Peerchat_sodergren_Erikson

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

This RFC specifies the requirements for messages sent/received by a peerchat program in which users connect to existing peers in a chat group to join the entire group. A program following this RFCs specifications will be able to communicate with any other program also following them.

For the purpose of this protocol, int and uint32_t are 4-byte values, and a char is a 1-byte value representing the ascii value of a string character.

Data about users other than address/port MUST be shared using the following userdata structure.

[Char username[12] | int age | int zip | int port | uint32_t address]

Username is the null-terminated name of the user sending the message.

Port is the value of the port that the user sending the message listens for new connections on.

Address is the ipv4 address of the user sending the message.

Age and zip are the age and zipcode of the user sending the message.

All messages sent by the program MUST be sent in little endian form.

A normal chat message MUST always consist of only the string to be shared with peers.

Any other message MUST begin with an integer(code) specifying what type of message it is.

1=request for userdata and list of addresses/ports for other peers

[code | userdata]

2=request for only userdata

[code | userdata]

3=response to message 1

[code | userdata | int numpeers | uint32_t address1 | int port1 | uint32_t address2…]

4=response to message 2

[code | userdata]

Numpeers is the number of peers the sending user is currently chatting with.

Address# and port# are the network form ipv4 address and the port of each peer the sending user is chatting with.

When a new user joins a group, they MUST first send a message with code 1 to the initial peer they joined and process the response, and then send all necessary messages with code 2 to other exiting peers and process those responses, before any other action can be taken.

All users MUST keep TCP sockets open for communication with their peers.

When a user leaves the group they MUST close the sockets related to each peer, no other action is required.

All other details of implementation have no impact on communication with another program making use of this protocol and MAY be chosen in any way.

EXAMPLE: user Dawn and user Lily are currently connected, and user George attempts to join. George sends a code=1 message to Lily and receives a code=3 message with only 1 pair of port/addr which tells George the port and ipv4 address that Dawn is listening for new connections on. George now sends a code=2 message to Dawn and receives a code=4 message in return. Now all 3 peers are connected by TCP sockets and can freely communicate.