SDChat Description
Draft 1.3

You will be writing a small client-server system. The system is modeled after chat programs. A user of SDChat has a screen name and a password. A screen name has to be unique.

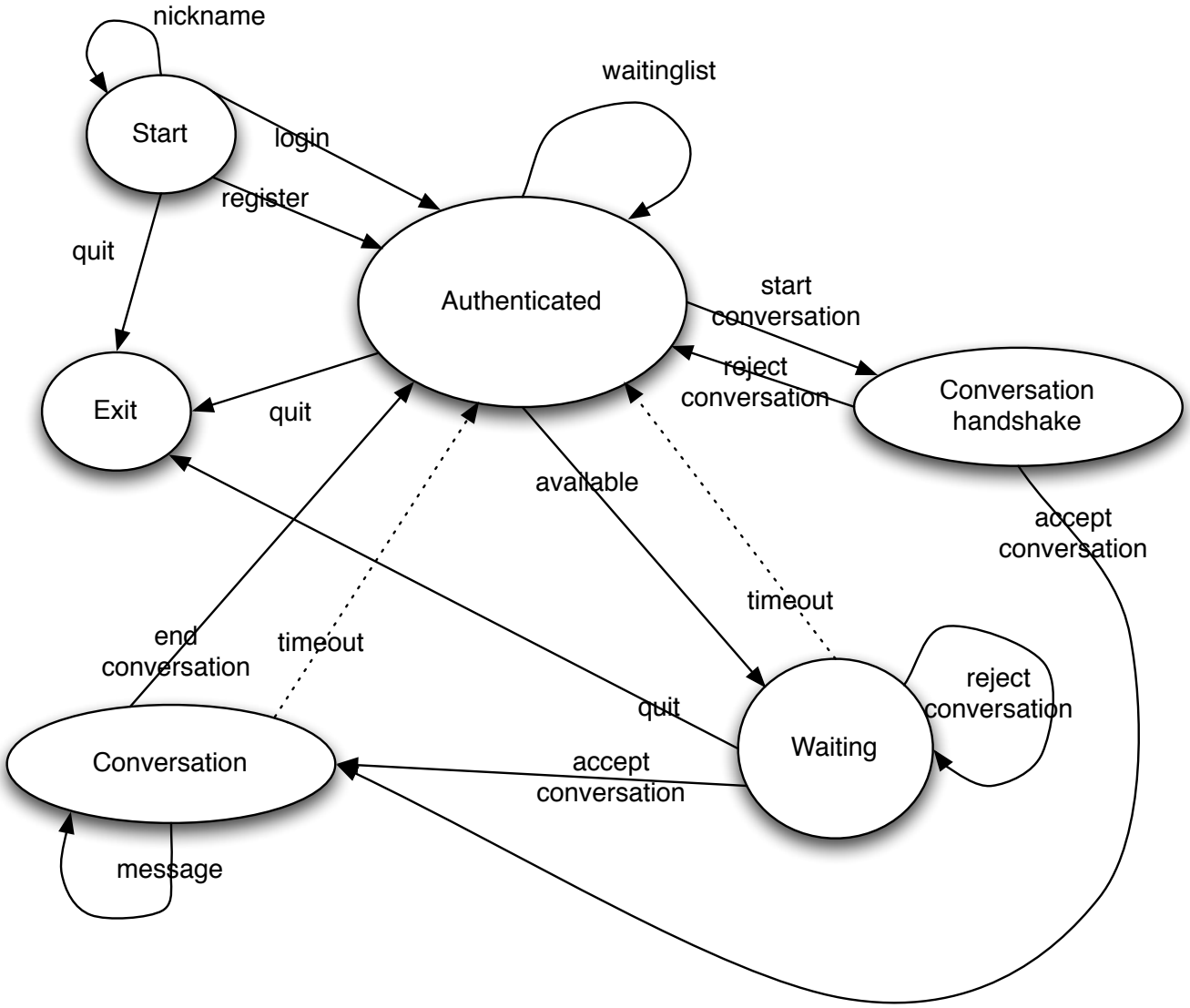
The protocol for the system is as follows. The client can send 11 commands to the server: "available", "login", "register", "nickname", "startconversation", "quit", "waitinglist", "acceptconversation", "message" , "rejectconnection", and "endconversation".

Connect State

Connections to the server have state. The connection starts in the "start" state. In this state only commands "login", "register", "nickname", and "quit" can be sent. After a successful "login" command the connection is in the "authenticated" state. In this state only the commands "available", "startconversation", "quit" and "waitinglist" can be sent. After the "startconversation" the connection enters the conversation handshake state. In this state the only valid commands are "rejectconnection" and "acceptconversation". After the "rejectconnection" the connection reverts back to the "authenticated" state. After the "acceptconversation" the connection enters the conversation state. In this state the only valid commands are "message" and "endconversation".

