Final Project Report

***Title**: Serious Messenger*

***Submitted by**:*
Corey Faibish - 100764177
Kevin Scroggins - 100679071

***Submitted to**:*
Michel Barbeau
Carleton University

TABLE OF CONTENTS

LIST OF FIGURES

1. INTRODUCTION

Currently existing implementations of instant messengers include popular clients like MSN (Windows Live Messenger), Pidgin, aMSN, Adium, and much more. These clients are programmed in languages like TCL/Tk (Tool Command Language / Toolkit), Python, Ruby, C++, and C. Although most mainstreamed clients do not, we have decided to design ours, called Serious Messenger, in Java.

1.1 Background Information

Serious Messenger is a cross-platform instant messaging program developed in Java. This program provides a method for real-time direct message communication. The user's text is transmitted over the network to the server, and then from there to the client that the user is trying to communicate with.

1.2 Project Goal

The goal of the project was a fully functional peer-to-peer (P2P) instant messaging client programmed in Java. This includes message communication directly between two clients, while leaving the server just for back-end authentication and data storage using a database. Unfortunately, the actual end result differs from our project's goal, as will be outlined in the next section.

1.3 Result Summary

The project result was slightly different from the goal that was set out at project start. We ended up using a client-server architecture for our networking components, as opposed to P2P. Now, the server, in addition to storing data and authenticating clients, acts as an intermediary for all transmissions between clients. The result has functionality for clients to create accounts, sign in/out, viewing, changing, and deleting account properties and user profile information, and obviously message sending/receiving, to name a few. As well, our program offers a server side with features such as database monitoring and manipulation, system and command logs, signal interpretation and forwarding, and data storage.

2. PROJECT DETAILS

Serious Messenger is a multi-threaded, cross-platform, client-server messaging program. It was programmed in Java, and used no existing libraries aside from the basic packages that Java comes with. Although we looked into things like the Extensible Messaging and Presence Protocol (XMPP) for a protocol implementation, and Juxtapose (JXTA) for a P2P implementation, we decided we'd rather go at it ourselves.

This section includes information explaining various project aspects.

2.1 General Program Elements

The program has two primary components: the client, and the server. A client is launched by a user, and there can be any number of clients connected to one instance of the server at the same time (a 1:N implementation). Meanwhile, the server is launched by the network administrator to not only provide the networking capabilities of the program, but so as to monitor for issues that may come up while the program is running.

2.1 The Client

The client contains features such as the ability to create an account, sign in/out of accounts, change account properties like password and nickname, add/delete contacts from a user's contact list, block/unblock a contact's ability to communicate with the user, have conversations with contacts, view, edit, and create user profiles containing personal information. All of these features can be accessed from the client side, although, most require to be authenticated by the server before they get broadcasted to other contacts on the user's contact list.

2.2 The Server

The server contains features related to the database and networking aspects of our program. It provides functionality for mentoring the database, resetting, creating, and deleting tables, and modifying values in the tables. Also, it contains a command and a system log for easy debugging from the network administrator's side. In addition, the server offers signal interpretation, processing, and forwarding to the client the signal was intended for.

2.3 The Database

We implemented the storage database using SQLite, in combination with Java Database Connectivity (JDBC) to allow the java code to interact with the database.

The database consists of four tables: User Data (for account-relevant info), User Profile (for personal information), User Contact (for keeping track of who's a contact of who), and User Group (for maintaining what groups a user has). To examine the layout of our tables, please see Appendix A.

Initially, we were using SQL Server 2008 for our database. We switched to using SQLite because it was much simpler, and because it was fully integrated, allowing our server to be multiplatform.

3. IMPLEMENTATION DETAILS

This section provides a description for implementations of various aspects of Serious Messenger.

3.1 GUI (in brief)

Since the GUI was not the focus of this project, there are various unimplemented elements of our GUI that have back-end functionally ready to go. However, in brief, the interface for Serious Messenger was designed in two parts, the Server and the Client. Both were designed in NetBeans IDE, using its built-in GUI editor, and then ported over into Eclipse IDE where we did the rest of the coding. They utilize the Java Swing package for their GUI elements and AWT for events and the like.

3.2 Queuing System

Both the client and server packages come with queuing systems, each with one for input and one for output. Our queues are implemented using ArrayDeque from the Java Collection Framework. The queue has no size restriction, and increases in size when required to do so.

The queue runs in a loop. As long as the client and server are connected, and the queue is not empty, it removes the top signal from the queue and processes it, and does the associated action.

Also, the queue has an interval set to 50 milliseconds, which is how frequent the queue processes a signal from itself. Once a queue has processed and finished executing the associated task of a signal, it is put to sleep for the queue interval, which in our case is 50 ms.

3.2 Signals

Serious Messenger uses signals to communicate with the server for events that require authentication, or to be transmitted to other clients. Each signal is capable of computing it's own checksum based on its specific needs, which includes accounting for all local data variables. As well, each signal can read from and write to a byte stream, and write to a data output stream. Signals also have a global variable, which defines their size in bytes. Signal sizes vary, they come in two flavors: fixed-length and arbitrary-length signals. Except for the *ContactList* and *Message* signals, all are of fixed-length. The only difference is that for arbitrary-length signals, the length variable is only the initial length; it can grow to any size. These two signals are arbitrary-length because there is no minimum or maximum size for them. Also, to clarify, every signal contains different information because they each have different purposes.

For a full list of the signal types utilized by Serious Messenger, please view Appendix B.

An example case is described in Figure 1:

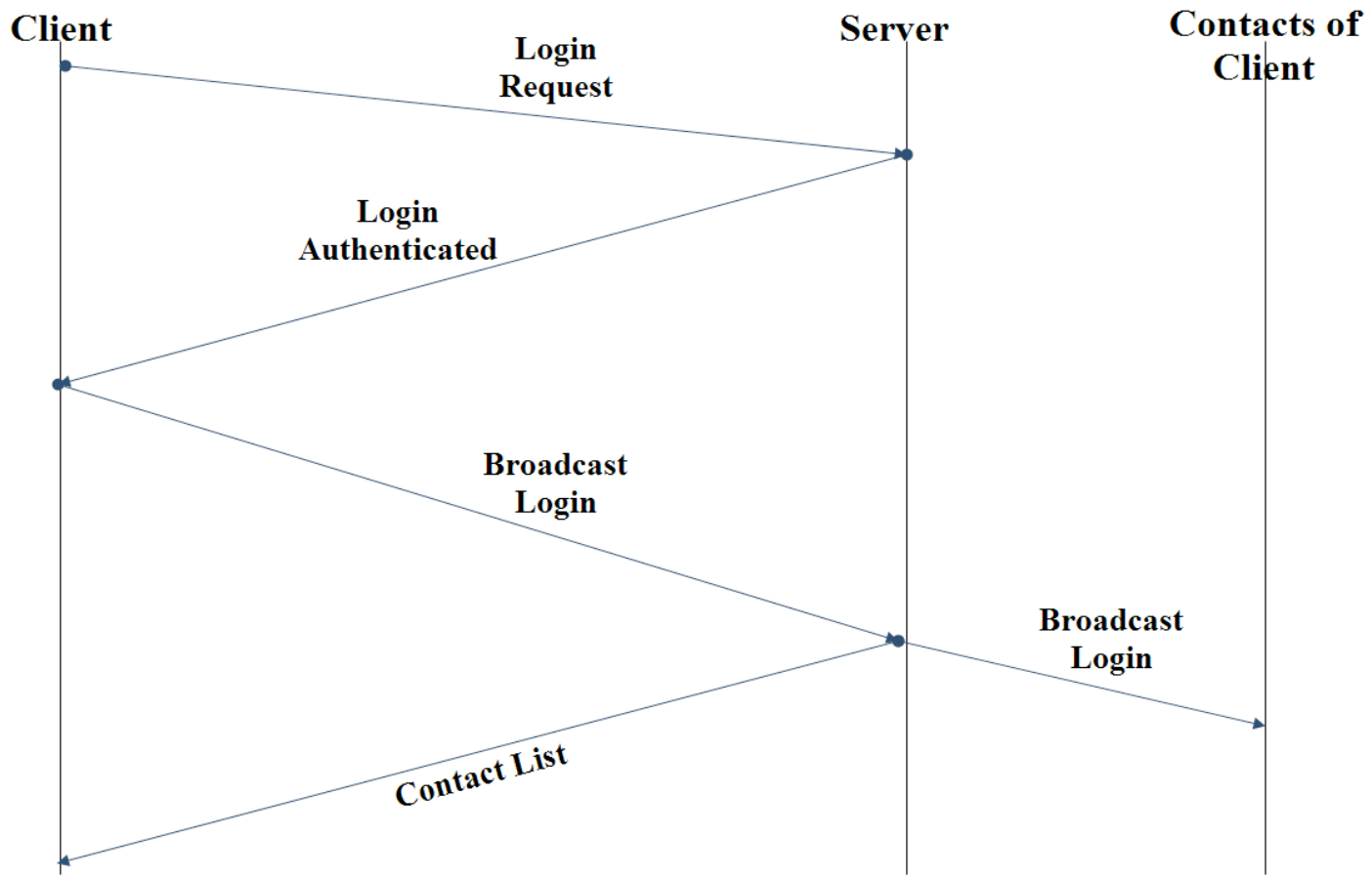

Figure 1 - Sample case: authenticating login and broadcasting to clients.

3.3 Determining Connectivity

In order to ascertain connectivity, we have employed a Ping / Pong system. In order to find out if the connection between the client and server is still active, one end will send a *Ping* signal. Then, the other end must respond promptly with a *Pong* signal, otherwise the connection is terminated. It should be noted that both the server and client are constantly using Ping / Pong to verify their connection.

