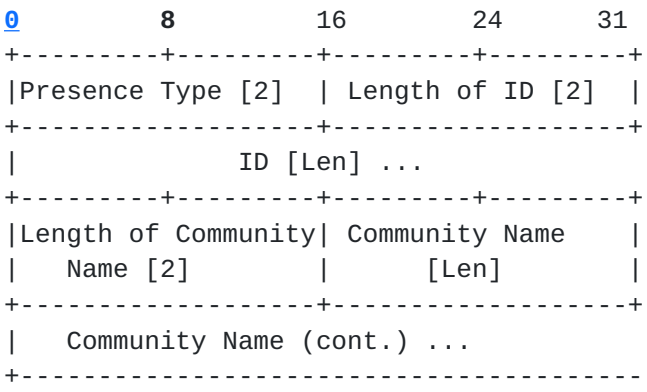
Service Type
A 4-byte field indicating the type of service on which the client is requesting notification. For a list of all service types, see 8.3.2.

8.4.2 Awareness Messages

Awareness messages are "transported" over an Awareness channel. These messages are used to subscribe (addWatch) to, and unsubscribe (removeWatch) from other users.

8.4.2.1 Awareness ID Block

This block is used as the IDs in the Awareness context. It is a combination of a presence type and a regular ID Block (see 8.2.4). For values of Presence Types see 8.3.8.2.

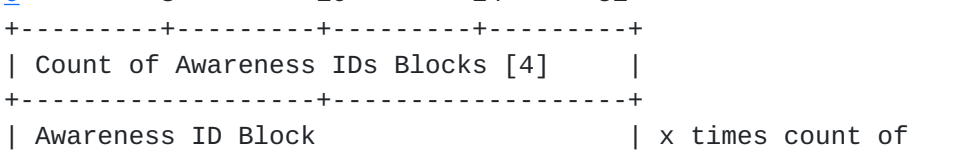


8.4.2.2 AddWatch

This message "subscribes" the initiator to a presence. It is sent from the client to the Awareness server application.

Message Type is ADD.
(For hexadecimal values, see 8.3.8.1).

Message Body



Description of Fields:

Count of Awareness IDs Block
The number of Awareness ID Blocks that follows in the message.

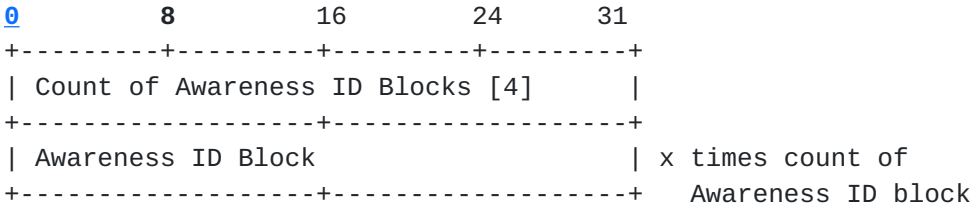
Awareness ID Block
The presence to subscribe to. See "Awareness ID Block" on 8.4.2.1.

8.4.2.3 RemoveWatch

This message "unsubscribes" the initiator from a presence. It is sent from the client to the Awareness server application.

Message Type is REMOVE.
(For hexadecimal values, see 8.3.8.1).

Message Body



Description of Fields:

Message Context
  The message context is a two-byte unsigned short. For this message
  the value is "REMOVE" (for hexadecimal values, see
  8.3.8.1).

