Request for Comments Sascha Strand
Dec. 2017

Catenary
A command-line mock of Slack

Status of This Memo

This is the first version of this document, very much open for revision.

Abstract

This is a protocol for a single-server implementation of a group messaging platform that defines a workspace of users and channels. The design is a hybrid of traditional IRC and that of the contemporary workplace group messaging platform Slack. While few of the defining features of Slack have been implemented—like persistence of messages in each channel or extensive emoji support—this protocol provides the channel-based group and individual messaging that could be readily extended into a more robust feature set.

Table of Contents

Section 1: Functionality

1.1 Platform

Catenary is a collection of workspaces and workspaces are collections of users who communicate in channels. Channels, like chat rooms, host multiple users at a time sending messages between one another.

In this implementation, there is only one workspace and it is open to the public. In this way, the protocol resembles that of IRC more than Slack.

Like Slack there are multiple channels a use may be a part of and a user can only be in one at a time. Unlike Slack, however, all channels are open to all users and there is no implementation of messaging a group separately from creating a channel.

1.2 Server

In this single server architecture, the server listens to messages from clients and acts on those messages based on their header ‘to’, ‘from’, and ‘body’ fields.

Even the graceful shutdown of the server is accomplished through sending the server a formatted message.

When a user joins a channel, an entry will be made in the server’s list of channels and channel members to store the user’s user ID under that channel. If that channel does not exist, the operation to join the channel will create it. When moving to a new channel, a user will be removed from the channel they were previously in, but if a channel’s user list reaches zero the channel does not disappear.

The message with its header form a JSON object transmitted as a binary string. Messages have no default text encoding, but in this implementation they are utf-8.

1.3 Client

The client is about as lightweight as possible. It renders the messages sent to it by the server and

sends messages. The only state it knows about the messaging platform as a whole is its own user ID and the current channel it’s in.

The character sequence ‘:::’ is used as the default special character sequence for the platform. A user can enter ‘:::’ at the prompt where they would normally send a message and receive instead a command prompt for the greater platform.

Here a user may request to join a channel (or, with the same operation, create a new channel), list all channels, list all users in the current channel, send a direct message, gracefully close, or send a message to the server to close the server.

This latter message has a mocked-in password field, though it is not implemented here in an even remotely secure fashion. Any client can turn off the server.

The user-level commands issued at the client command prompt are as follows:

- list or l

  This returns a list of all channels in the workspace and returns the user to the messaging interface for their current channel.

- join or j

  This command further prompts the user for a channel they would like to join or create and moves them to that channel.

- quit or q

  This command initiates a graceful shutdown of the client.

- users or u

  This command prints a list of all users in the given user's current channel as well as their IP addresses and port numbers.

- close server or cs

  This command sends a shut-down message to the server, which will naturally end all communication between users.

- direct message or dm

  This command prompts the user for the username of the other user they would like to message and for the content of a message. This message is then sent to the server which parses the header and forwards it to the recipient with an indication of who it came from and the fact that it is a private message.

Section 2: Messages

These messages are sent at the application level between the client and server. Because they are included in the 'to' field of the message format and channel names are also included in this field, channels cannot be named with these effectively reserved words.

In all cases, parameters to these operations are included in the 'body' section of the message and where there is more than one parameter, delimited by ':::'.

The 'from' section of a message is always the user ID of the client.

2.1. LISTCHANNELS
parameters: none

2.2. JOINCHANNEL
parameters: <channel to join>

2.3. LEAVEWORKSPACE
parameters: none

In this single-workspace implementation, this command notifies the server than the client is closing its connection and shutting down.

2.4. LISTUSERS
parameters: <current channel>

2.5. CLOSESERVER
parameters: <password>

2.6. DIRECTMESSAGE
parameters: <recipient user name>:::<private message content>

Section 3: Replies

In order to keep the client as light-weight as possible, the server processes the command-line appearance of each message entirely before sending it as a single string to be displayed by the client. Because of this, the client does not need to parse any message or code from the server.

In the even the server shuts down gracefully, it will send a message to all clients to this end. If the server crashes, the client will recognize its connection to the server is no longer active and will issue a message to the user indicating this.