There are two time intervals relevant to the Ping / Pong system. One is the interval between when you receive a *Pong* and when you send your next *Ping*. This is known as the ping interval, and is set to 5 seconds. The second is the period of how long you wait for a response. This is called the connection timeout interval, and is set to 7.5 seconds.

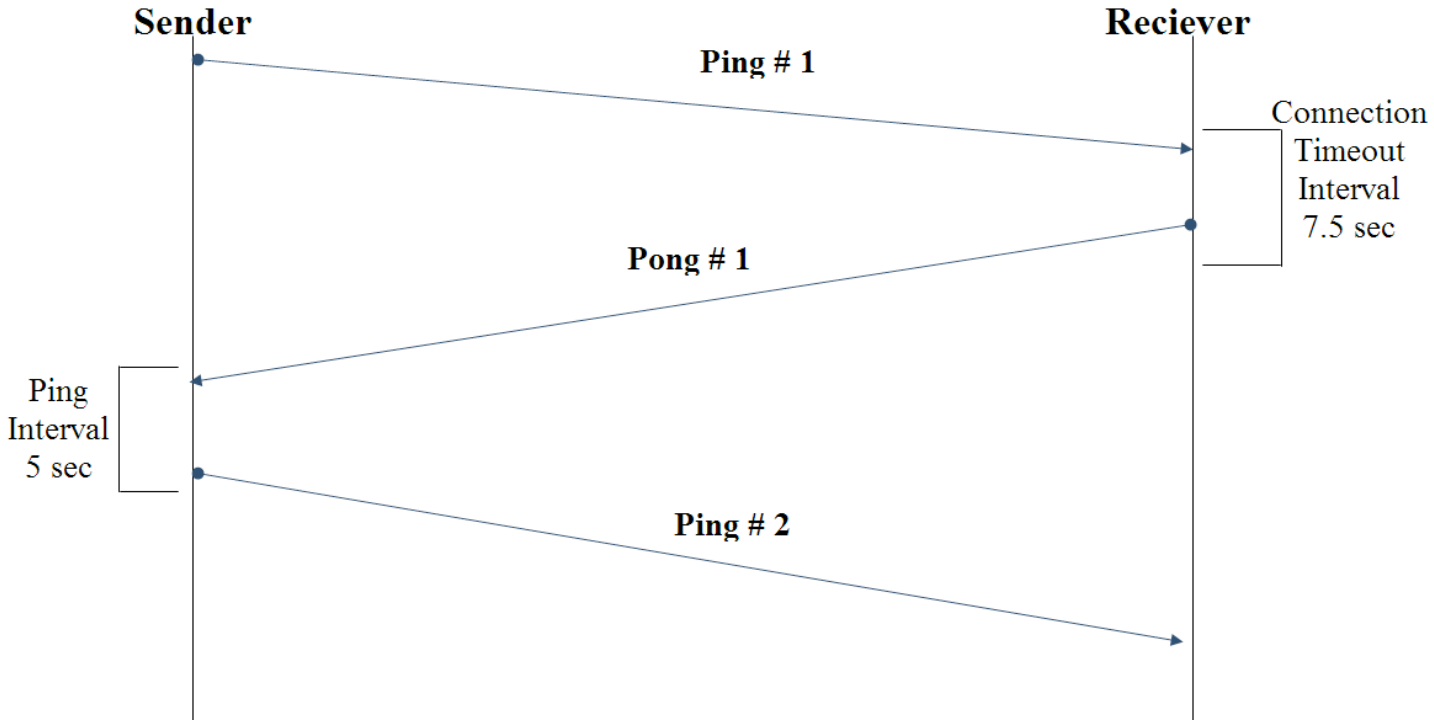

Figure 2 - Sample Ping / Pong case.

Figure 1 displays an example case of the Ping / Pong system. In this example, we see how when a *Ping* is sent, it has a 7.5 sec interval for the corresponding *Pong* to be sent. Then, we see the responding *Pong* received back on the sender's end. Next, the sender waits the 5-second ping interval, and then sends its next *Ping*, and the cycle starts again.

4. EVALUATION OF RESULT AND SUGGESTIONS FOR IMPROVEMENT

The current version of Serious Messenger is complete as per our deadlines. It possesses, most importantly, the functionality for sending/receiving messages between clients, amongst other various client and server side features.

Unfortunately, we didn't get a chance to implement everything we'd hoped for. The following is a list of possible future features of Serious Messenger:

- Display Pictures
- Profiles
- Contact List Searching
- Groups and Group Tags
- Enhanced GUI customizability
- Enhanced Key and Mouse listeners and more event interactions overall
- Nicer look & feel
- File transfer over a P2P connection
- Ability to work on public networks

APPENDIX A – DATABASE

This appendix contains table representations of our database, as well as description of what each entry's purpose is.

Please note:

- If an element is underlined, then that means it is a primary key of that table.
- If it says "References …", this means that this element is a foreign key, referenced from another table.

User Data	**Description**
UserName	User's unique account name
Password	User's password
NickName	User's display name
PersonalMessage	User's personal message
Status	User's current status (i.e. Away, Busy, etc.)
LastLogin	Last time user logged in
JoinDate	Date user created account

User Group	**Description**
UserName	References UserData.UserName
GroupName	Name of user-created group

User Profile	**Description**
UserName	References UserData.UserName
FirstName	User's first name
MiddleName	User's middle name
LastName	User's last name
Gender	User's gender
BirthDate	User's birthday
Email	User's email
HomePhone	User's home phone number
MobilePhone	User's mobile phone number
WorkPhone	User's work phone number
Country	User's country
StateProv	User's state/province
City	User's city
ZipPostal	User's zip/postal code

User Contact	**Description**
UserName	References UserData.UserName
Contact	A contact of a user; References UserData.UserName
GroupName	Group the contact belongs to; References UserGroup.GroupName
Blocked	Whether the user has blocked communications with the contact

APPENDIX B – SIGNAL TYPES

This appendix contains a list, and brief descriptions of, the various signals utilized by Serious Messenger.

Signal Name	Description
Ping	Used in conjunction with the Pong signal to determine if the connection to the client or server is still active or not. As soon as the other end of the connection receives a Ping signal, it must respond in a timely fashion with a Pong signal, or else the connection will be terminated.
Pong	The response signal to Ping. See description for Ping.
LoginRequest	Sent when a user attempts to log into the server. Contains the user's UserName and Password. No encryption is used.
LoginAuthenticated	Sent as a response to a LoginRequest signal from the server. Contains some relevant information about the user and whether the user was authenticated or not.
Logout	Sent by the client when signing out of the messenger program. Contains the user's name, but it is not actually used for anything.
BroadcastLogin	Sent after a user receives a LoginAuthenticated signal. The signal contains information about the user and is forwarded to all of their contacts which are online and do not have the user blocked.
Message	Represents a message being sent from one user to another. Contains the message number, the source UserName, the destination contact's UserName, the length of the message and the message itself which can be an arbitrary length, up to the maximum length of a message (currently 1024 characters).
UserTyping	Sent when a user begins or stops typing. Contains a boolean which defines whether the user is typing or not, the source UserName and the destination contact's UserName.
ChangeFont	When a user changes their font, a ChangeFont signal is sent which contains the source UserName, the destination contact's UserName and all the font style information including face, size, bold/italic/plain and colour.
ChangePassword	Sent by the client when they wish to change their password. Contains the user's UserName, their old password (for authentication) and their new password. No encryption is used.
PasswordChanged	Sent by the server as a response to the ChangePassword signal. Contains a boolean indicating whether the password was successfully changed or not.
AddContact	Sent by the client when they wish to add a contact to their contact list so they can communicate with them. Contains the UserName of the contact that they wish to add.
ContactAdded	Sent by the server as a response to the AddContact signal. Contains various information about the contact if they were successfully added, as well as a boolean indicating whether the user was added as a contact or not.
DeleteContact	Sent by the client when they wish to delete a contact from their contact list. Contains the UserName of the contact that the user wishes to delete.
ContactDeleted	Sent by the server as a response to the DeleteContact signal. Contains the UserName of the contact that was deleted and a boolean indicating whether the user was actually deleted or not.
BlockContact	Sent by the client when they wish to block a contact on their contact list. Contains the UserName of the contact they wish to block, and a boolean indicating whether they want to block or unblock the user.

ContactBlocked	Sent by the server as a response to the BlockContact signal. Contains the UserName of the contact that was blocked, a boolean indicating whether the contact is currently blocked or not and a boolean indicating whether the block was successful or not.
ChangeNickname	Sent by the client when they wish to change their NickName. Contains the user's UserName and their new NickName. The signal is then forwarded to all of the contacts on their list that are online and do not have the user blocked.
ChangePersonalMessage	Sent by the client when they wish to change their Personal Message. Contains the user's UserName and their new Personal Message. The signal is then forwarded to all of the contacts on their list that are online and do not have the user blocked.
ChangeStatus	Sent by the client when they wish to change their Status. Contains the user's UserName and their updated status. The signal is then forwarded to all of the contacts on their list that are online and do not have the user blocked.
CreateUser	Sent by the client when they wish to create a new account. Contains the user's UserName and Password.
UserCreated	Sent by the server as a response to a user sending a CreateUser signal. Contains a boolean indicating whether the account was successfully created or not. The connection between the client and server is terminated immediately after.
ContactList	Sent by the server after receiving a BroadcastLogin signal from a client. It is an arbitrary length signal and contains all of the relevant information regarding the contacts on the user's contact list.

APPENDIX C – USE SEQUENCES

The following sections describe various sequences of use of the interface of Serious Messenger.

