Data Comm & Networks
Request for Comments: 101011
Obsoletes: 0
Category: Standards Track

Colin Murphy
Rochester Institute of Technology
2019

# PeerChat Communication Protocol

Status of this memo

This document specifies the most up to date protocol specifications for communicating with other users using the PeerChat protocol. Distribution of this memo is unlimited and all copyright is waived.

Abstract:

To enable communication between peers a protocol has been established. This document fully outlines the data, format and order in which contents may be sent to be parsed by peers in the network. This document only outlines the protocol, how individual clients choose to handle .the received data is outside of this documents scope.

Table of Contents



## 1. Introduction

All communication with the health service is done using TCP networking. Clients must be provided with a host and port to connect to. Once a connection has been opened between peers a series of messages are passed to exchange information and allow the connecting client to discover peers.

## 2. Data types and definitions

All messages are passed as JSON objects terminated by a newline. Newlines may be in Windows (\r\n) or Unix (\n) style. All JSON messages conform to the standardization

outlined in RFC-7159 with data being escaped as described. Any messages that do not follow the standards set in this document will be ignored with no indication.

## 3. Message Types

Every JSON object will contain a “type” field, the value of this field determines the format of the rest of the object. Any messages with an unknown type will be ignored allowing compatibility with future versions of this protocol. All data will be sent as text with numbers being sent as their string representation.

Note: While all sample messages in this document are valid the editor seems to be inserting invisible characters that invalidate it.

Version 1.0 Supports the following types

1. join
2. join-reply
3. who
4. Who-reply
5. leave
6. message

### 4a. join

A request to join contains 4 additional fields, name: the users name, age, the users zip code and the port the peer listens on. Name is a string with no set length, zip and age and port are ints, again with no restrictions on values. Regardless of the datas validity the connection is accepted, any invalid data besides port should be filled in by implementing code.

Example: {“type”:“join”, “name”:“Bob”, “age”: “15”, “zip”:”12345”, “port”: “1234”}

### 4b. join-reply

On receiving a join the receiving client sends their information to the connecting client. This message is of the same format as joins, the only difference being the type.

Example: {“type”:“join-reply”, “name”:“Kevin”, “age”: “20”, “zip”:”12345”, “port”: “1235”}

### 4c. who

After joining a peer a user can learn about the other connected peers by issuing a who. The message contains no additional fields.

Example: {“type”: ”who”}

### 4d. who-reply

On receiving a “who” the receiver sends a list of all the peers in the chat. The list only contains the users that the sender should learn about so it does not contain the receiver or the sender. For a session with n connected members a who request returns n-2 entries, the reason being that a peer already knows who they’re talking to and doesn’t need to be told about themselves.

The list of peers is a JSON array containing JSONObjects, these objects contain an ip and port field with the values being the string representation of the peers ip and port. The array is inserted into the main object with a key of “peers”

Example: {“type”:”who-reply”, “peers”: [{“ip”: “1.2.3.4” “port”:”1234”}, {“ip”: “2,3,4,5” “port”:”4567”}]}

4e. leave

A peer wishing to disconnect issues a leave message. The message contains no additional fields and has no reply, on receiving this message a peer should assume that the sender is already gone and handle that accordingly.

Example: {“type”: “leave”}

4f. Message

To send a message the sender issues a message with type “message”. This message contains a single field called message, the value of this being the JSON escaped string typed by the user. No response is necessary, tcp ensures that everyone got the message.

Example: {“type”:”message”, “message”: “Hello There!”}

5. Message Flow

When joining a peer in the chat a tcp connection must first be established. Immediately after the connection is made the joining client issues a join. On receiving the join the receiver sends back a join-reply welcoming them to the chat. From there the joining client issues a who to their peer, the peer responds with a who-reply. Following this exchange the joining client knows about all peers in the network and messages are sent to everyone.

| Person A | Person B | Person C |
| --- | --- | --- |
| Opens TCP connection | Accepts Connection | |
| Join A-> | | |
| | <- join-reply | |
| Who -> | | |
| | <- who-reply | |

| | | |
|---|---|---|
| Opens TCP connection<br>Join C -> | | Accepts connection<br><br><- join-reply |
| Message -> | Receives message | Receives message<br><-leave |
| Terminates C's connection | Terminates C's connection | |
| Message -> | Receives Message | |

6. Security Considerations

This protocol contains absolutely no security or mechanisms to maintain data integrity. Integrity is insured by the TCP networking but all data will be sent in plain text with no obfuscation. No mechanism exists to detect messages sent are actually from the sender. For casual chatting this is fine, but sensitive information should never be communicated over this protocol.

Authors Address
Colin Murphy
Rochester Institute of Technology
Email: clm3888@rit.edu