                              minIRC
                         Client Protocol

Abstract

   The minIRC protocol is for use with text-based conferencing. The
   features supported by minIRC are guided by and are similar to the IRC
   protocol; however, the focus of minIRC is for learning how to develop
   a client/server protocol. So minIRC will not focus on duplication of
   the IRC protocol, rather, minIRC will be IRC-like in how the protocol
   is implemented and will have similar functionality from the point of
   view of the end user.

   This document defines the client protocol. It would benefit the
   reader to be familiar with the IRC protocol, [RFC2812].

Status of This Memo

   This Internet-Draft is submitted in full conformance with the
   provisions of BCP 78 and BCP 79.

   Internet-Drafts are working documents of the Internet Engineering
   Task Force (IETF). Note that other groups may also distribute
   working documents as Internet-Drafts. The list of current Internet-
   Drafts is at http://datatracker.ietf.org/drafts/current/.

   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."

   This Internet-Draft will expire on December 2, 2017.

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.

Table of Contents

1. Introduction

   minIRC is a minified version of IRC that is intended to satisfy the
   requirements for CS594 Internetworking Protocols at Portland State
   University. The services provided by an minIRC server for clients
   are connecting to the server, creating channels, listing channels,
   joining channels, leaving channels, listing members of the channels,
   support for multiple clients, sending messages by clients to a
   channel, joining multiple (selected) channels, sending distinct
   messages to multiple (selected) channels, disconnecting from the
   server, and being disconnected by the server. Some possible extra
   features include private messaging, secure messaging, file transfer,
   and a cloud connected server.

1.1. Terminology

In this document, the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" are to be interpreted as described in BCP 14, RFC 2119 [RFC2119].

2. The miniIRC Client Specification

2.1. Overview

The protocol that follows is intended to be used as a client connected as a user to a server. The communications described in this RFC will occur over port 10101. This message sending is necessarily asynchronous. Communication to the server from the Client or to the Client from the Server can happen at any time. Either the server or the Client can disconnect at any time. It is polite to close the connection upon disconnect.

2.2. Character codes

The miniIRC protocol uses UTF-8 character encoding.

2.3. Message Basics

Clients interact with the server via messages encoded as valid JSON objects (see [RFC4627]). Invalid JSON WILL be rejected with an error. JSON is a collection of key, value (or ordered values) pairs. The keys in the incoming JSON message will correspond to one the various commands discussed below, the values will correspond to the parameters, if any, of that command. If a command has no parameters, the value null WILL be passed instead.

Multiple commands WILL be passed as individual messages. Each message WILL be limited to a total of 512 bytes. Any message that exceeds 512 bytes WILL be rejected with an error.

What follows is an example JSON formatted message in string format as the server will see it after decoding the binary over the socket.

'{"PRIVATEMSG": {"USERS": ["Socrates", "Plato", "Aristotle"], "MSG": "We're having a party tonight! Someone tell Xenophon..."}}\n'

In this example, the user has sent a private message that has targeted 3 other users. Here PRIVATEMSG is a command that takes two parameters: USERS with value username or a list of usernames and MSG with the message to send. NOTE Any single quotes inside any strings

MUST be escaped. The stringified JSON above corresponds to the following JSON object:

```
{
    "PRIVATEMSG":{
       "USERS":[
           "Socrates",
           "Plato",
           "Aristotle"
        ],
       "MSG":"We're having a party tonight! Someone tell Xenophon..."
    }
}
```

The keys in the message can be in any order. For example, in the message string above, the MSG key, value pair could have come before the USERS key, value pair.

All example messages in this document will use the string form listed above. Furthermore, this document will use the following formatting for describing messages:

Command: <Command Name>

Parameters:

o   <Key>: <value description>

o   [<Key>: <value description>] (Optional)

The example used in this section would look like this:

Command: PRIVATEMSG

Parameters

o   USERS: <User nickname or list of user nicknames>

o   MSG: <Message to send>

If a parameter value is not required, it will have the word Optional next to it. The client can send either the relevant parameter or it can send a null value.

2.4. Status and Error Codes

The miniIRC server sends numeric error codes and short messages. The
error codes mimic the HTML error codes when possible.

o 100-level codes are informational status codes.

o 200-level codes are success codes.

o 300-level codes are not used.

o 400-level codes are client error codes.

o 500-level codes are server error codes.

Familiarity with HTML [RFC2616] might be beneficial.

3. Message Details

3.1. Connecting and Disconnecting

The commands in this section describe how the user connects to and
disconnects from the server.

3.1.1. Logging into the server

Command: LOGIN

Parameters:

o NICK: <user nick>

The JOIN message binds a nickname value to a given client. If the
nickname is already taken, an error will be returned.

Responses:

```
200 OK
409 NICK CONFLICT
```

Examples:

```
'{"LOGIN": {"NICK": "Leibniz"}}\n'
```

3.1.2. Logging out of the server

    Command: QUIT

    Parameters:

    o null

    The user SHALL make every effort to quit properly so that the server
    can close the TCP connection properly.

    Responses:

        None

    Examples:

        '{"QUIT": null}\n'

    Note:

    The QUIT command will notify all channels that the client has joined
    that the client has quit.

3.2. Channel operations

    This section discusses the channel-related actions available to a
    client.

3.2.1. List all channels

    Command: LIST

    Parameters:

    o FILTER: <Regex filter> Optional

    The LIST command will display a listing of all the channels available
    to join. The user can pass a regex string to use as a filter of the
    listings.