You will probably start in a directory such as this:

Figure 1 - Directory view of program portable version (not the code, just 2 jars, a batch file, and the database file).

Unless you're running the software through an IDE such as Eclipse, in which case you will just run it through that interface, without needing to use our batch or jar files, and the database will be in the Java project folder.

Signing In

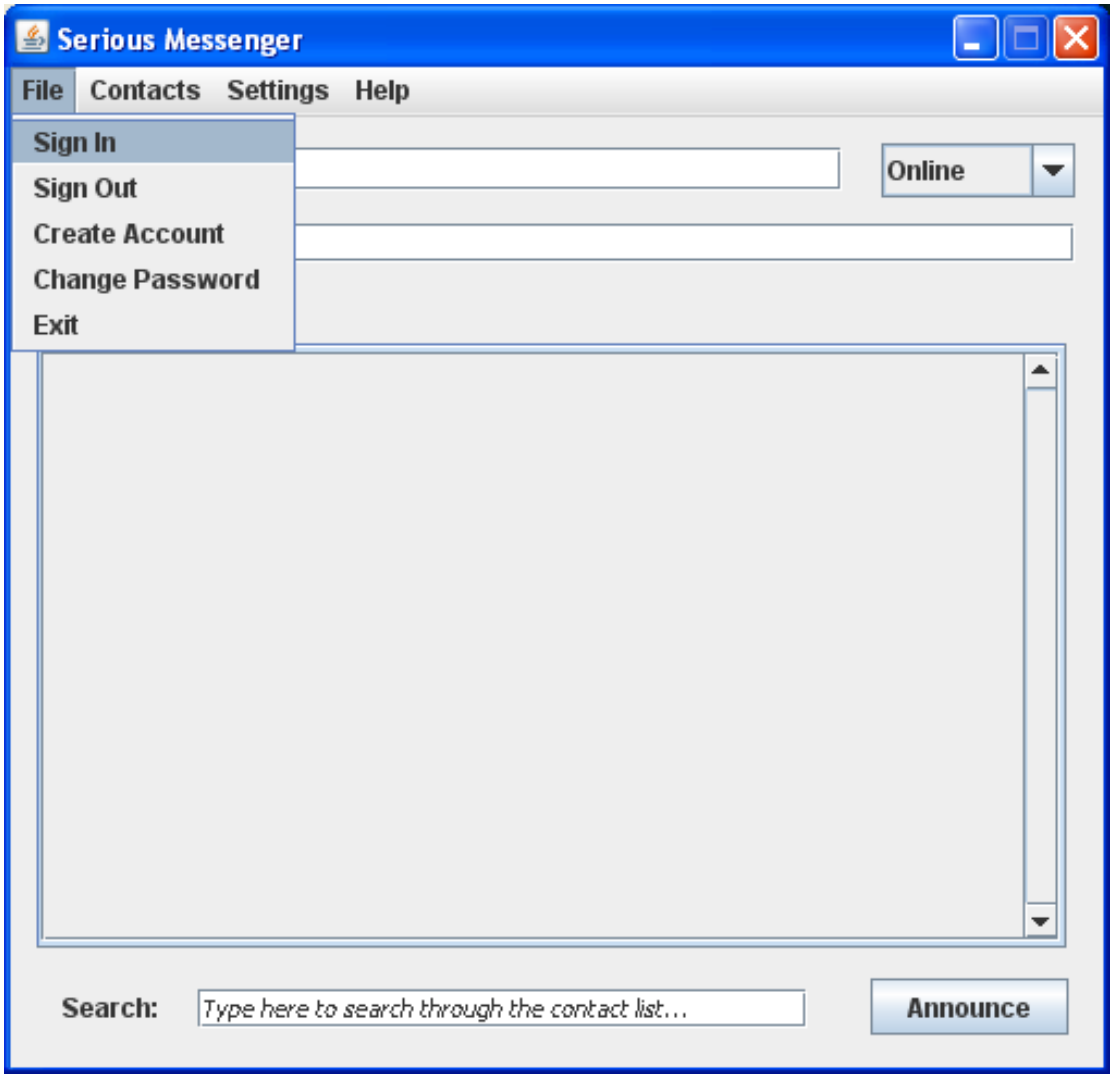

Figure 3 - Select "Sign In" from the File menu.

Then, follow the prompts as such:

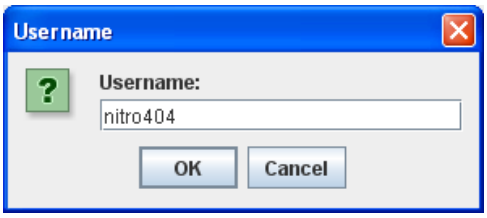

Figure 3 - Enter Username.

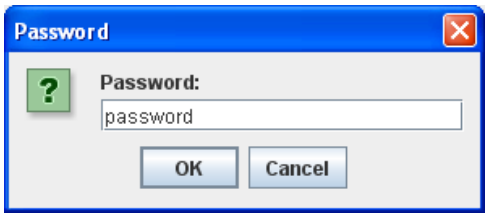

Figure 4 - Enter password.

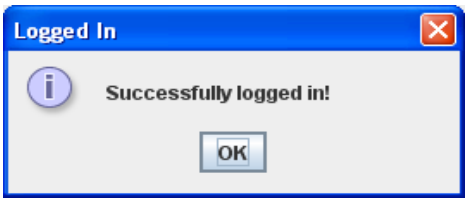

Figure 5 - Prompt notifying you of your successful login.

After this, you will arrive at the main window of the client:

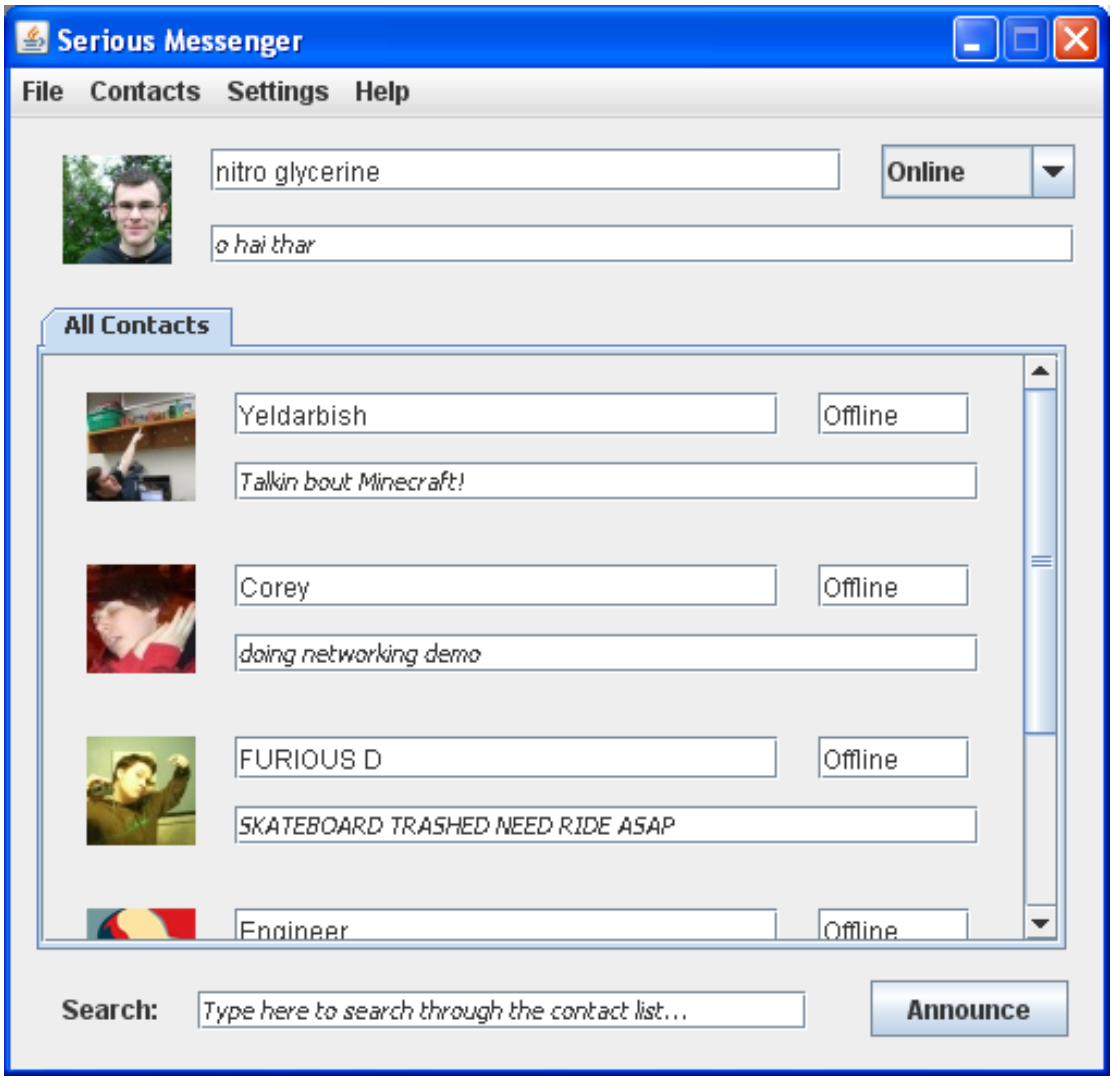

Figure 6 – Main window of Serious Messenger.

