Spring 5-1-2011

Java Implementation of an Office Communicator

Swathy Kodati
Dakota State University

JAVA IMPLEMENTATION OF AN OFFICE COMMUNICATOR

A graduate project submitted to Dakota State University in partial fulfillment of the requirements for the degree of

Master of Science

in

Information Systems

May, 2011

By
Swathy Kodati

Project Committee:

Stephen Krebsbach
Ronghua Shan
Surendra Sarnikar

PROJECT APPROVAL FORM

We certify that we have read this project and that, in our opinion, it is satisfactory in scope and quality as a project for the degree of Master of Science in Information Systems.

Student Name: Swathy Kodati

Master's Project Title: **Java Implementation of an Office Communicator**

Faculty supervisor: Stephen Krebsbach Date: 4/27/11

Committee member: Ronghua Shan Date: 4/27/11

Committee member: Surendra Sarnikar Date: 4/27/11

ACKNOWLEDGMENT

I take this opportunity to express my sincere gratitude and thanks to everyone assisted me in completing this project on time. First of all I would like extend my special thanks to my project supervisor **Dr. Stephen Krebsbach** for his kind attention and support during the entire phase of project development. I am blessed to have a family that supported and helped me in achieving my goals as a student. I also extend my sincere thanks to fellow students, friends and several professionals that I have met during this course of project work.

I have learned so many things about Java Programming in a real world scenario and was able to solve several programming and technical issues that came along the way to the completion of the project. This wouldn't have been possible without the help of my family, husband and all others whose names are not mentioned here.

ABSTRACT

This project aims to design, develop and implement an Office Communicator in Java. Multiple clients will be able to interact with each other in a shared environment over a multithreaded peer to peer communication system. Broadcast message option will be provided to the clients along with an option to edit and view the information shared simultaneously. The information can be text, audio, image.

Client/Server architecture over TCP/IP network is used as the underlying architecture for the entire project. Clients connected to the server will be able to perform some or all of the below features depending on their status:

- Broadcast of multimedia messages.
- Creation/administration of Public/Private channel.
- Group/Private instant messages.
- Offline messages.
- Drawing board.
- Record and relay of voice messages.

DECLARATION

I hereby certify that this project constitutes my own product, that where the language of others is set forth, quotation marks so indicate, and that appropriate credit is given where I have used the language, ideas, expressions or writings of another.

I declare that the project describes original work that has not previously been presented for the award of any other degree of any institution.

Signed,

Swathy Kodati

TABLE OF CONTENTS

LIST OF FIGURES

INTRODUCTION

Audiovisual coding systems and network protocols such as TCP, UDP and RTP form the basis for the Multimedia communication over IP. Communication can be one way, from a sender to one or more receivers, or two-way or an interaction between two or more parties i.e. Broadcast, Multicast or Unicast. Streaming, video or audio conferencing are some examples of multimedia communication.

Ever since computers came in to use attempts were made to take the existing communication media to next level. In the last two decades we have seen significant advances in this direction as a result of web revolution, availability of greater speed network access to users and the adoption of popular protocols such HTTP and HTML as standards. These impacted heavily in making the streaming media practical and affordable to ordinary users. Huge investments are ongoing in this industry to compete with the traditional communication industry of telecom and television with the corresponding web equivalents such as Voice over IP (VoIP) and IP Television (IPTV).

Coming to the communication in the Business industry, we have gone past the times where in the entire official work has been transformed from paper to electronic form. Emails have become the official communication media. Organizations equip the employees with all communication resources that help them to do their day to day office work. Office Communicator has become more of a necessity now a day's rather than just a tool.

Background of Project:

The drastic increase in the need for interactive communication has made people and businesses increasingly (another word) dependent on well networked computer systems to support distributed applications. The level of interaction for these distributed applications is not just limited to local area networks (LAN) but must also cater/be viable to access external networks which is a link to anything on/in the World Wide Web.

To make the most of the services offered and thereby improve the efficiency on these applications, information regarding the services, resources, users and other vital aspects needs to be collected, analyzed and organized in a consistent manner.

Since these distributed applications offer unlimited access to sharing data, security is of at most importance especially when dealing with sensitive information. In order to prevent unauthorized access or modification of information the application has to be designed to have various levels of authentication.

In the existing legacy system, the User or the community member uses these following methods to transfer information from one community to the other. Interoffice mail service where the mails are personally delivered by a messenger or the Intercom service provided for conference calls and at times the Electronic mail (E-mail).

Below are the disadvantages for the existing legacy system:

- Tedious message broadcasting system.
- Communication is not instant.
- Message transfer is done through insecure communication media.
- Communication delays.
- Maintaining the identities of all the members is not efficient.

To overcome the above demerits of the existing system a new application which enables the ease of use, cost effective administration and above all to provide a reliable and secure way of communication is very essential. Therefore Office Communicator over IP networks is developed to effectively utilize resources, process data and diagnose faults at a low-cost while still maintaining the accuracy and performance levels.

OBJECTIVES

The goal is to design an integrated system to allow users to participate in an instant message communications with other community members either in groups or as peers. A system shall support public and private channels of communication and uses its own internal permission system to more securely determine who can enter, create or delete channels/rooms. An attempt is being made to develop a communication system with graphical user interface that support's Office Communicator, and data sharing in a real time environment i.e., the data broadcasted can be edited online and viewed simultaneously. The data transmitted can be text, voice, picture or a drawing image, etc.

Many companies around the globe offer communication facilities via chatting, etc., by adding a wide range of services to those systems. Some of these chatting applications also support different media of communications with well designed user interface.

