CS 594

Internet Draft

Intended Status: IRC Class Project Specification

Expires: June 2020

MacKenzie Cooper  
Carissa Allen

Portland State University

December 6, 2019

Internet Relay Chat Class Project

Protocol Draft

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 accessed at http://www.ietf.org/shadow.html

This Internet-Draft will expire on June 6, 2020. Copyright Notice

Copyright (c) 0000 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Abstract

This memo describes the communication protocol for an IRC-style client/server system for the Internetworking Protocols class at Portland State University.

Table of Contents

[1. Introduction 2](#_Toc26529482)

[1.1 Server 3](#_Toc26529483)

[1.2 Client 3](#_Toc26529484)

[1.3 Rooms 3](#_Toc26529485)

[2. IRC Concepts 3](#_Toc26529486)

[2.1 One-to-one Communication 3](#_Toc26529487)

[2.2 One-to-many Communication 4](#_Toc26529488)

[3. Message Details 4](#_Toc26529489)

[3.1 Creating a Connection 4](#_Toc26529490)

[3.2 List Rooms 4](#_Toc26529491)

[3.3 Create a Room 4](#_Toc26529492)

[3.4 Join a Room 4](#_Toc26529493)

[3.5 Leave a Room 5](#_Toc26529494)

[3.6 Broadcast a Message 5](#_Toc26529495)

[3.7 Send a Public Message 5](#_Toc26529496)

[3.8 Send a Private Message 5](#_Toc26529497)

[4. System Failures 6](#_Toc26529498)

1. Introduction

This specification describes a simple Internet Relay Chat (IRC) protocol by which clients can communicate with each other. This protocol allows a user to communicate to multiple other connected users in common virtual rooms.

This system utilizes a main server, which acts as a central hub for clients to connect to. Once the client is connected, the server will relay messages received from the client to the users connected to the same room.

In the initial implementation, the functionality of the IRC client will be limited to the following: clients will have the ability to create a room, join a room, leave a room, and list rooms available. Additional features may be added in the future and this document will be updated as necessary.

## Server

A single server will enable client communication, serving as the central point of contact for all messages sent through the application. Once the server is started on a valid port, clients can connect to the server. The server will initialize a thread pool, running an infinite loop to listen for incoming connection requests from the client. When the client sends a message, the server will receive it first, and will then transmit the message to all connected users, or to a particular room.

## Client

A client is any other computer connected to the server. Once a client is connected to the server running on the same port, the thread count is incremented and associated to that client. The client may also select a unique username, which will appear in the window next to the number associated with that client. Any time a client broadcasts a message to other users, the client’s unique username and number is displayed. This allows other clients to identify who is connected to a room, or who sent a particular message.

## Rooms

Clients will have the ability to connect to virtual rooms. Rooms are specified by a unique name and are public to all clients. A client can create a new room, join a room, leave a room, list available rooms, or send messages to other users connected to a particular room.

1. IRC Concepts
   1. One-to-one Communication

One-to-one communication indicates messaging between clients, as there is only one server available to the system. If Client #1 wants to send a message to Client #2: the message from Client #1 is first received by the server, which then forwards the message to Client #2.

* 1. One-to-many Communication

One-to-many communication indicates messaging between multiple clients. If Client #1 wants to send a message to multiple users: the message from Client #1 is first received by the server, which then forwards the message to all clients connected to the same room.

1. Message Details

## Creating a Connection

To establish a connection, a user must run Server.main(), which will open a window to enter a valid port number, username (default is HOST), and start the server. Once this is complete, clients can connect by running Client.main(), which will open a window to enter a valid port number, unique username, and host (default is localhost). This initiates a connection between the client and the central relaying server. At this point, clients can broadcast messages to all connected users. If the client joins a room, the client can send messages to other users connected to that room.

## List Rooms

Both the server and client have access to the rooms available. The client may view the list of rooms available through the Graphical User Interface (GUI), where they will all be displayed by default. Rooms are publicly accessible to all clients.

## Create a Room

To create a room, the client must enter a command. All commands are prefaced by the ‘@’ symbol, followed by an action. Clients can create a room using the action word ‘create’ followed by the name of the room they wish to create. Once created, the client automatically joins the room, and the server will display a message to confirm the client has successfully created the room.

Example:

<client> : @create *<room name>*

<server> : Room *<room name>* has been created under id # *<room id>* with you in it.

## Join a Room

To join a room, the client must enter a command. All commands are prefaced by the ‘@’ symbol, followed by an action. Clients can join a room using the action word ‘join’ followed by the room number they wish to join. The server will display a message to confirm the client has been successfully added to the room number specified.

Example:

<client> : @join *<room id>*

<server> : You have joined room *<room id>*!

## Leave a Room

To leave a room, the client must enter a command. All commands are prefaced by the ‘@’ symbol, followed by an action. Clients can leave a room using the action word ‘leave’ followed by the room number they wish to leave.

Example:

<client> : @leave *<room id>*

<server> : You have left room *<room id>*!

## Broadcast a Message

To broadcast a message, the client must first establish a connection to the server. Broadcasting a message will send text to all connected users. This is the default behavior when the client submits text without a command.

Example:

<client> : Hello, all!

*<username>* (# *<user id>*): Hello, all!

## Send a Public Message

To send a public message, the client must first establish a connection to the server and join a room. Once this is established, the client can enter a command to send text to all other users connected to that room. All commands are prefaced by the ‘@’ symbol, followed by an action. Clients can send a public message to a room by using the action word ‘room’ followed by the room number they wish to send a message to.

Example:

<client> : @room *<room id>* Hello, room!

*<username>* (# *<user id>*): Hello, room!

## Send a Private Message

To send a private message, the client must first establish a connection to the server. Once this is established, the client can enter a command to send text to a specific user. All commands are prefaced by the ‘@’ symbol, followed by an action. Clients can send a private message to a user by using the action word ‘user’ followed by the user number they wish to send a message to.

Example:

<client> : @user *<user id>* Hello, user!

*<username>* (# *<user id>*): Hello, user!

* 1. End a Connection

To terminate the connection to the server, the client can exit the GUI. At that point, the server will exit the infinite loop handling incoming connections. Once the server is closed, communication is disabled until the server is started again.

1. System Failures

The application is configured to handle system crashes gracefully. If the server crashes, a message is displayed to any connected clients stating that the connection has been closed.

If the client crashes, the server removes closed connections when a new client tries to connect, and it will be removed from its list of connected clients.