Starting a Conversation

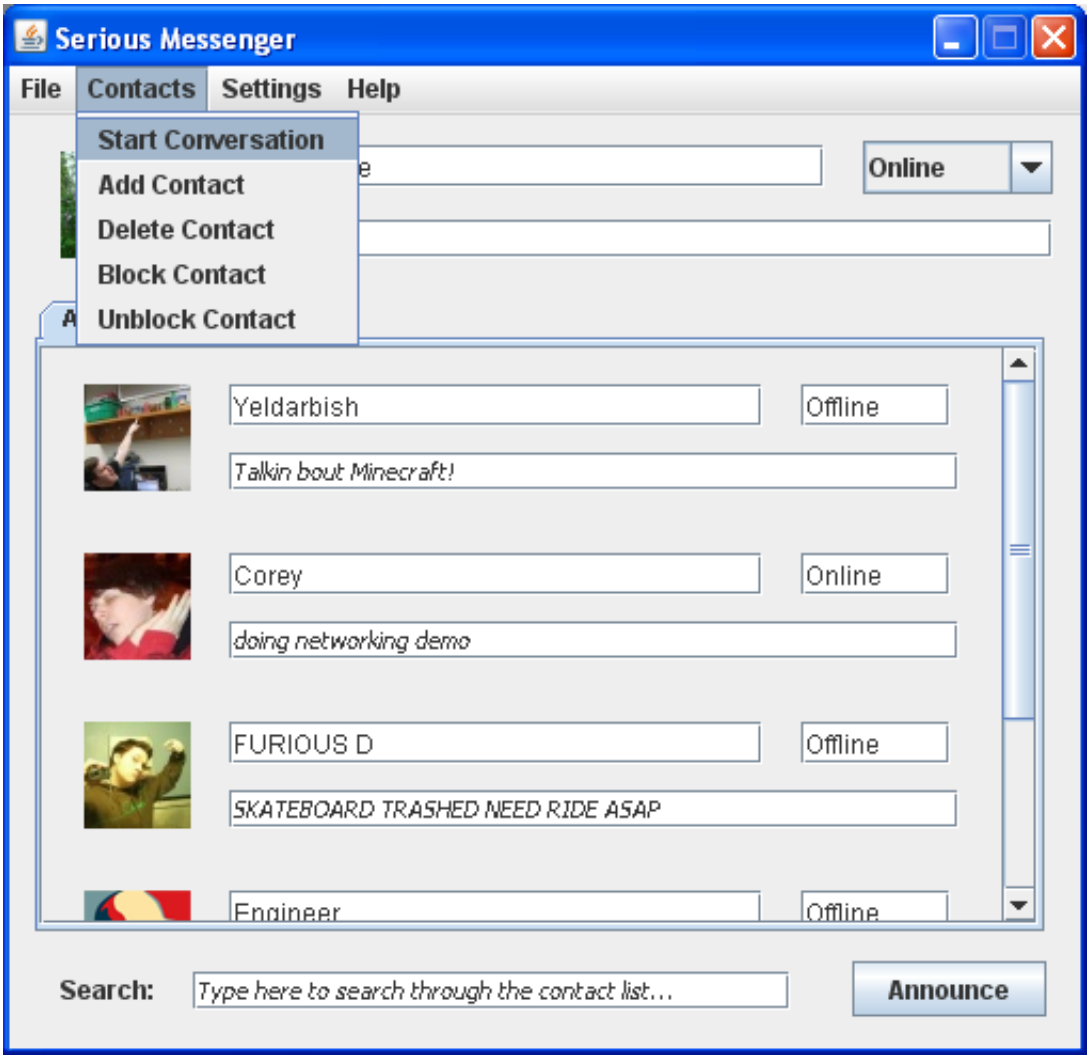

Figure 7 - Select "Start Conversation" from the Contacts menu.

Next, you will have to say whom you would like to talk with:

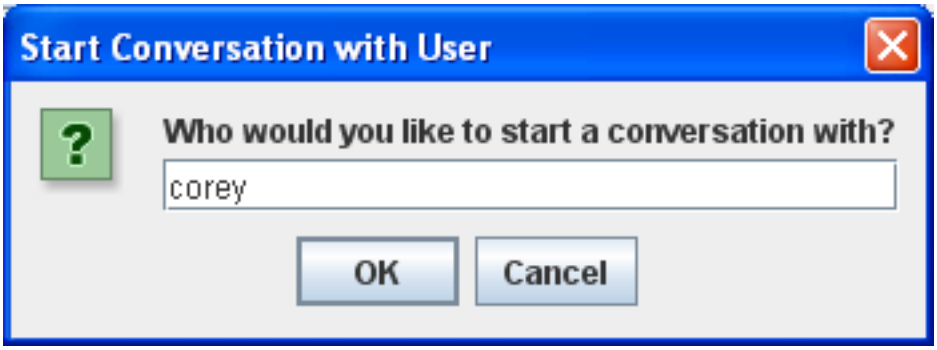

Figure 8 - Prompt for user to enter who to talk to.

Then, you will have a window like such:

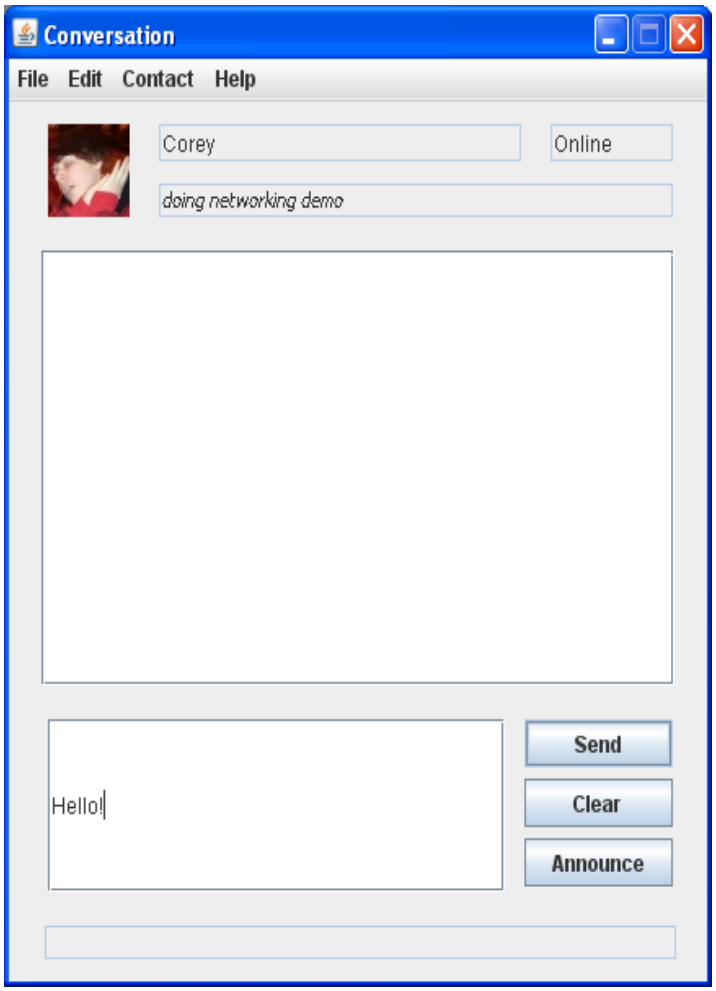

Figure 9 - Start typing a message to Corey.

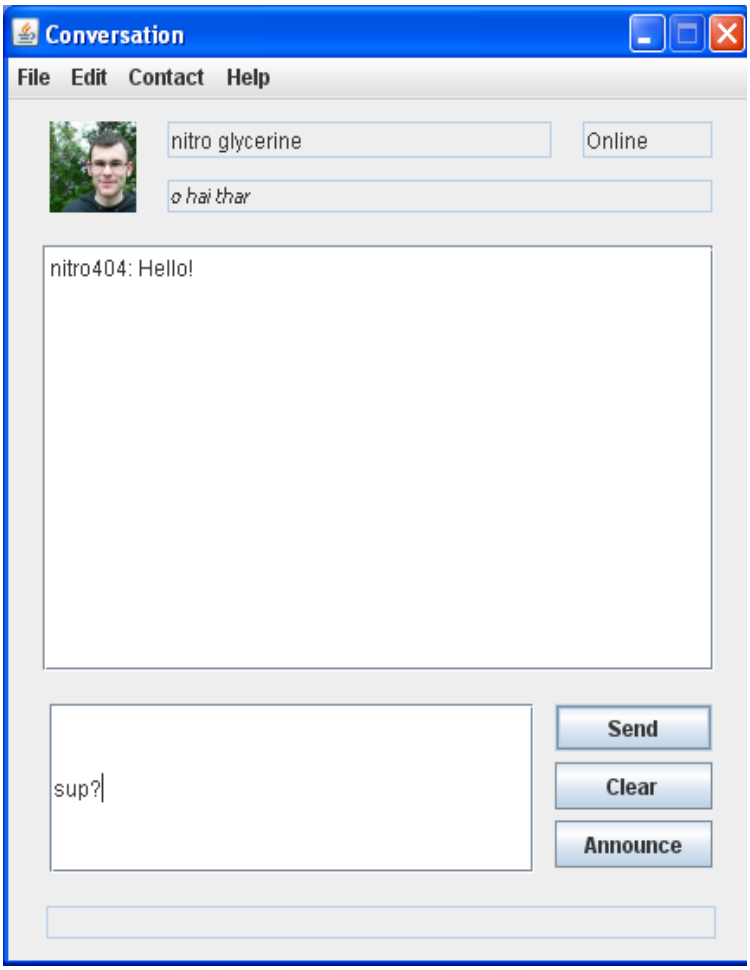

Figure 10 - A new conversation window appears on Corey's client.