But unfortunately, their chat software is proprietary and tend to be expensive to maintain and will require a separate community data model. Hence the challenge is to design a communication system for an existing network as per the organizations requirements with an ease of maintenance by the local programmers.

Purpose of Project:

The main purpose of the project is to deliver messages between server and multiple clients on user to user basis or within private/public channel. It aims to satisfy the task of transmitting a single message to all users through a single server by a single click. The entire architecture is maintained by the server, to which one of the clients shall send a message and the server routes or passes the message to the other required clients in the network.

An attempt is being made to develop a communication system with graphical user interface that support's Office Communicator over TCP/IP network. It shall allow data sharing in a real time environment i.e., the data communicated can be edited online and viewed simultaneously. It shall provide fast, secure, reliable and cost effective communication

medium between multiple clients and server. The data transmitted can be text, voice, picture or a drawing image, etc.

Many organizations all over the world offer different modes of communication facilities like chat, mails… etc., with well designed user interface, which is proprietary and expensive to maintain. Now the challenge is to design a communication system for an existing network as per the organizations requirements with an ease of maintenance by the local programmers.

LITERATURE REVIEW

Current businesses are increasingly relying on networked computer systems to support their business model. In a distributed network model different applications and users will need to interact with computers on one or more local area network (LAN) typically addressed as Intranet in the modern day corporate. A messaging system is inevitable to have in large corporate to effectively communicate with various users within the organization.

Why Office Communicator?

When there are several messaging services available on the market, the need to have an office communicator is very minimal and of course that was arguably the first thought that would be in everyone's mind.

Office Communicator is specifically designed to small and large business that actively uses messaging systems to complete their daily activities. There are numerous IMs (Internet Messaging Systems) applications available but Office Communicator comes as a package by filling the gaps that are in current day IMs. The following key features make Office Communicator as special messaging system.

1. No database or DBMS system used (Improves Performance).
2. Highly customizable and scalable.
3. Developed based on Open Source platform (Java) which is very popular in current day networking and internet applications.
4. Platform Independent implementation as Java is platform independent.
5. Drawing board feature is an innovative feature that can be used by exchanging messages and drawings.
6. Data sharing in a real time environment. Data broadcasted can be viewed and edited simultaneously
7. Multiple media transmission support; like voice, text, images, etc.
8. P2P (Peer-to-Peer) messaging service that improves performance and security.

9. Multiple media transmission support for voice, text, images, etc
10. Rich and user friendly interface.

SYSTEM DESIGN

Office communicator design is based on Object Modeling and Object Oriented techniques using Java as the primary language. A dedicated chat server will be available and TCP listener is used to establish a client connection. A high level architectural diagram is presented below.

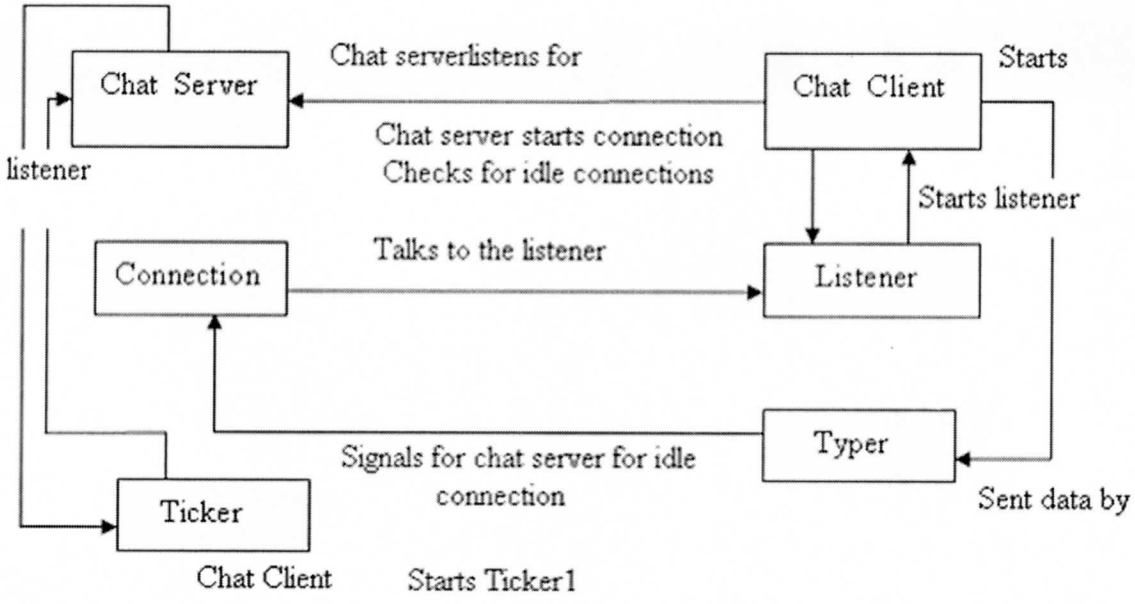


Figure 1. Office Communicator Diagram

The project can be classified into the following modules:

- **Server Module**
- **Client Module**

Server Module

In the client server architecture, the server acts as a main module, the job of the server is to wait for the clients request and respond accordingly with a specific service. The administrator initiates the server and monitors the connections to the clients. The server module validates the clients with their username & password against the login database and authenticates them (see figure 2). When the client is not authenticated the connection is refused and an error message is sent back to the client. The server module acts as an interface for the communication between different clients as well as send broadcast messages to multiple clients like, Disconnecting Users, Server Shutdown, etc. Many clients can connect to the server and it is the single point of contact to all the clients. The administrator of the server module is in control for all the communications happening between the clients. The sub modules which are part of the server are