After the "available" command in the "authenticated" state the connection moved to the "waiting" state. The "waiting" state operates different than the other states. When client issues the "available" command it gets a reply from the server. The client then waits for another response from the server, which will be "requestconversation". So a client will get two incoming messages from the server in a row. If a client wishes to leave the "waiting" state before it receiving the "requestconversation" message it can send "quit" to the server, which ends the connection with the server.

States & Commands



Commands

Two characters: ":" and ";" are used as meta characters in commands. The meta character ":" is used to separate keys and values in a command. The meta character ";" is used to end command parts (headers) and to end a message. A command always ends in two ";" characters, one to end the last part and one to end the command. Whitespace (spaces, tabs, line feeds and character returns) can be be used immediately before and after a meta character. So the following versions of the login command are both valid.

```
login;nickname:foo;password:foo;;
```

```
login;
nickname:foo
;password : foo;;
```

Values, which follow the meta character ":", may contain ";" or ":". In order that these characters are not confused as meta characters they must be escaped with a "\". Since the character

"\" has special meaning it too must be escaped if it occurs in a value. If a user's password is "a:b;c\d" then it must be converted to "a\:b\;c\\d" before is it sent as part of the login command. The server removes the extra "\" characters.

Anytime there is an error on the server side the server return with an error message. The format is:

error:ErrorString;;

ErrorString is a description of the error. If it contains special characters it will be escaped.

All commands and keys in commands contain only ascii characters. Values in key value pairs can contain unicode characters. All communication is done using UTF-8.

login

Use to authenticate a user. The format of the command is:

login;nickname:aNicknameString;password:aPasswordString;;

Where aNicknameString and aPasswordString are strings that can contain any ascii characters. They need to be escaped if they contain ":", "\" or ";".

The server response with one of the following:

ok:success;;
error:No such user;;
error:Invalid password;;
error:Already logged in;;

If the command is successful the connection moves into the "authenticated" state. Otherwise it remains in the "start" state.

register

This command is use to register a new user. If the command is successful the connection moves to the "authenticated" state. If the command fails the connection remains in the "start" state. The format of the command is:

register;nickname:aNicknameString;password: aPasswordString;;

aNicknameString and aPasswordString are strings determined by a user. They need to be escaped if they contain ":", "\" or ";". If the selected nickname is already used the request fails. The server responds with one of the following messages:

ok:success;;
error:ReasonForFailure;;

ReasonForFailure may contain ":", "\" or ";". If it does the characters will be escaped.

nickname

This command is use to determine if a nickname is already used by a user. The format of the command is:

nickname:aNicknameString;;

If aNicknameString is available to be used the server responds with:

ok:available;;

If aNicknameString is being used by another user the server response with:

ok:used;;

available

This command is use to indicate that a user is available for a conversation with another user. The format of the command is:

available;;

The server responds with:

ok:success;;

The user is placed on the server's waiting list and the connection enters the "waiting" state. This is a list of people available for conversations with another user. After the server responds with "ok:success;;" the client must wait for another response from the server. This will be sent when another user sends the "startconversation" message to the server to start a conversation. The client will receive the "requestconversation" from the server. It can respond with either "rejectconversation;;" or "acceptconversation;;". See startconveration below.

The client can exit the "waiting" state by sending the command "exit" to the server. This will end the connection. If the client wishes perform any other commands it has to log in again.

waitinglist

This command is used to get the list of users on the servers waiting list. That is the list of users that have indicated they are available for a conversation. The format of the command is:

waitingList;;

The server responds with a list of the nicknames of the users on the servers waiting list. The format of the response is:

ok:N;nickname:nicknameString1;nickname:nicknameString2;nickname:nicknameStringN;;

where N is the number of nicknames returned. For each nickname returned there is a key-value pair returned. The key-value pair has the format "nickname:nicknameString;". Once a client finds nickname they wish to communicate with use the startconversation command to start the conversation.

startconversation

This command is used to initiate a conversation with another client. The format of the command is:

startconversation:aNicknameString;;

If aNicknameString is not in the waiting list on the server the server responds with an error message. If aNicknameString is in the waiting list the server sends a requestconversation message to the client with that nickname. The other client will respond with either:

rejectconversation;;
acceptconversation;;

The server will pass the other client's response back. This means the server response to a startconversation message is either "rejectconversation;;" or "acceptconversation;;". If the response is "acceptconversation;;" then the connection goes into the conversation state. The other client is removed from the servers waiting list. If the response is "acceptconversation;;" then the connection remains in the authenticated state.