    Responses:

        200 OK
        500 INTERNAL SERVER ERROR

    Example:

'{"LIST":{"FILTER":"$[a].*^"}}\n'

3.2.2. Join a channel

Command: JOIN

Parameters:

o CHANNELS: <list of channel names>

This command will associate a user's nickname with one or more
channels. If a valid channel name is not passed, a warning will be
returned alerting the user that the given channel name is invalid.

Responses:

```
200 OK
401 MALFORMED REQUEST
402 USER ALREADY IN CHANNEL
404 NOT FOUND
```

Examples:

```
'{"JOIN":{"CHANNEL":["funny"]}}\n'
'{"JOIN":{"CHANNEL":["random", "funny", "#mods"]}}\n'
```

3.2.3. Leave a channel

Command: LEAVE

Parameters:

o CHANNEL: <list of #channel names>

This command will disassociate a user's nickname with one or more
channels. If a valid channel name is not passed or if the user
nickname is not associated with a channel name, a warning will be
returned alerting the user that the given channel name is invalid.

Responses:

```
200 OK
401 MALFORMED REQUEST
403 USER NOT IN CHANNEL
404 NOT FOUND
```

Examples:

       '{"LEAVE":{"CHANNEL":"funny"}}\n'
       '{"LEAVE":{"CHANNEL":["random", "funny", "#mods"]}}\n'

3.2.4. Create a channel

   Command: CREATECHAN

   Parameters:

   o NAME: <Channel name>

   This command will create a channel with the name supplied as a
   parameter. If the channel already exists, this command will return
   an error. If a list of channel names is passed, this will only
   create the channel of the first name in the list and the rest of the
   list will be ignored.

   Responses:

       200 OK
       409 CHANNEL EXISTS

   Examples:

       '{"CREATECHAN":{"NAME":"funny"}}\n'

3.2.5. List the members of a channel

   Command: USERS

   Parameters:

   o NAME: <Channel name>

   o FILTER: <Regex filter> Optional

   This command will list all of the user nicknames that are associated
   with a given channel name. If a list of channel names is passed,
   only the first name in the list will be used and the rest will be
   ignored.

   The regex filter is not required. If passed, the list of names will
   be filtered based on that regex.

   Responses:

```
      200 OK
      401 MALFORMED REQUEST
      404 CHANNEL NOT FOUND
```

   Example:

      '{"USERS":{"NAME":"general","FILTER":"^[A|a]{2}.*$"}}\n'

3.3. User messaging operations

   This section deals with the operations that deal with sending public
   messages to channels and private messages to users.

3.3.1. Send a message

   Command: SENDMSG

   Parameters:

   o NAME: <channel or user nick or list of channel names or user
      nicks>

   o MESSAGE: <Message to send>

   This command sends a message to a channel, to many channels, to
   another user, or to multiple other users.

   Messages sent to channels will be displayed for all user nicknames
   that are associated with that channel. Messages sent to users will
   be displayed in a private messaging channel. The private message
   channel will only ever have at most 2 users, so even if a list of
   users is passed as a parameter.

   Responses:

```
      200 OK
      401 MALFORMED REQUEST
      404 USER NOT FOUND
```

   Example:

      '{"SENDMSG":{"NAME":["#funny", "Descartes"],"MESSAGE":"Hello!"}}\n'

3.4. Server actions

   The server actions are geared towards client maintenance. A server
   should be able to kick users out. Additionally, a server must be
   able to query clients on a regular schedule to programmatically

verify they are still connected. To facilitate this, this section
must also detail the client response to this verification.

3.4.1. Kick a user

Command: KICK

Parameters:

o NICK: <User nickname or list of user nicknames>

o MSG: <Message> Optional

Kicks a user or a group of users off a server.

Responses:

    200 OK
    401 UNAUTHORIZED
    404 USER NOT FOUND

Example:

    '{"KICK":{"NICK":"Leibniz","MSG":"Stop being a jerk!"}}\n'

Note:

The KICK command will notify all channels that the client has joined
that the client has quit.

3.4.2. Ping query

Command: PING

Parameters: null

This server query is designed to ensure that all clients that are
registered as active are actually active and have not crashed. Ping
queries must be responded to by a PONG response within 1 second, or
the user associated with that client will be kicked.

3.4.3. Pong response

Command: PONG

Parameters: null

This is the client response to a Ping query. This response must
happen within one second of the preceding Ping query.

4. Normative References

   [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate
              Requirement Levels", BCP 14, RFC 2119,
              DOI 10.17487/RFC2119, March 1997,
              <http://www.rfc-editor.org/info/rfc2119>.

   [RFC2812] Kalt, C., "Internet Relay Chat: Client Protocol",
              RFC 2812, DOI 10.17487/RFC2812, April 2000,
              <http://www.rfc-editor.org/info/rfc2812>.

   [RFC4627] Crockford, D., "The application/json Media Type for
              JavaScript Object Notation (JSON)", RFC 4627,
              DOI 10.17487/RFC4627, July 2006,
              <http://www.rfc-editor.org/info/rfc4627>.

   [RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H.,
              Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext
              Transfer Protocol -- HTTP/1.1", RFC 2616,
              DOI 10.17487/RFC2616, June 1999,
              <http://www.rfc-editor.org/info/rfc2616>.

Author's Address

   Michael Lane
   Portland State University

   Email: lane7@pdx.edu