- User Management Module
- Connection Management
- Transmission Management
- Log Management

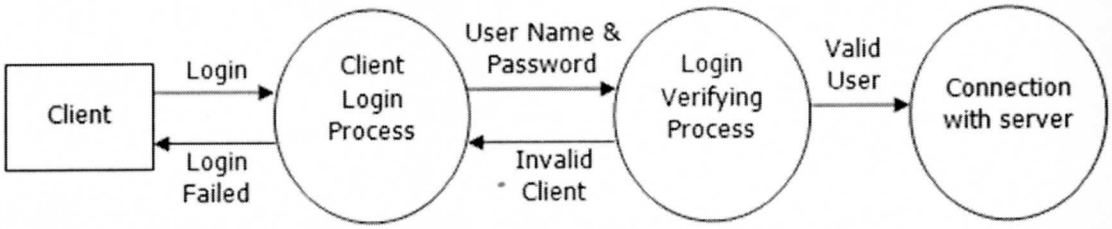


Figure 2. Authentication data flow diagram.

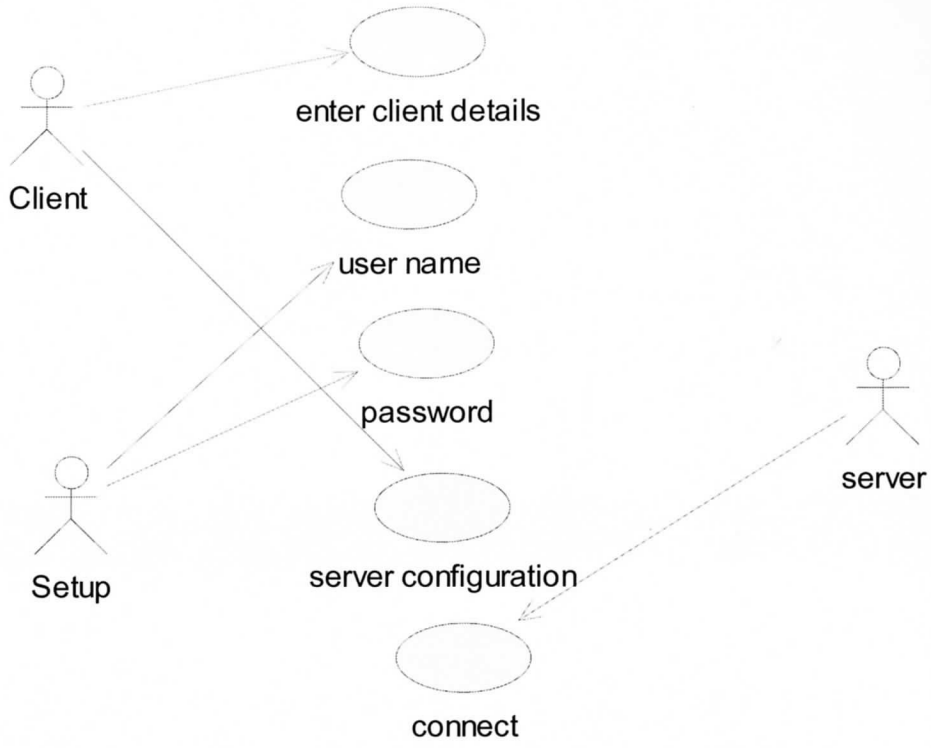


Figure 3. Use case diagram.

Client Module

The job of client module is to transmit message to other clients by establishing connection through the server module. The user enters the username and password thorough the client module to authenticate against the server. The module enables the user to communicate through different channels or create new channels (chat rooms) for group of users to share common messages through that channel. Once the client gets connected he can start communicating with other users from server module, the user can transmit the data in many ways like text, voice, picture etc. or he can initiate an encrypted private channel with a specific client.

This module provides a common place to edit the data for all the clients that form the broadcast domain.

Sub modules involved in the client are:

- Client Interface Module
- Client Chat Room Management
- Message Transmission Module
- Log Session Management

Work Flow Description

First up, the Chat Server is started. Administrator can choose to login or shutdown the server. Once the administrator signs in, he can have additional options to disconnect just the users or the entire group of users from the system, including the administrator user.

Once the administrator signs in, another window pops up that shows who all users are signed in. On the main window the administrator has an option of "User Management" using which he can add new users or delete existing users.

For the client user to login, once the client chat program is started, he/she can connect to the server by entering the login credentials in the prompt that comes after the user clicks "connect" button on the file tab of the client GUI. Once the user is authenticated against the credentials that are already stored in the server various features are available for the user.

A field of "currently sending to" on the right hand side shows the list of available online users present and gives an option for the client to decide who all shall receive the messages from the client. Similar field is present on the Server chat window as well.

Client user has an option to enter in to existing chat rooms or create new chat rooms, both public and private that shall require a password authentication. Once the chat room is created the user has an option to invite different users among the ones that are available to join the chat room.

In the messaging feature, the available online users present are shown and the user can decide whom to send a specific messages. If the destination user is offline, user name can be typed and an offline message can be sent. Same messaging feature window can be used by the end user to read the offline messages.

Drawing canvas is one unique feature that is available to all users. The drawing canvas can be shared between a group of users or by all users who are online. Users have an option to draw things by hand or use specific shapes or even just type. Both private drawings and public drawings can happen in the same canvas that is shared by all. There is also an option for the users participating in the conference to clear the canvas and start from scratch.