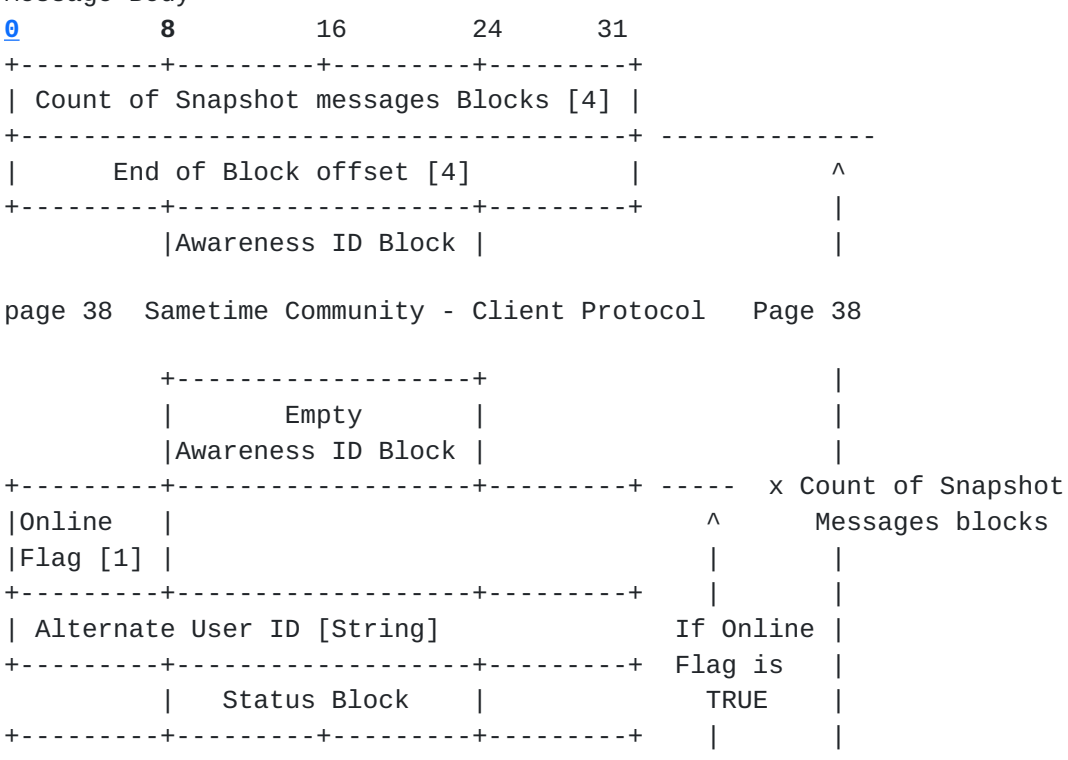
Count of Awareness IDs Block
  The number of Awareness ID Blocks that follow in the message.

Awareness ID Block
  The presence to see subscribe to. See "Awareness ID Block" on
  8.4.2.1.

8.4.2.4 Snapshot

This is sent from the Awareness service application to the clients.
For each subscribe request, the Awareness service synchronizes the
subscriber with the current Awareness knowledge of the presence.

Message Body



```
| User Name [String]                          v       v
+-------------------------------------+ ------------
```

Description of Fields:

Count of Snapshot Messages Blocks
  The number of Snapshot messages to follow.

End Of Block Offset
  The offset (in bytes) of the end of the current block. Used for
  future compatibility.

Awareness ID Block
  The presence in question. See "Awareness ID Block" on
8.4.2.1.

Empty Awareness ID Block
  For internal use. Should be empty. See "Awareness ID Block" on
  8.4.2.1.

Online Flag
  The current presence of the user. If TRUE, the user is online and
  the rest of the block contains details of the user. If FALSE, this
  is the end of the block.

Alternate User ID
  A string, for internal use. Should not be used.

Status Block
  The current status of the user. See "Status Info Block" on
8.2.3.

User Name
  The display name of the user. Should not be used; this information
  should be known to the client.

8.4.2.5 Update

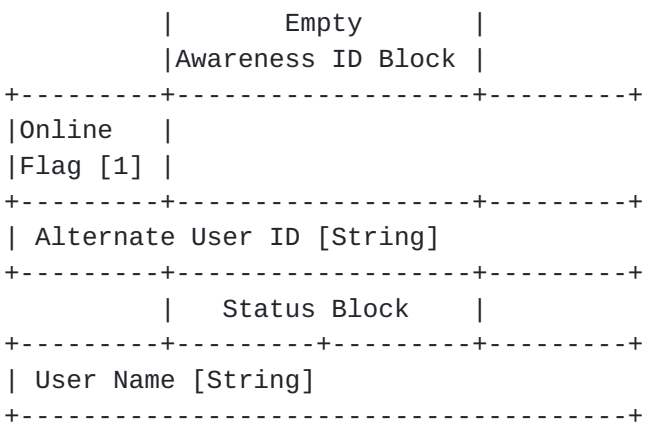
Every time the presence changes it's status, or logs off, an update is sent to it's subscribers. It is sent from the awareness service provider to the client.

Message Body

```
0         8         16        24      31
+-------------------------------------+
```



```
|      End of Block offset [4]          |
+---------+------------------+---------+
          |Awareness ID Block |
          +------------------+
```



Description of Fields:

End Of Block Offset
  The offset (in bytes) of the end of the current block. Used for
  future compatibility.

Awareness ID Block
  The presence in question. See "Awareness ID Block" on
8.4.2.1.

