|  |
| --- |
| Blue Husky Programming |
| BHIM Concept Document |
| The high-abstraction description of how BHIM is to be programmed |

|  |
| --- |
| Kyli Rouge, Ben Leggiero  8-31-2015 |

Table of Contents

[2 High-Level Concept 2](#_Toc406059436)

[3 Messages 2](#_Toc406059437)

[3.1 Message Parsing 2](#_Toc406059438)

[3.1.1 Examples of Roleplaying Messages 3](#_Toc406059439)

[3.1.2 Text to Transmit 3](#_Toc406059440)

[3.1.3 Text to display 3](#_Toc406059441)

[3.2 Message Sending 3](#_Toc406059442)

[3.3 Message History 3](#_Toc406059443)

[4 Contacts 4](#_Toc406059444)

[4.1 UUIDs 4](#_Toc406059445)

[4.2 Profiles 4](#_Toc406059446)

[4.2.1 Presence 4](#_Toc406059447)

[5 Roleplaying 4](#_Toc406059448)

[6 Interface 5](#_Toc406059449)

[7 Preferences 5](#_Toc406059450)

[7.1 Formatting 5](#_Toc406059451)

[7.2 Security and Privacy 5](#_Toc406059452)

# High-Level Concept

An instant messenger designed for roleplaying.

# Messages

A message is anything sent from one client to another.

The following are all the possible types of messages (User initiated, automated information, automated error):

1. Plain text
   * A message to another user or chatroom
2. Media
   * An alert that there is a large packet of data to be transmitted only when both sender and receiver agree a transmission should take place
3. Request for message change
   * A request that a previously-sent message should be edited or removed entirely
4. Announcement of information update
   * An announcement that the user’s meta information has changed
5. Request for information update
   * A request that another user’s or chatroom’s meta info be sent back to the user for purposes of updating the local contact list
6. Announcement of successful message receipt
   * An announcement that a message was successfully received
7. Request for re-send
   * A request that a specific message should be re-sent
8. Sending failed
   * An announcement that a message could not be sent
9. Sending failed permanently
   * An announcement that a message could not be sent and re-sending is impossible

Automated errors should be rare, as IM messages should all be sent over TCP, which should catch and fix these before the user notices and error has occurred. However, they’re kept in for the sake of a complete and more abstract design

## Message Parsing

If a message is a plain text message, it might be formatted for roleplaying. To detect this, we loop through a set of rules, either predefined or set by the user, on what constitutes a roleplaying message.

Each segment of a roleplaying message will be denoted differently. The following are all possible ways to denote a segment of a roleplaying message:

* No denotation
  + Exactly one type of segment must have this rule
* Preceding denotation
  + Rule must also define surrounding denotation to escape this context
* Surrounding denotation
  + Beginning and ending delimiters may be different, or may be identical

### Examples of Roleplaying Messages

* Plain text denoting speech
* Plain text denoting an action or narration
* \*Text surrounded by identical delimiters denoting action or narration\*
* "Text surrounded by identical delimiters denoting speech"
* [Text surrounded by delimiters denoting out-of-character communication]
* /me Text preceded by a delimiter denoting action or narration
* /me Text preceded by a delimiter denoting action or narration “along with text surrounded by identical delimiters denoting speech”
* \*Some combination of\* the above described [methods of delimiting] \*different scenarios\*
* //Note that “these example delimiters” are not the ((only way)) of “denoting a -roleplay message-“

### Text to Transmit

* Speech segments should not be denoted specially
  + But if necessary, they can be denoted with <speech>special speech tags</speech>
* <move>Actions and narration segments should be surrounded by ‘move’ XML tags</move>
* <ooc>Out-of-character segments should be surrounded by ‘ooc’ XML tags</ooc>
* Mixed segments should be <move>surrounded by</move> their respective XML tags, <move>even if <ooc>they are nested</ooc></move>.

### Text to display

Once received, a roleplaying message must be detected and parsed. All <move></move> and <ooc></ooc> tags must be replaced by the user’s preferred move and out-of-character delimiters, and speech surrounded by their preferred speech delimiters, if defined and applicable.

## Message Sending

To send a message, the client must understand what is being sent, from whom it is being sent, and to whom it is being sent.

To understand these, each user and chatroom is given a UUID, which is sent with the message, and each client has a connection to a database that will fetch and update what IP addresses that UUID is currently using. The user’s records expire after 1 day, so the client must tell the server its IP and user’s UUIDs more than once per day to ensure smooth communication between clients. Chatroom records expire once every user has left the room.

## Message History

Each client keeps its own history of private and chatroom messages. Each conversation stores the history in a single file. Each historical message is stored with a timestamp of when it was sent, the UUID who sent it, and the parsed (XML-like) text of the message.

# Contacts

Each contact may be a user (player), chatroom (roleplay), or character.

## UUIDs

* A player’s UUID is determined the first time they use the messenger.
* A character’s UUID is determined the first time the character is created
* A roleplay’s UUID is determined the moment a roleplay is created and ready to be played in.

As implied by the name, each UUID must be universally unique, and never have any duplicates. For this reason, when the user first connects to a server, the client sends its player’s name and IP, and the server uses these to create a UUID, store it, and send it back to the player. From that moment on, the player will only use that UUID, even if playing on multiple clients or servers.

The process is similar for roleplays, except the roleplay’s UUID is salted with a combination of the initial name of the roleplay and the UUIDs of all its players.

## Profiles

A profile contains a contact and all the meta info about that contact. This meta info includes:

* Display Name
* UUID
* Tagline
* Current Presence

One user may have several characters, and thus several profiles. However, multiple users cannot share a single profile.

### Presence

A profile’s presence describes their current relationship with the client. It can be:

* Online & Seeking RP
* Online
* Away
* Do Not Disturb
* Invisible
* Offline
* Unknown

All other players will see “Invisible”, “Offline”, and “Unknown” as the same thing.

# Roleplaying

The main feature of BHIM is roleplay friendliness

# Interface

A clean and efficient interface is key to gaining many users quickly

# Preferences

Aim for few preferences that easily give the user powerful customizability

## Formatting

To be considered

## Security and Privacy

To be considered