From the Audio tab available, the users can create new voice messages and send them to various users. Also they can receive and listen to new messages as well.

Hardware Requirements

Hardware	Server	Client
Processor	Intel Pentium IV	Intel P-III and above
Processor Speed	3.0 GHz	2.6 GHz
RAM	512 MB and above	128 MB and above
Hard Disk	40 GB and above	10 GB and above
Floppy Drive	1.44 MB	1.44 MB

Software Requirements

Language	:	Java 1.5 and above
Operating System for Server	:	Windows XP/NT/2000/ 03/08 servers
Operating System for client	:	Any windows OS

Basic Work Flow:

*chatServer.java

```
231
232     protected chatChatRoom findChatRoom(String roomName)
233     {
234     // This will return the chat room object with the corresponding
235     // name
236
237     chatChatRoom tempRoom = null;
238     chatChatRoom returnRoom = null;
239
240     // Is it the main chat room?
241
242     if (mainRoom.name.equals(roomName))
243     {
244        returnRoom = mainRoom;
245     }
246     else
247         {
248
249         for (int count = 0; count < chatRooms.size(); count ++)
250             {
251             tempRoom = (chatChatRoom)
252                 chatRooms.elementAt(count);
253             System.out.println("temp :"+tempRoom);
254
255             if (tempRoom.name.equals(roomName))
256                 {
257                 returnRoom = tempRoom;
258                 break;
259                 }
260             }
261         }
262
263     return (returnRoom);
```

The initial flow is well known login module. Since we are all much familiar with the login module, let us skip that part and concentrate only on the chat mode. Firstly, we assume that we have all the code written and executed to make the design function. After logging in, we will be joining in any of the chat rooms available or create new chat room. There will be a main room and some temporary rooms available. These temporary rooms will be assigned a temporary session. In that particular session, the room acts as a main room. If a new room is created then the new room will be forced to be in a new session and the earlier session will be

the main one. These temporary sessions expire or timeout after the room is closed. The above piece of code returns the room.

*chatServer.java

```
266     public boolean isUserAllowed(chatChatRoom room,
267                     chatClientSocket client, String password)
268     {
269     // Return true if a user is allowed to enter a chat room.  False
270     // otherwise.
271     System.out.println("room.priv"+room.priv);
272     // Private room?
273     if (room.priv)
274         {
275         boolean invited = false;
276
277         // Make sure that this user was either invited, or supplied
278         // the correct password.  Check the password first
279
280         if (!room.password.equals(password))
281             {
282             // No correct password supplied.  Check the list
283             // of invitees
284
285             for (int count2 = 0;
286                  count2 < room.invitedUsers.size(); count2 ++)
287                 {
288                 Integer Id = (Integer) room.invitedUsers
289                     .elementAt(count2);
290                 System.out.println("ID"+Id);
291                 System.out.println("Id.intValue()"+Id.intValue());
292                 System.out.println("client.user.id"+client.user.id);
293                 if (Id.intValue() == client.user.id)
294                     {
295                     invited = true;
296                     break;
297                     }
298                 }
299
300             if (!invited)
```

*chatServer.java

```
298                 }
299
300             if (!invited)
301                 {
302                 try {
303                     client.sendServerMessage(
304                      "Not invited to/incorrect password " +
305                      "for the private room " +
306                      room.name);
307                 }
308                 catch (IOException e) {
309                     disconnect(client, false);
310                     return (false);
311                 }
312                 return (false);
313                 }
314             }
315         }
316
317     // Make sure the user has not been banned from the room
318     for (int count1 = 0; count1 < room.bannedUserNames.size(); count1 ++)
319         {
320         if (room.bannedUserNames.elementAt(count1)
321             .equals(client.user.name))
322             {
323             // This user has been banned from this room.  Send
324             // them a message and quit
325             try {
326                 client.sendBanUser(client.user.id, room.name);
327             }
328             catch (IOException e) {
329                 disconnect(client, false);
330                 return (false);
331             }
332             return (false);
333             }
334         }
335
```

The above two screen shots comprise of the piece of code whether the user is eligible to enter into a particular room or not after selecting a room. There will be different scenarios, where a user is not allowed to enter into a room. Few of them are like an administrator's room where regular users are not allowed. Some banned users will not be allowed to enter into any room or a particular room based on the privilege until the ban is lifted. Some old users or ex-employees will no longer have a privilege. So their ID's will be removed. These ID's should also be removed from the data base server. If not removed, the code will allow them to utilize the chat privilege.

*chatServer.java

```
339
340     public synchronized void disconnect(chatClientSocket who,
341                     boolean notify)
342     {
343     int count;
344     chatChatRoom chatRoom;
345
346     if (notify)
347         {
348         try {
349             // Try to let the user know they're being disconnected
350             who.sendDisconnect(who.user.id, "You are being " +
351                     "disconnected.  Goodbye.");
352         }
353         catch (IOException e) {}
354         }
355
356     // Shut down the client socket
357     who.shutdown();
358
359     // Remove the user from their chat room.
360     try {
361         who.leaveChatRoom();
362     }
363     catch (IOException e) {}
364
365     // Remove the user from our list of connections
366     synchronized (connections)
367         {
368         connections.removeElement(who);
369         connections.trimToSize();
370         currentConnections = connections.size();
371         }
372
373     serverOutput("User " + who.user.name + " disconnected at "
374             + dateFormatter.format(new Date()) + "\n");
375
```

It is also important that the user is to be notified when disconnected from the chat server. There could be many reasons few of which are a connection lost, banned scenarios, sessions timed outs etc. In such cases, the user's session is terminated and should be removed from the list in that particular room and also should be notified.