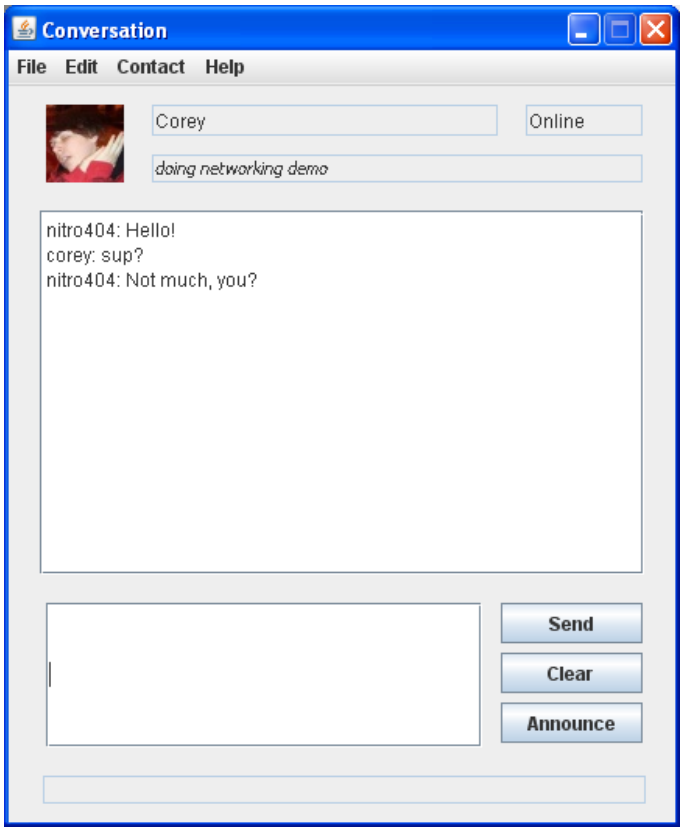

Figure 11 - A conversation is undergone.

Announce

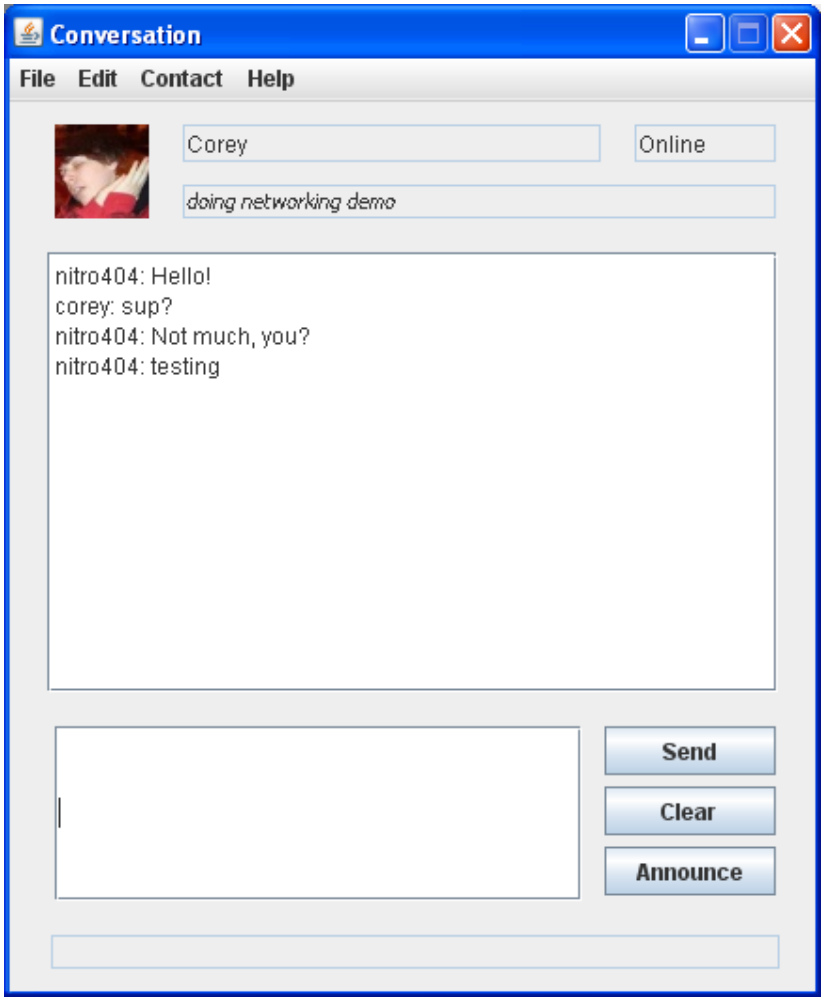

Figure 12 - Click the 'Announce' button.

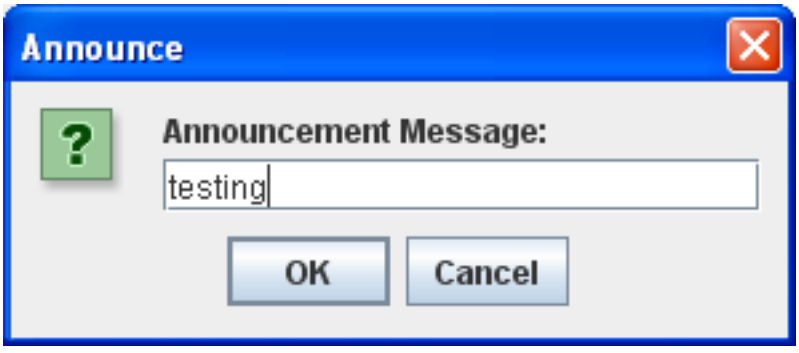

Figure 13 - You will be prompted for the message to send. Once done, click 'OK'.

Deleting a Contact

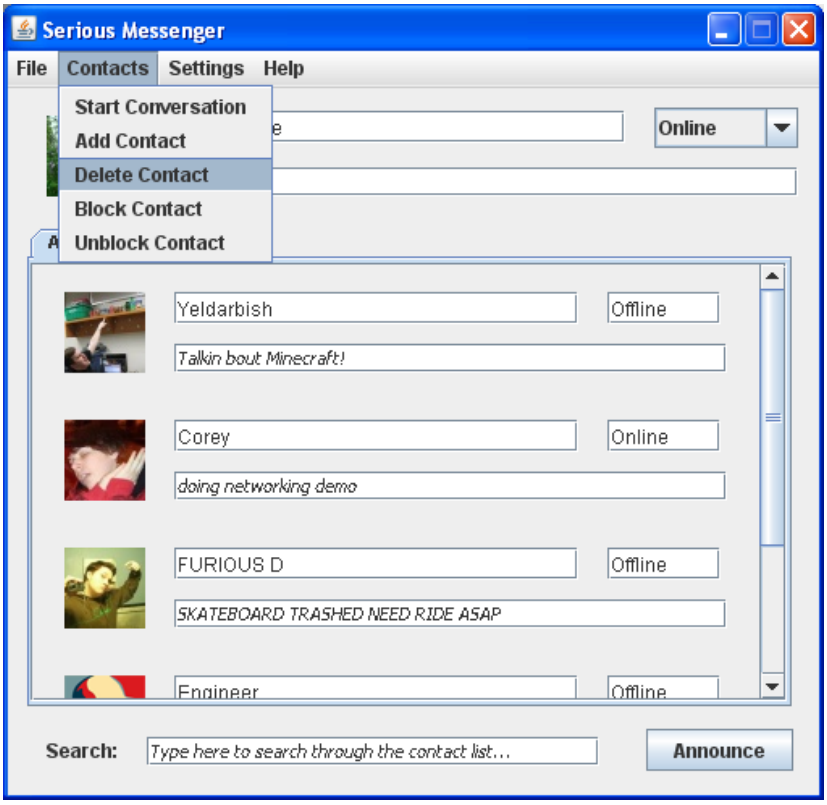

Figure 14 - Select 'Delete Contact' from the Contacts menu.

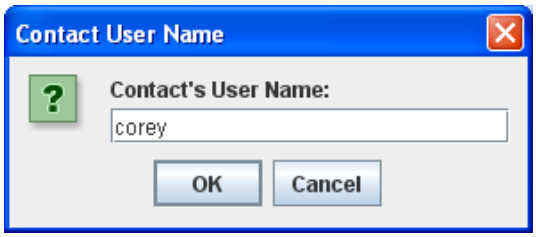

Figure 15 - Prompt to enter name of contact. When done, press 'OK'.

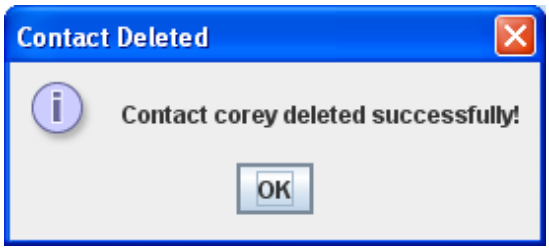

Figure 16 - Dialogue alerting you that the deletion was successful.