Empty Awareness ID Block
  For internal use. Should be empty. See "Awareness ID Block" on
  8.4.2.1.

Online Flag
  The current presence of the user. If TRUE the user is online, and
  the rest of the block contains details of the user. If FALSE, this
  is the end of the block.

Alternate User ID
  A string, for internal use. Should not be used.

Status Block
  The current status of the user. See "Status Info Block" on
  8.2.3.

User Name
  The display name of the user. Should not be used, this information
  should be usually known to the client.

8.4.3 Instant Messaging

Instant Messages are sent over an "Instant Message Channel". Messages are typed in order to convey various types of data. When the IM channel is created, certain values should be sent in the opaque that is contained in the channel creation.

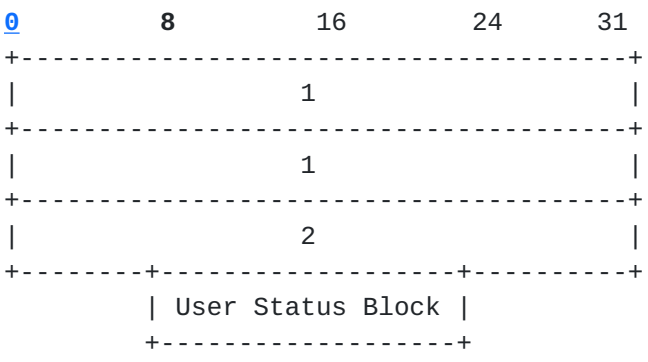
8.4.3.1 IM channel creation

When creating an IM channel, the opaque that is sent as part of the channel creation should contain two 4-byte words each of them containing 1.

Create IM channel requests that contain other values are beyond the scope of this document and should be rejected by clients that their implementation is based on this document.

8.4.3.2 IM channel accepting

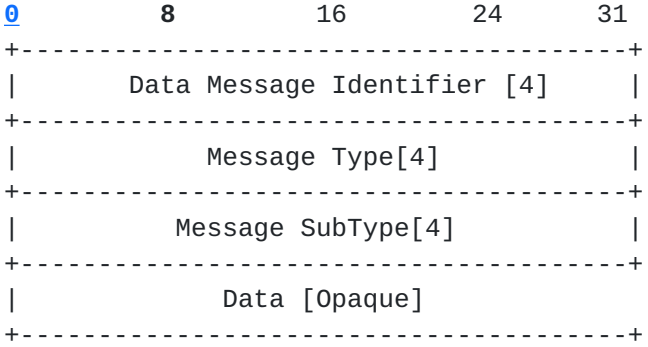
When accepting an IM channel, the opaque that is sent as part of the channel acceptance should contain the following:



User Status Block should be filled with the current status of the user to whome the IM channel was created.

8.4.3.3 Data Message

Message Body



Description of Fields:

Data Message Identifier
  The Value is 0x00000001. It identifies the message (to the receiver) as a "Data Message".

Message Type

Used to identify the type of the data sent.

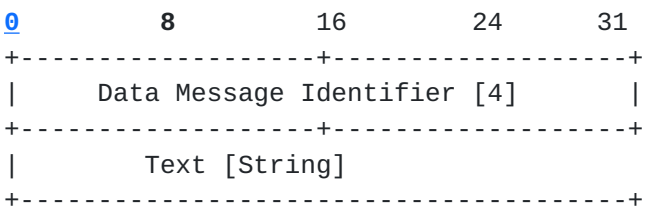
Message SubType
Used to further identify the type of the data sent.

Data Length
The length of the data sent.

Data
The raw data.

8.4.3.4 Text Message

Message Body



Description of Fields:

Data Message Identifier
The Value is 0x00000002. It identifies the message (to the receiver) as a text message.

Text Length
The length of the text message

Text
The message itself.

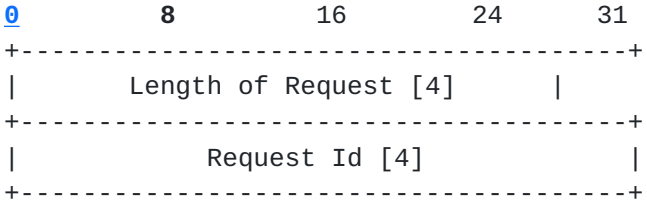
8.4.4

start talking - data with 0 bytes.

8.4.4.1 Resolve Request

Used to request the Resolver Service to resolve user name(s) to user ID(s).