CONCLUSIONS

I am satisfied with the way the project has come out. It was a challenge initially when I decided to pursue an office communicator project using Java. After much analysis and detailed study of several messenger products available, I decided to continue with the office communicator. After carefully planning on the deliverables and defining scope it was wise decision to continue this project without a database management system as with the application of RDBMS it will overwhelming for the scope of this project.

Office Communicator project helped me identify the gaps in the current messenger applications and improved my pro-active thought process that will help the product sustainability in the current expanding technology trends. End of the day I was happy to get this product delivered as planned.

My technical knowledge in Object Oriented programming has improved to a great extent and I learnt a lot about Java and Socket programming that helps in network communication. I was also able to understand and use a vast variety of Java libraries that I can use to achieve anything from simple to complex in the application programming world. With the Java implementation I was able to understand the importance of platform independence. Since Java is open source, it helped me to find out solution for numerous issues that I came across during the course of the project.

This project is scalable and there is a lot of scope for improvement with the introduction of database RDBMS in to this, you will be able to keep history of the messages, track and audit the messages for Super Admin of the system by introducing several roles for the Office Communicator. We can also expand this project to allow real time video messaging or video conferencing system.

REFERENCES

Schildt, H. *JAVA: The Complete Reference J2SE 5th Edition*. United States: Mc Graw Hill Osborne.

http://www.buyya.com/java/Chapter13.pdf

http://www.ibiblio.org/java/slides/sd2003west/sockets/Java_Socket_Programming.html

http://www.oracle.com/technetwork/java/socket-140484.html

http://www.tutorialspoint.com/java/java_networking.htm

http://www.cyberconf.org/~cynbe/java/classes/applets.html

http://download.oracle.com/javase/tutorial/networking/sockets/index.html

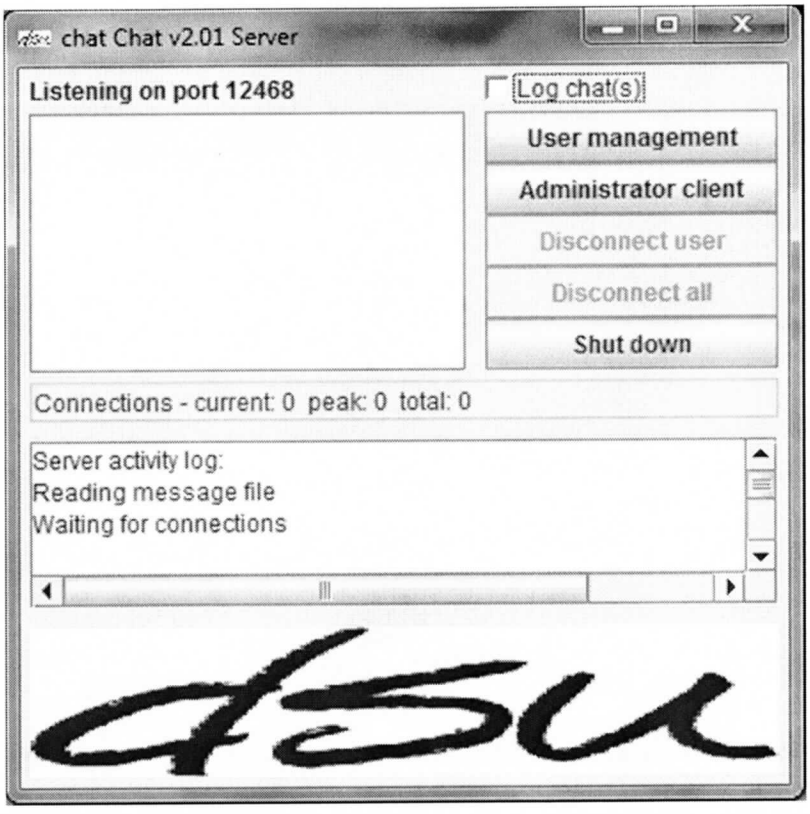
http://www.tutorialspoint.com/java/java_variable_types.htm