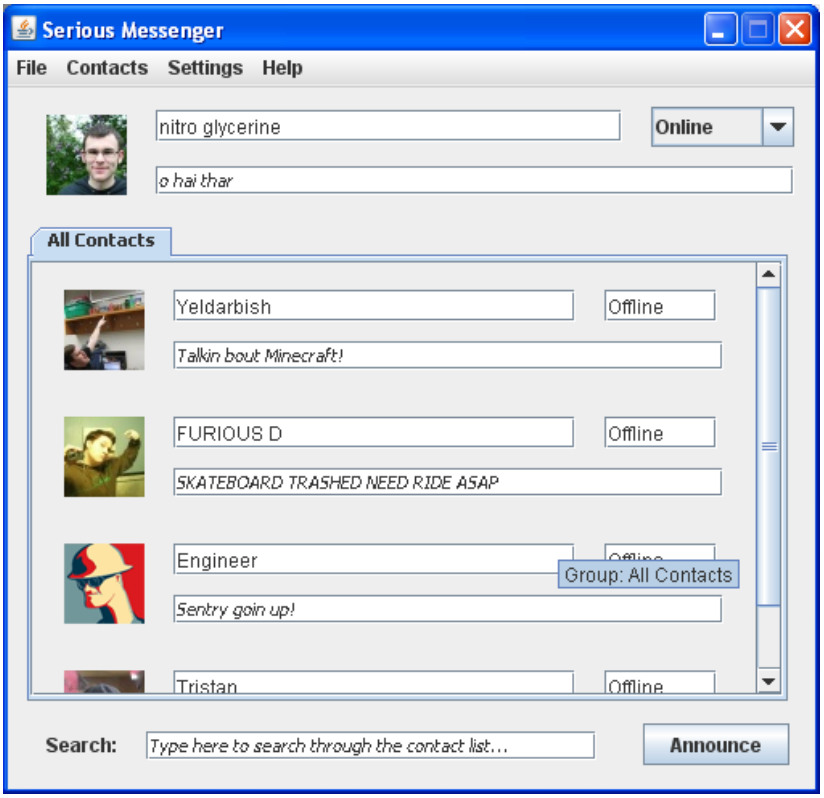

Figure 17 - The contact list after deleting contact Corey. Note, Corey is no longer in the list

Adding a Contact

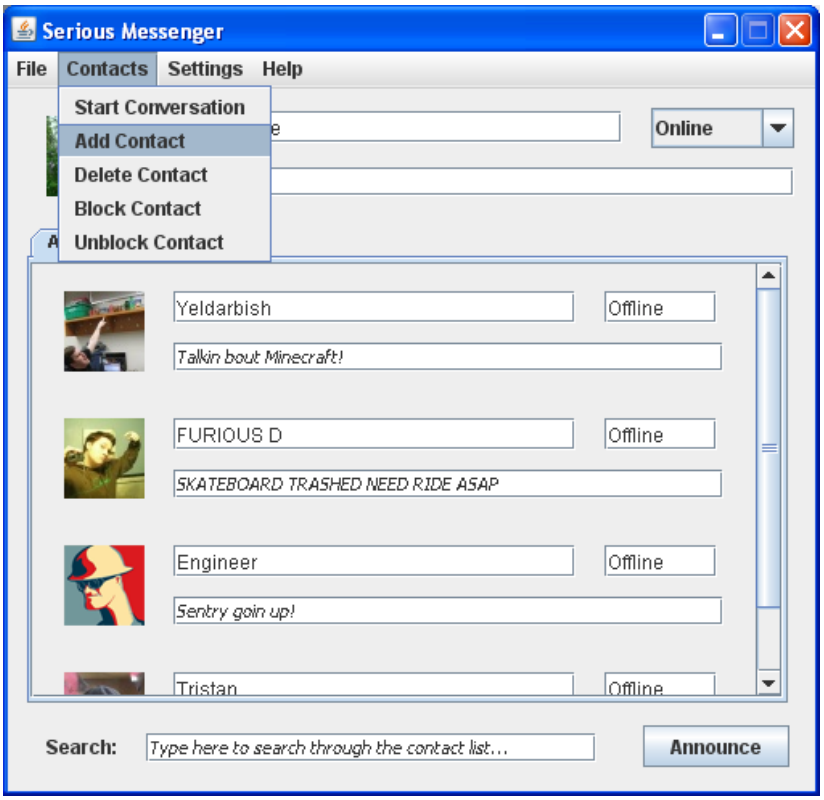

Figure 18 – Select 'Add Contact' from the Contacts menu.

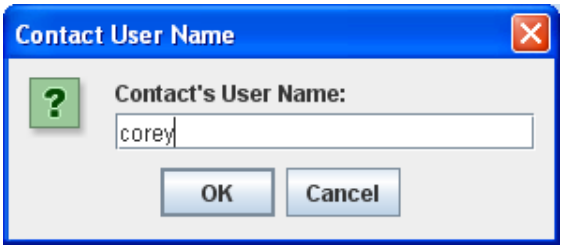

Figure 19 - Prompt to enter the contact's unique user name.

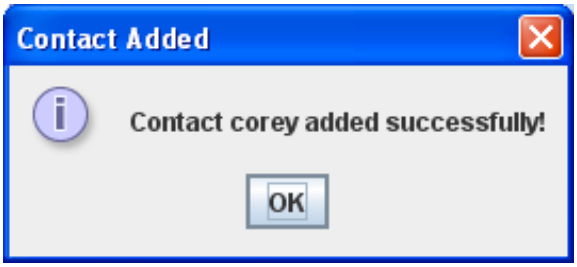

Figure 20 - Dialogue alerting you that the add was successful.

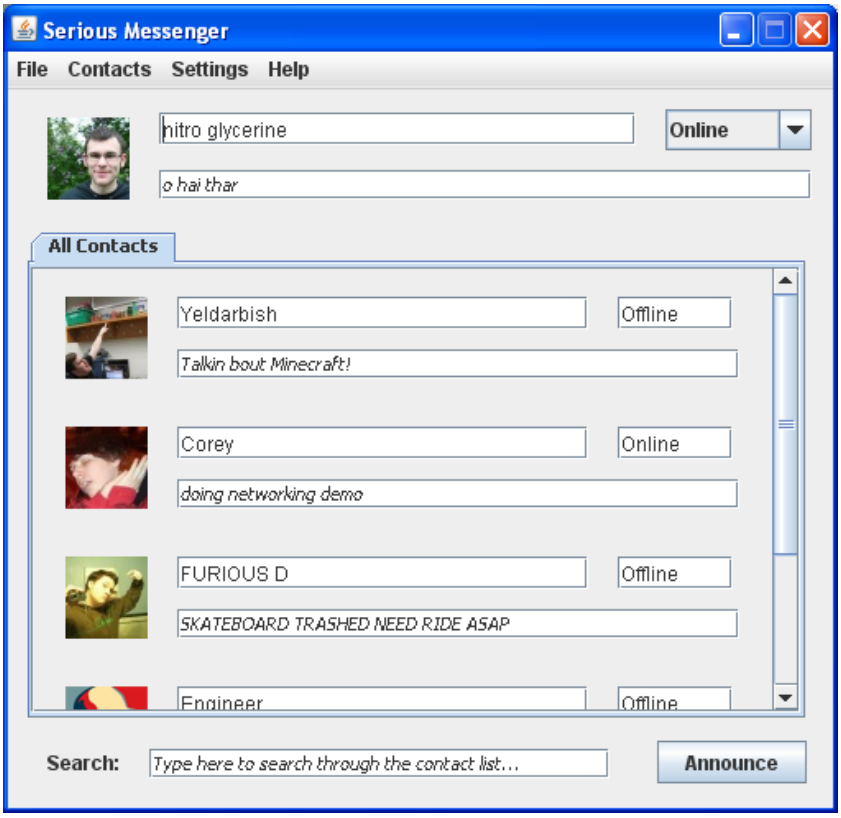

Figure 21 - The contact list after adding contact Corey.

Change Password

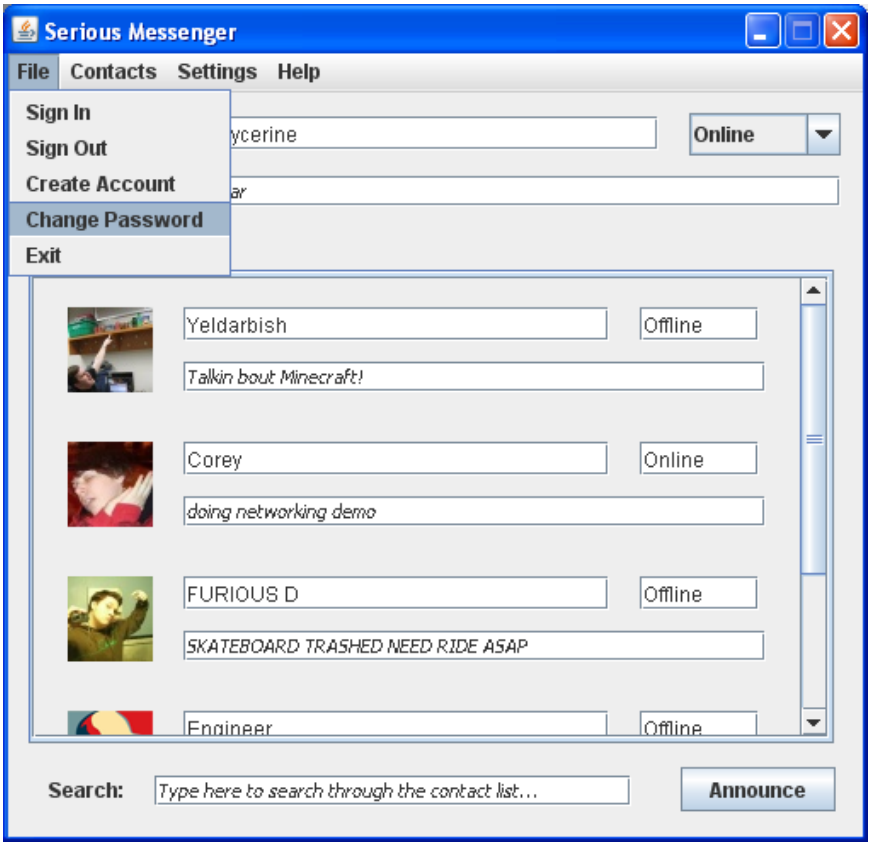

Figure 22 - Select 'Change Password' from the File menu.