Message Body



```
|    Number or Names to Resolve [4]     |
+-------------------+-------------------+
| Name [String]                            X Times number of names
+-------------------+-------------------+
|            Request Options [4]        |
+---------------------------------------+
```

Description of Fields:

Length of Request
  Size of Message Body in bytes including this field.

Request Id
  Issued by the client, used to identify the request.

Number of Names to resolve
  Indicates how much names there are to follow...

Names
  Strings of names to resolve. Appears as times as indicated before.

Options
  A bit mask of the following options:

    0x00000001 - Return only unique matches (if number if matches <> 1, return empty list of matches

    0x00000002 - Return first match only

    0x00000004 - Search all available directories (otherwise do not search beyond the directory were the first match was found

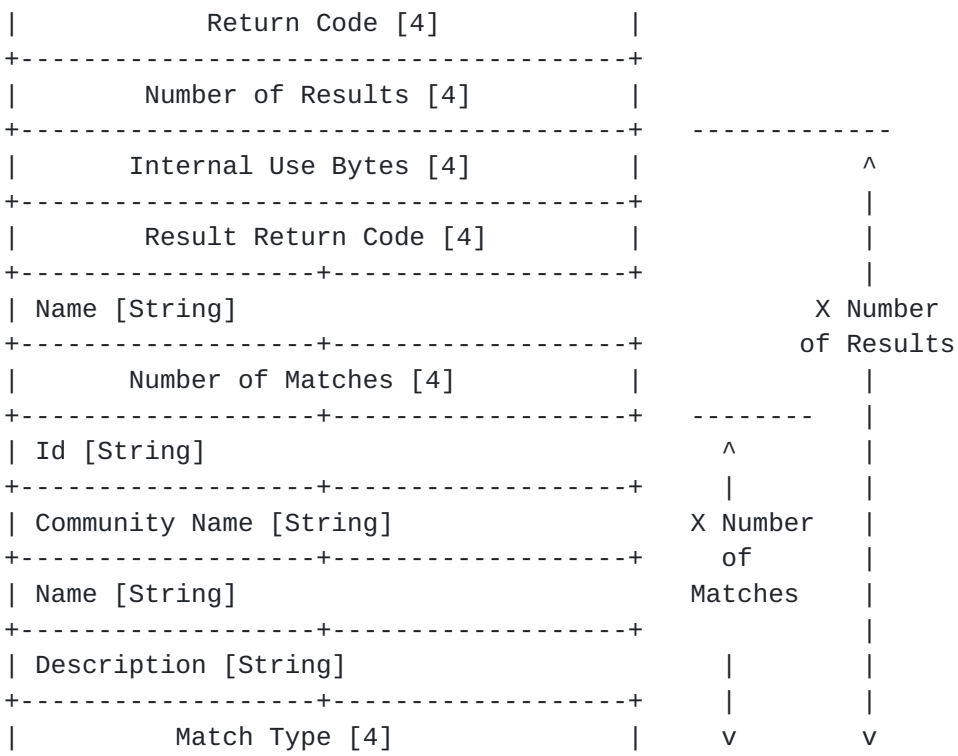
    0x00000008 - Users bit (should always be on).

8.4.4.2 Resolve Response

The response for the clients ResolveRequest. This message is a list of results, each result being a list of matches.

Message Body

```
0         8         16        24      31
+---------------------------------------+
|       Internal Use Bytes [4]          |
+---------------------------------------+
|            Request Id [4]             |
+---------------------------------------+
```



Description of fields:

Internal Use Bytes
  These bytes are used for backward / forward compatability. These bytes should be ignored in the resolve response.

Request Id
  Identifies the response for the request Id'ed by the client

Return Code
  Of the entire response.

Number of Results
  Number of nested results to follow

Result Return Code
  A return code for a specific name.

Number of matches
  Number of matches to follow

Id & Community
  Created for the UserId resolved

Name

The "official" user name

Description
Description of the user as kept in the database

Match Type Should always be 0x00000001.

9. Acknowledgements

The authors of this document wish to acknowledge the following that contirbuted to this document: Mark Day, Alon Kleinman, Maxim Kovalenko & Marjorie Schejter.

10. Authors Addresses

Avshalom Houri
Ubique/Lotus
Building 18/D, Sceince Park
Kiryat Weizmann, POB 2523
Rehovot 76123 Israel
avshalom@ubique.com

Ittai Golde
Ubique/Lotus
Building 18/D, Sceince Park
Kiryat Weizmann, POB 2523
Rehovot 76123 Israel
ittai@ubique.com