http://download.oracle.com/javase/tutorial/deployment/applet/index.html

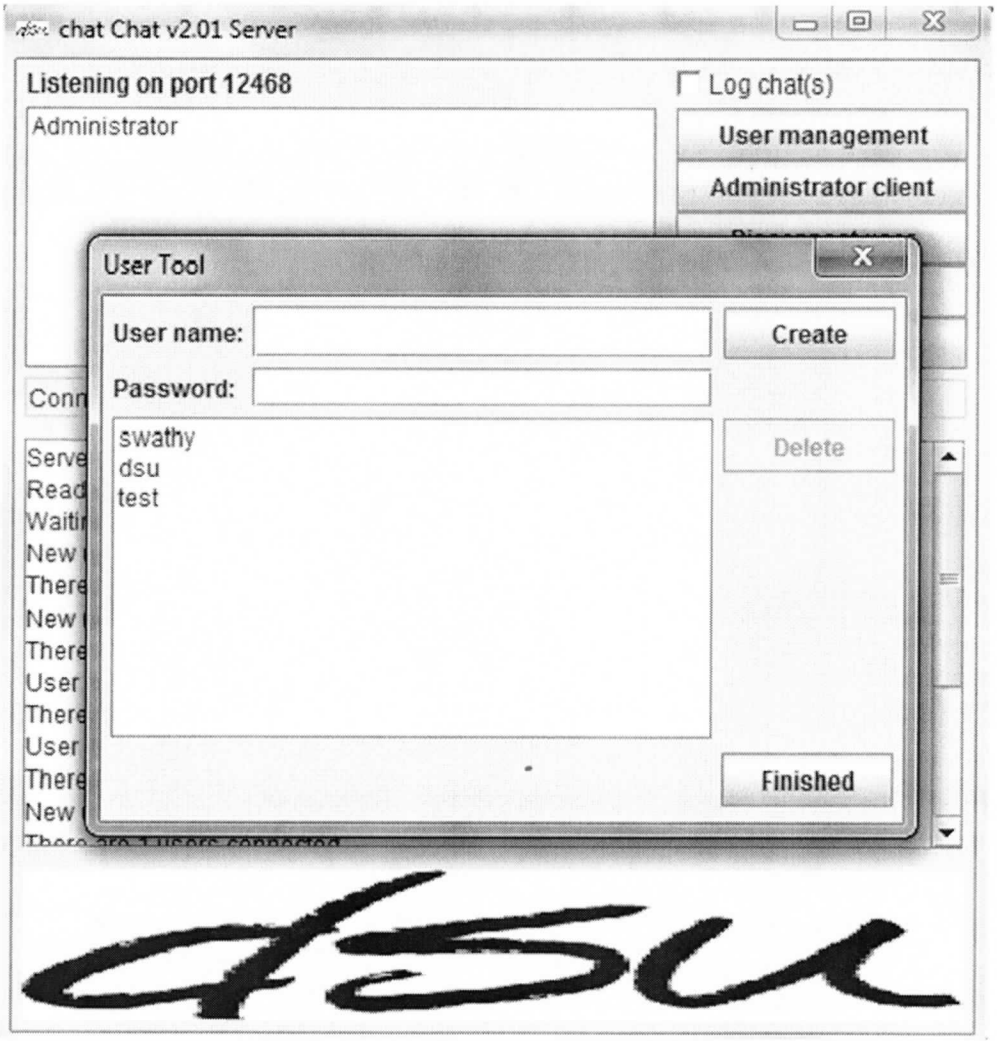
APPENDICES

APPENDIX A: Screen Shots

