C494
Internet Draft
Intended status: IRC Class Project Specification
Expires: July 2016

Lisa Gray
Portland State University
June 2, 2016

Internet Relay Chat Class Project
lisa-gray-irc-rfc-pdx-cs494.pdf

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79. This document may not be modified, and derivative works of it may not be created, except to publish it as an RFC and to translate it into languages other than English.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet- Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at
http://www.ietf.org/ietf/1id-abstracts.txt

The list of Internet-Draft Shadow Directories can be access at http://ietf.org/shadow.html

This Internet-Draft will expire on Fail 1, 0000.

Copyright Notice

Abstract

The following memo describes the communication protocol for a client/server Internet Relay Chat system for the Internetworking Protocols class at Portland State University.

Table of Contents

Introduction

This document specifies a simple Internet Relay Chat (IRC) protocol where clients communicate to one another by sending packets with messages and commands to a central server which relays the information to clients connected to the server. Users consist of client programs and bots (programs designed to communicate on certain events or provide services to other users/bots) implementing the protocol discussed in this document.

Users join various rooms containing other users (or just one user for private messaging) and submit messages to be relayed to other users. Users may also issue various commands to the server for listing rooms or users, creation and deletion of rooms, and more.

Conventions used in this document

TBD

Basic Information

Clients communicate to the server using a TCP connection. The server listens on port 1994 for connections sent from client applications and manages a persistent connection asynchronously. This allows the client and server to exchange messages via the same channel. Since the application and protocol is managed asynchronously clients are able to send messages and commands at any time and the the server can respond to the client at any time as well. Users and Rooms are managed with a SQlite database on the server-side (note: the SQlite database might not be on the same machine as the IRC server).

Message Infrastructure

Generic Packet Structure

The generic packet header information from which all other packets are derived from is found below:

```
class Packet(object):
    """generic packet class - used as a keep-alive message as well"""
    def __init__(self, opcode, status, err):
        self.opcode = opcode
        self.status = status
        self.err = err
```

Field definitions

- self.opcode - this field specifies the type of packet to follows
- self.status - this field is used on the server as a way to respond to client whether the command or message sent worked as intended (uses an Enum which contains OK, and ERR)
- self.err - this field is used to carry error message between the server

Operation Codes (opcodes)

```
CONNECT = 1
DISCONNECT = 2
MSG = 3
PRIVATE_MSG = 4
BROADCAST_MSG = 5
LIST = 6
JOIN = 7
LEAVE = 8
```

```
CREATE = 9
DESTROY = 10
FILE_TRANSFER = 11
```

Usage

The generic base packet/header used to contain information that exists in all derived packets

Username and Room Name Semantics

Users will be differentiated between each other through the use of usernames and a room can be thought of as a list of users that defines the membership between what users are in the channel and will receive messages from the channel. These names must follow the rules below:

- Names must be unique.
- Names must consist of ASCII character values.
- If any of the above rules are broken, the server will send a Connect packet back with appropriate error and terminate the connection.

Validation will be handled both on the server and client programs.

Packet Types

Connect Packet

```
class Connect(Packet):
   def __init__(self, username, config={}, status = None, err = None):
      super().__init__(Opcode.CONNECT, status, err)
      self.username = username
      self.config = config
```

Usage

The connect packet is the first message a client sends to the server. It is used to define the client's username that will be used in subsequent messages. The server will respond by returning the packet with the status field filled with either an OK or ERR. If an error occurs an exception will be raised and the client will handle any issues. (eg. the username has already been taken). If a config file exists on the client, it will allow the client to join any specified rooms or create them if they do not exist.

Field Definitions

- self.username - the username to be used for the client
- self.config - a dictionary containing any configuration information such as rooms to automatically join or create.

Disconnect Packet

```
class Disconnect(Packet):
   def __init__(self, username, status = None, err = None):
      super().__init__(Opcode.DISCONNECT, status, err)
      self.username = username
```

Usage

The disconnect packet is used by the client and server to disconnect from one another. The username field will allow the server to clear the user from the database and any rooms they have joined. The server will send out a Message packet to all the rooms the user have joined displaying that the user has disconnected.