When a client moves into the "waiting" state it remains in that state until it enters the conversation state or the exit state. If the client closes the socket, say by the client program crashing, the client will remain on the servers waiting list. If another client tries to start a conversation with this phantom user the server will respond with "rejectconversation;;" and remove the phantom user from the waiting list.

requestconversation

This message is sent to a client as part of the handshake to start a conversation between two clients. The first step is when a client, call it A, sends the "available**"** command to the server. This puts client A in the server's waiting list. After receiving the server's response client A waits for a requestconversation message from the server. This message is sent when another client, call it B, sends the startconversation message to the server with client A's nickname. The server then sends the message:

requestconversation;user:aNicknameString;;

to client A. aNicknameString is the nickname of client B. Client A then responds with one of:

acceptconversation;;
rejectconversation;;

depending on whether client A wants to enter into a conversation with client B. Client A response is forwarded to client B. If client A responses with acceptconversation;; the connection enters the conversation state. At this point client B can send the message command to the server. Client A response to the message command with another message command.

acceptconversation

When client A receives the requestconversation messages it responds with an acceptconversation message to accept the conversation request. There are two forms of the acceptconversation as shown below.

```
acceptconversation;;
acceptconversation;host:IpAddress;port:portNumber;;
```

The first version of the message is sent if client A wants the conversation messages sent through the server. The second version is sent when client A wants the conversation messages sent directly to it. It provides its IP address and port number that it will be listening on for messages from the other client. When the client sends this message its connection with the server is placed in the authorized state. If the format of the message is not valid the server sends an error message to client A, which can respond again.

message

When the connection is the conversation state either client in a conversation can send the message command. Let the two clients in the conversation be called A and B. Client A sends a message to client B using the command:

```
message;text:aString;;
```

where aString is the text of the message. The server adds client A's nickname and a time-stamp to the message so client B receives the message:

```
message;text:aString;sender:clientANickname;time:02/08/2010 20\:13\:37;;
```

The format of the time-stamp is dd/mm/yyyy hh:mm:ss, where dd is the date of the month in two digits, mm is the month using two digits and yyyy is the year using four digits. The date and time are separated by one space. The time is in 24 hour time. The hour is given in either one or two digits. The hour, minutes and seconds are separated by a ":". Since the time is a value in a key-value pair the ":" character has to be escaped.

Client B responds with either a message or an endconversation.

While client A & B are corresponding one of the client programs could close their connection without ending the conversation. This could happen if a client program crashes. The current server only detects the dead connection when it tries to send a message to the dead connection. So if client A crashes when client B sends a message to client A the server will detect the

dead connection. The server will respond to B with "endconversation;;". However if client A crashes after accepting a message from B but before responding the server will not detect the dead connection and client A will be left waiting for a message.

endconversation

When two clients are in the conversation state either client ends the conversation by sending the message:

endconversation;;

The server forwards the message to the other client without modification. Clients can only send the endconversation in response to a message command from the server. The connection then reverts back to the authenticated state. Neither client is in the waiting list after leaving the conversation state.

quit

The format of the "quit" command is:

quit;;

The server responds with result below and closes the connection with the client.

ok:quit;;

Server timeouts

When a client connects to the server it has 5 minutes to send a command. If the client does not send a command in that time the server will close the connection. If a client is idle for more that 50 minutes while in the "waiting" state the client connection is moved to the "authenticated" state. If there is no communication between two clients in a conversation for 50 minutes the conversation is ended with "endconversation;;" send to both clients. Both client connections are moved to the "authenticated" state.

Sample Interaction

The interaction below assumes that the client with nickname foo has already registered and the client with the nickname bar has already logged on and issued the available command.

Client foo Command	Server Response
login;nickname:foo;password:foopass;;	ok:success;;
waitingList;;	ok:1;nickname:bar;;
startconversation:bar;;	acceptconversation;; (assuming bar accepts)

Client foo Command	Server Response
message;text:Hello;;	message;text:Message from bar: sender:bar;time:02/08/2010 20\:13\:37;;
quit;;	ok:quit;;

Revision Notes

Changes from Draft 1.2 to 1.3 (Mar 8, 2010)

- Added the section on acceptconversation and introduced the second version of the acceptconversation message.

Changes from Draft 1.1 to 1.2 (Feb 23, 2010)

- Added Server Timeouts

Changes from Draft 1.0 to 1.1 (Feb 22, 2010)

- Added State diagram
- Added "waiting" state
- Added discussion about clients crashing in waiting state and in conversation state. See startconversation and message commands.
- Changed server response to nichname request from "ok:success;;" to "ok:available;;"