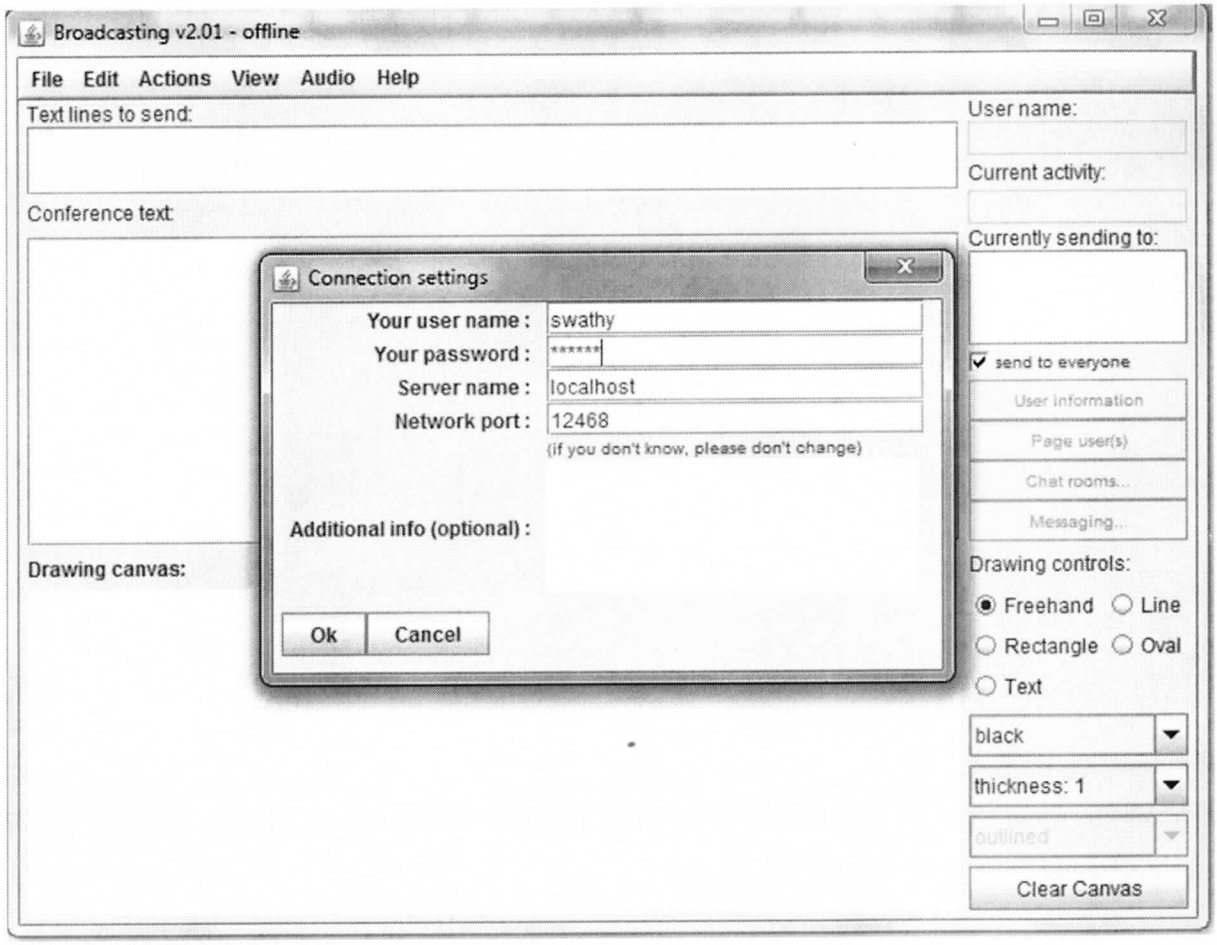
Server Main Screen



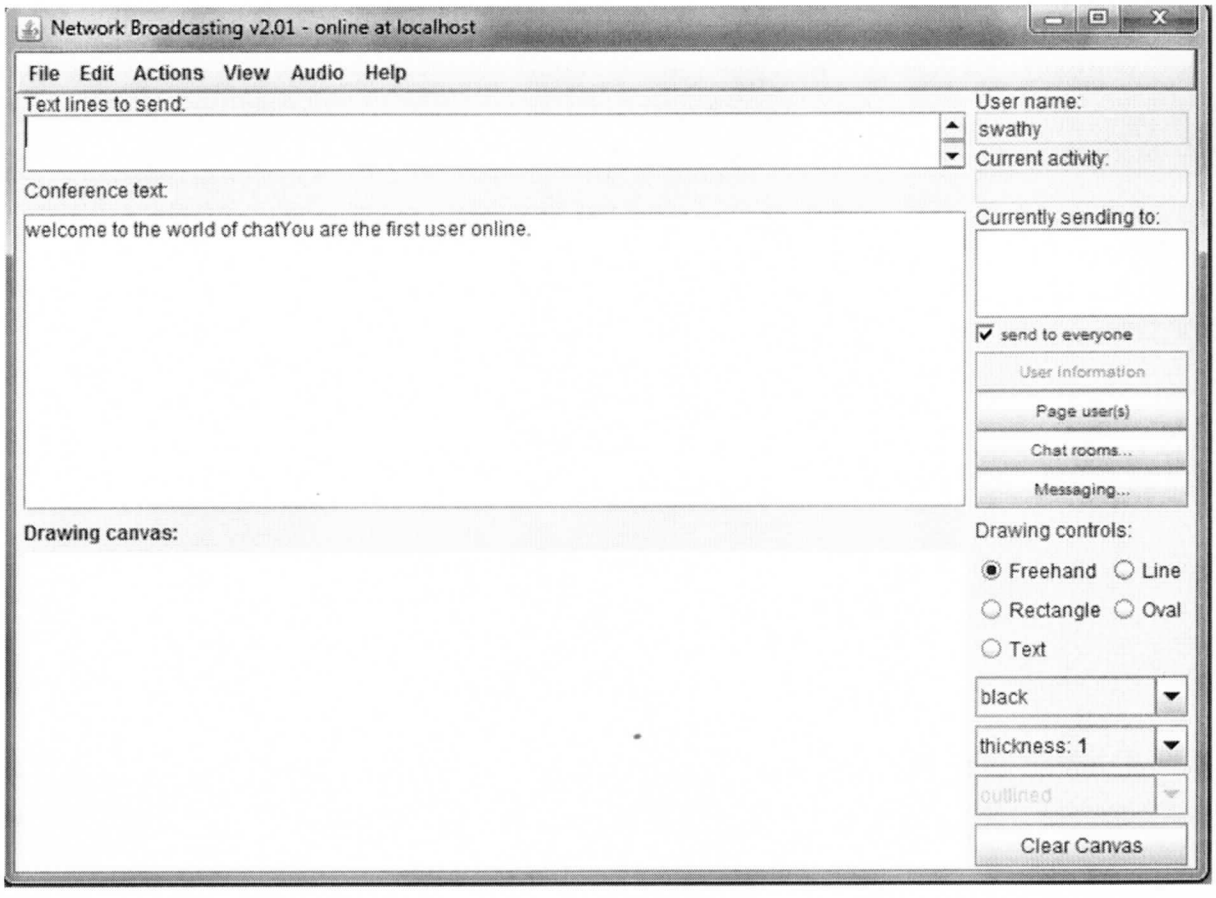
User Management Screen