The server may issue a disconnect packet if the client has not sent a keep_alive packet within the last 5 minutes, or an exception on the server has been raised and it is gracefully shutting down. Also, used to disconnect clients for server maintenance.

Field Definitions

- self.username - the username of the client disconnecting

Message Packet

```
class Message(Packet):
   def __init__(self, username, message, room, status = None, err = None):
      super().__init__(Opcode.MSG, status, err)
      self.username = username
      self.message = message
      self.room = room
```

Usage

The message packet is the basic form of communication between clients. A client sends this packet to the server with their username, message, and current room with which the server distributes the message to the other clients in the room.

Field Definitions

- self.username - the username of the client disconnecting

- self.message - the message to be distributed
- self.room - the room to distribute the message to

Private Message Packet

```
class PrivateMessage(Packet):
    def __init__(self, username, message, send_to, status = None, err = None):
        super().__init__(Opcode.PRIVATE_MSG, status, err)
        self.username = username
        self.message = message
        self.send_to = send_to
```

Usage

The private message packet is used to send messages between the specified users. The server creates a room between the two or more users and distributes messages between them.

Field Definitions

- self.username - the username of the client disconnecting
- self.message - the message to be distributed
- self.send_to - the user(s) to send the message to

Broadcast Packet

```
class Broadcast(Packet):
    def __init__(self, username, message, rooms=[], status = None, err = None):
        super().__init__(Opcode.BROADCAST_MSG, status, err)
        self.username = username
        self.message = message
        self.rooms = rooms
```

Usage

The broadcast packet is used to send messages to multiple rooms. This can be issued by users and the server. The broadcast message has a label BROADCAST prepended to the message.

Field Definitions

- self.username - the username of the client
- self.message - the message to be distributed
- self.rooms - the room(s) to broadcast the message to.

List Packet

```
class List(Packet):
   def __init__(self, room=None, response = [], status = None, err = None):
      super().__init__(Opcode.LIST, status , err)
      self.room = room
      self.response = response
```

Usage

The list packet is used for listing the rooms or users within a room.

Field Definitions

- self.room - the room specified to list users from
- self.response - the response issued by the server

Join Packet

```
class Join(Packet):
   def __init__(self, username, room, status = None, err = None):
      super().__init__(Opcode.JOIN, status, err)
      self.username = username
      self.room = room
```

Usage

The join packet is used to join the specified room. If the room does not exist the room will be created by the server. When a user joins a room, the server issues a message packet to notify that the specified user has joined the room.

Field Definitions

- self.username - the username of the client joining
- self.room - the room specified to join

Leave Packet

```
class Leave(Packet):
   def __init__(self, username, room, status = None, err = None):
      super().__init__(Opcode.LEAVE, status, err)
      self.username = username
      self.room = room
```

Usage

The leave packet issued by the client to specify that the user is leaving the room. When a user leaves a room, the server issues a message packet to the room stating that the user has left the room.

Field Definitions

- self.username - the username of the client leaving
- self.room - the room specified to leave

Create Packet

```
class Create(Packet):
   def __init__(self, room, status = None, err = None):
      super().__init__(Opcode.CREATE, status, err)
      self.room = room
```

Usage

The create packet is issued by the client to create the specified room.

Field Definitions

- self.room - the room specified to create

Destroy Packet

```
class Destroy(Packet):
   def __init__(self, room, status = None, err = None):
      super().__init__(Opcode.DESTROY, status, err)
      self.room = room
```

Usage

The destroy packet is issued by the client to destroy/remove the specified room.

Field Definitions

- self.room - the room specified to destroy/remove

Error Handling

TBD

Extra Features Supported

TBD

Conclusion & Future Improvements

Future improvements include: adding username validation, creating error codes and giving appropriate error messages rather than generic catch all statements, bot framework, a gui client, secure messaging, file transfers

Security Considerations

Input validation, encrypted messaging,

IANA Considerations

None

Acknowledgements

This document was prepared using google docs.
I would like to thank Byron Marohn for the RFC template.

Author's Address

Lisa Gray
Portland State University Computer Science
1825 SW Broadway, Portland OR 97201

Email: gray7@pdx.edu