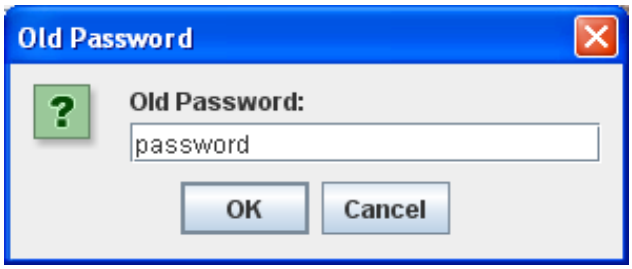

Figure 23 - Prompt for old password (security reasons).

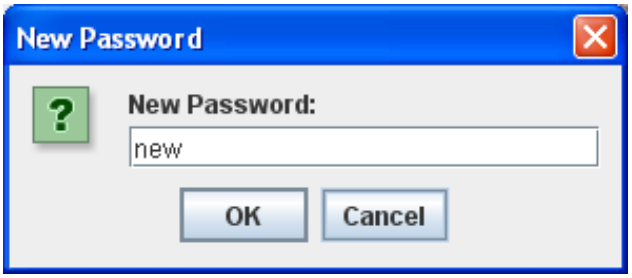

Figure 24 - Prompt for new password.

Figure 25 - Prompt to confirm new password change.

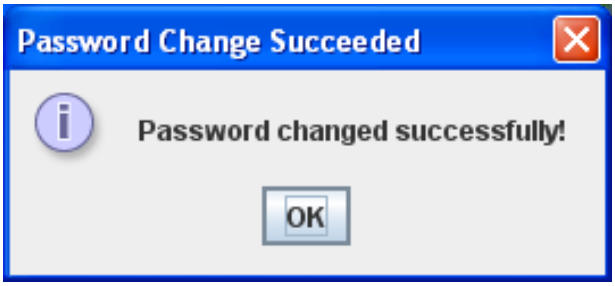

Figure 26 - Dialogue alerting you that the password change was successful.

Now, if you were to log in, you will require the *new* password, the old one is no longer stored anywhere.

Create Account

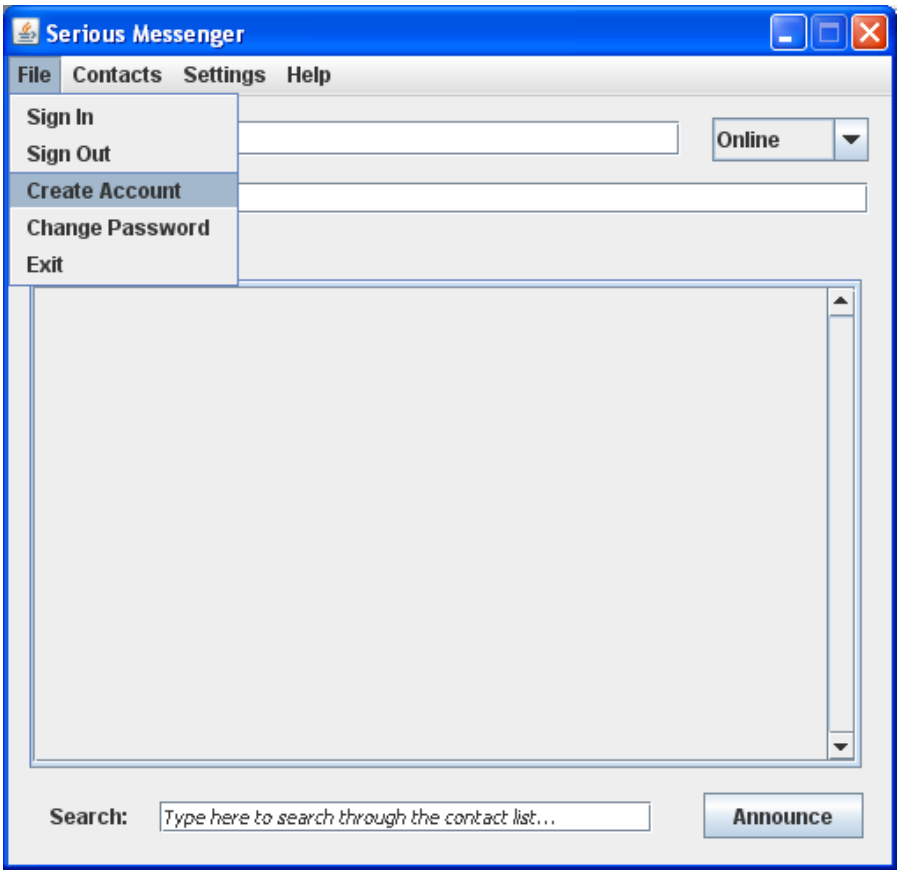

Figure 27 – Select 'Create Account' from the File menu.

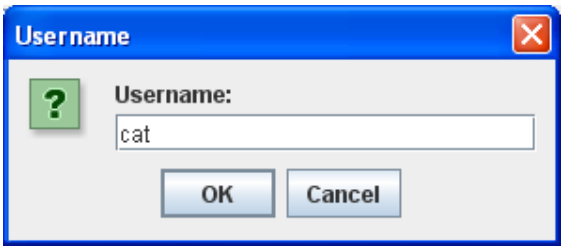

Figure 28 - Prompt user to enter their desired username.

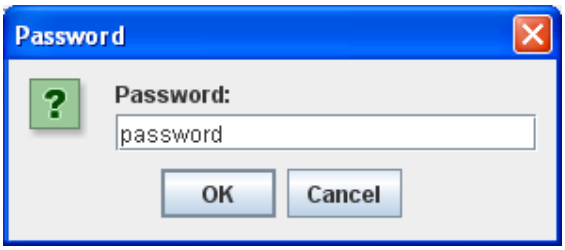

Figure 29 - Prompt user for desired password.

Provided your desired username was not already taken, and since there is are invalid passwords, you will arrive at this:

Figure 40 - Dialogue notifying you that the account creation was successful.

Once you're done with that, try signing in with your new account! You should end up at the main screen, looking something like this:

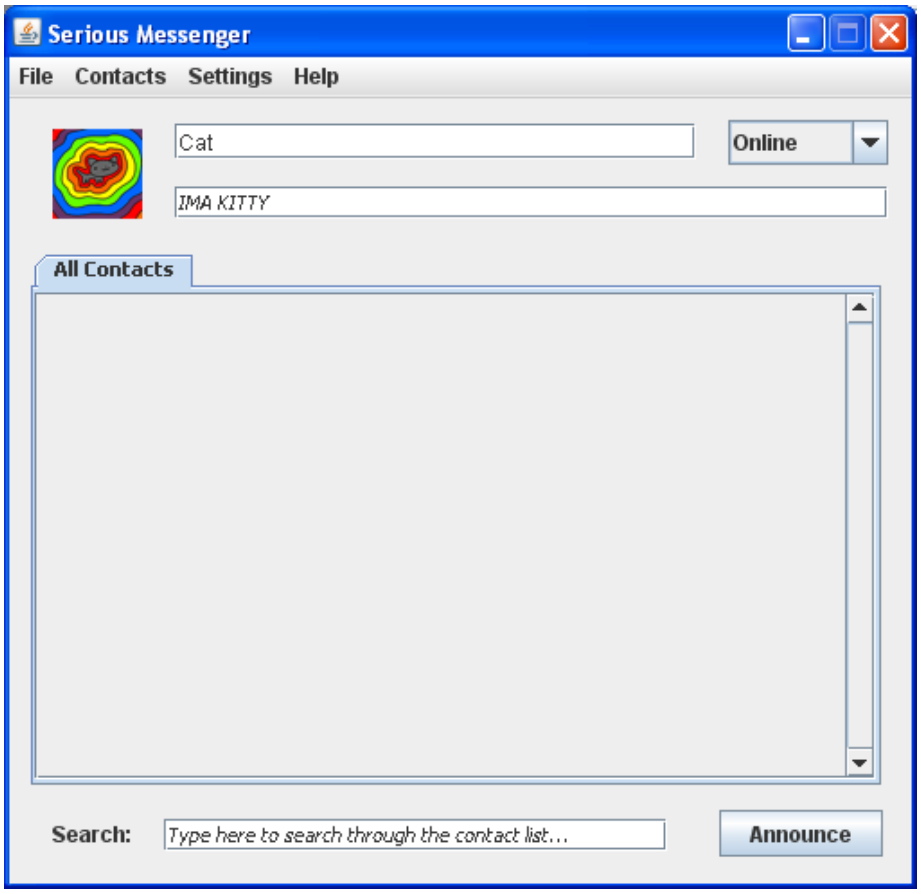

Figure 31 - The main window. Logged in using the new account.

Sample Server Screenshots

Here are some sample screenshots of what the tabs in the server look like with varying data in it.

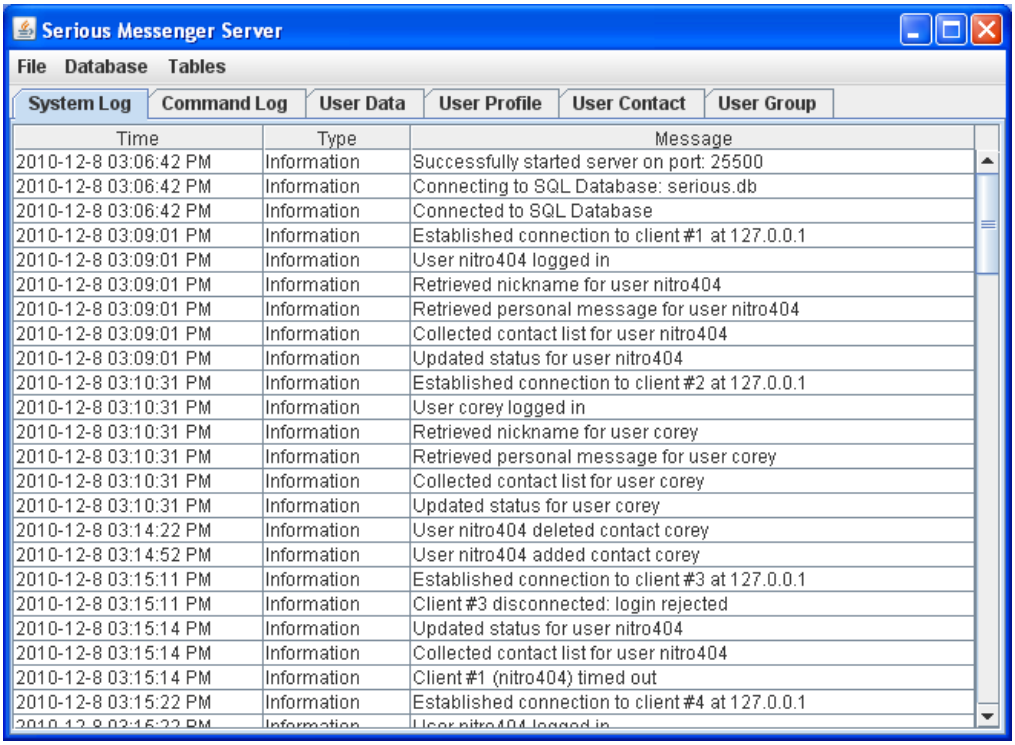

Figure 32 - The System Log.

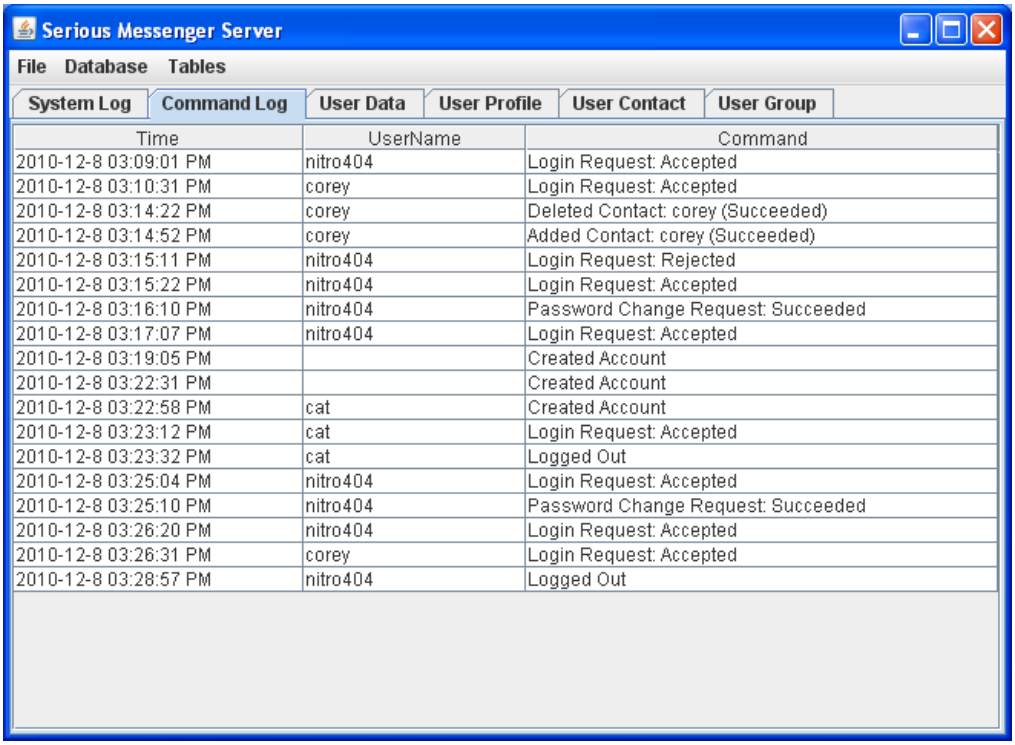

Figure 33 - The Command Log.

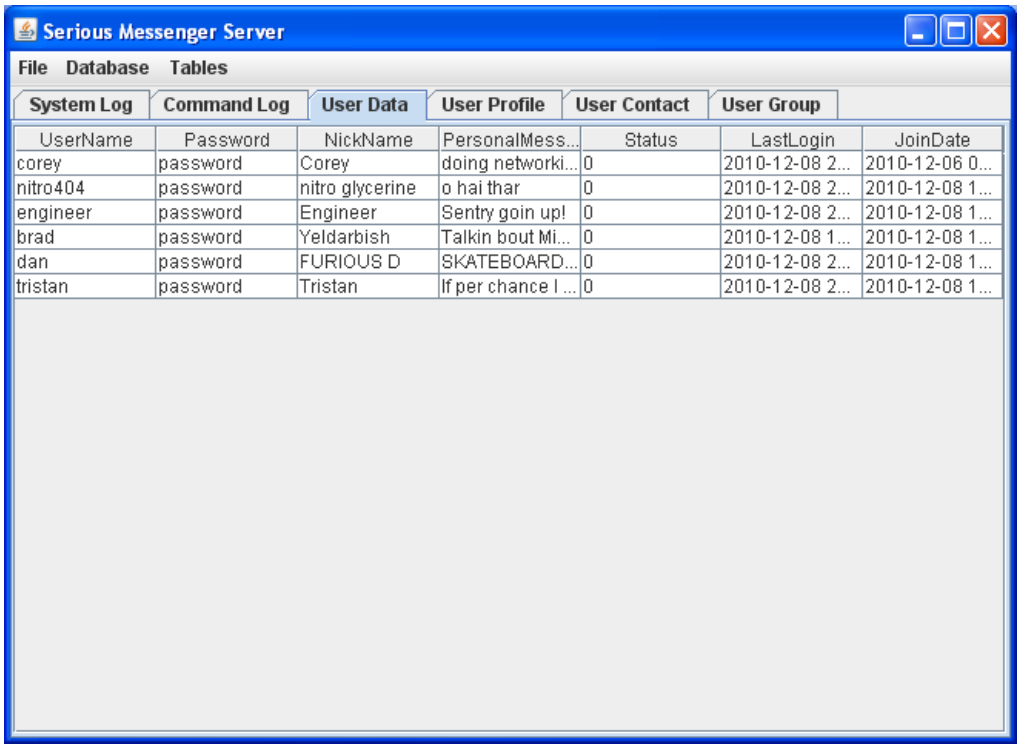

Figure 34 - The User Data table view.

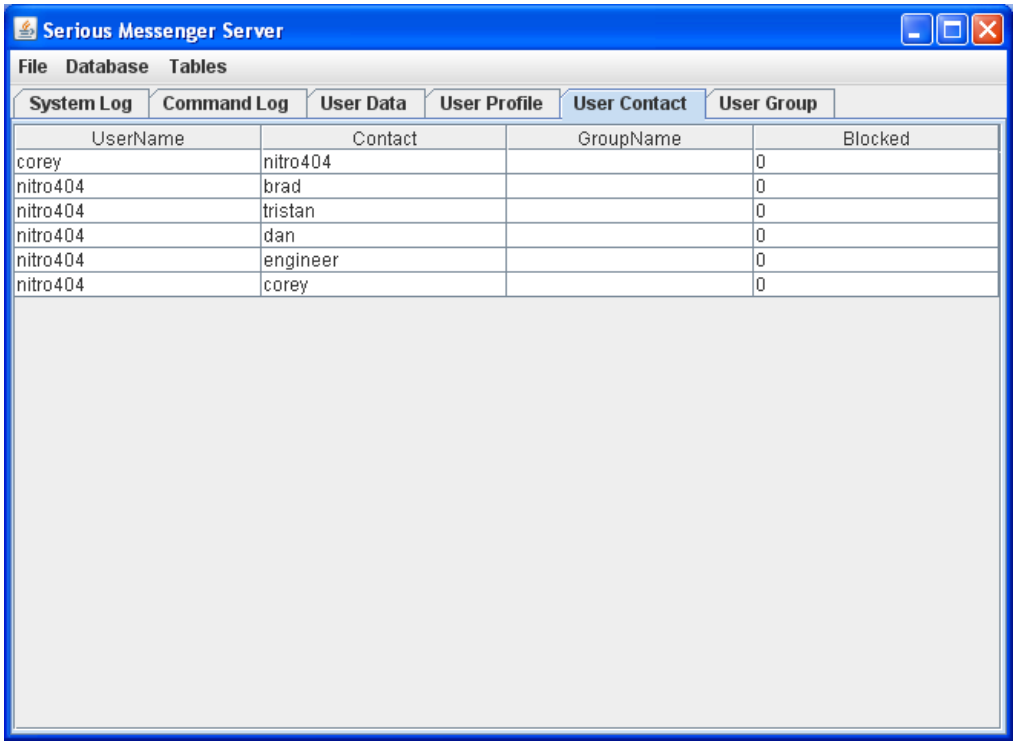

Figure 35 - The User Contact table view

TEAM CONTACT INFO

Corey Faibish
Student #: 100764177
Email: corey.faibish@gmail.com OR cfaibish@connect.carleton.ca

Kevin Scroggins
Student #: 100679071
Email: kscrogg2@connect.carleton.ca OR nitro404@hotmail.com