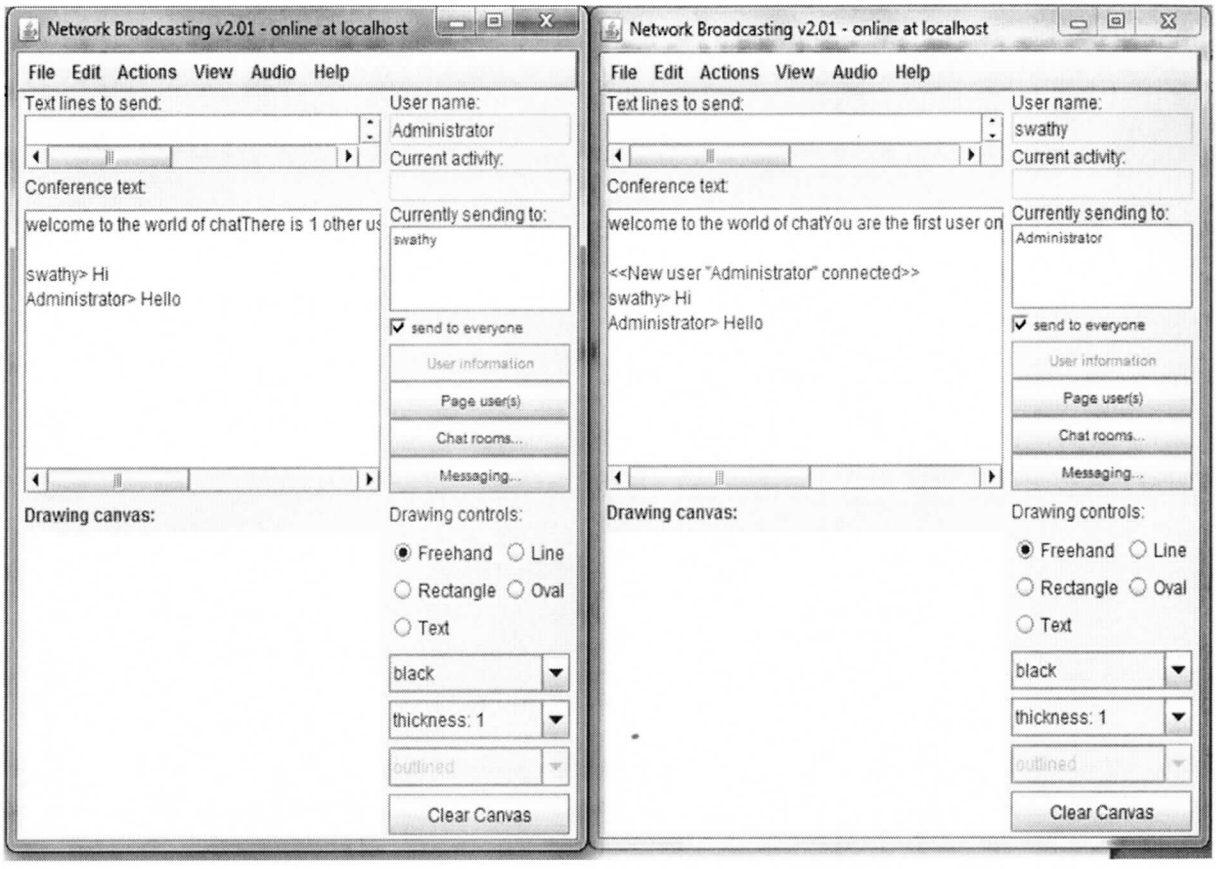
Client Connection Window Screen



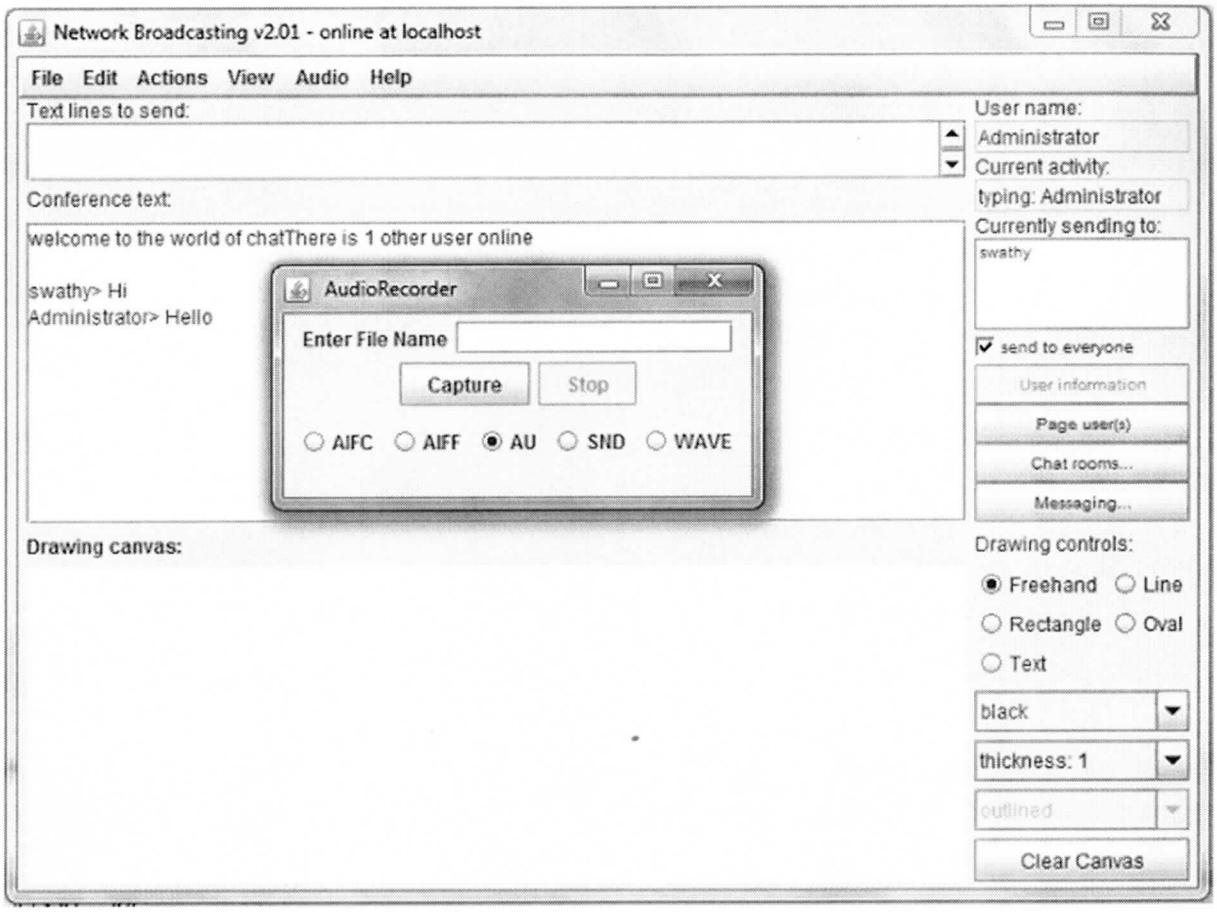
Client Login Screen



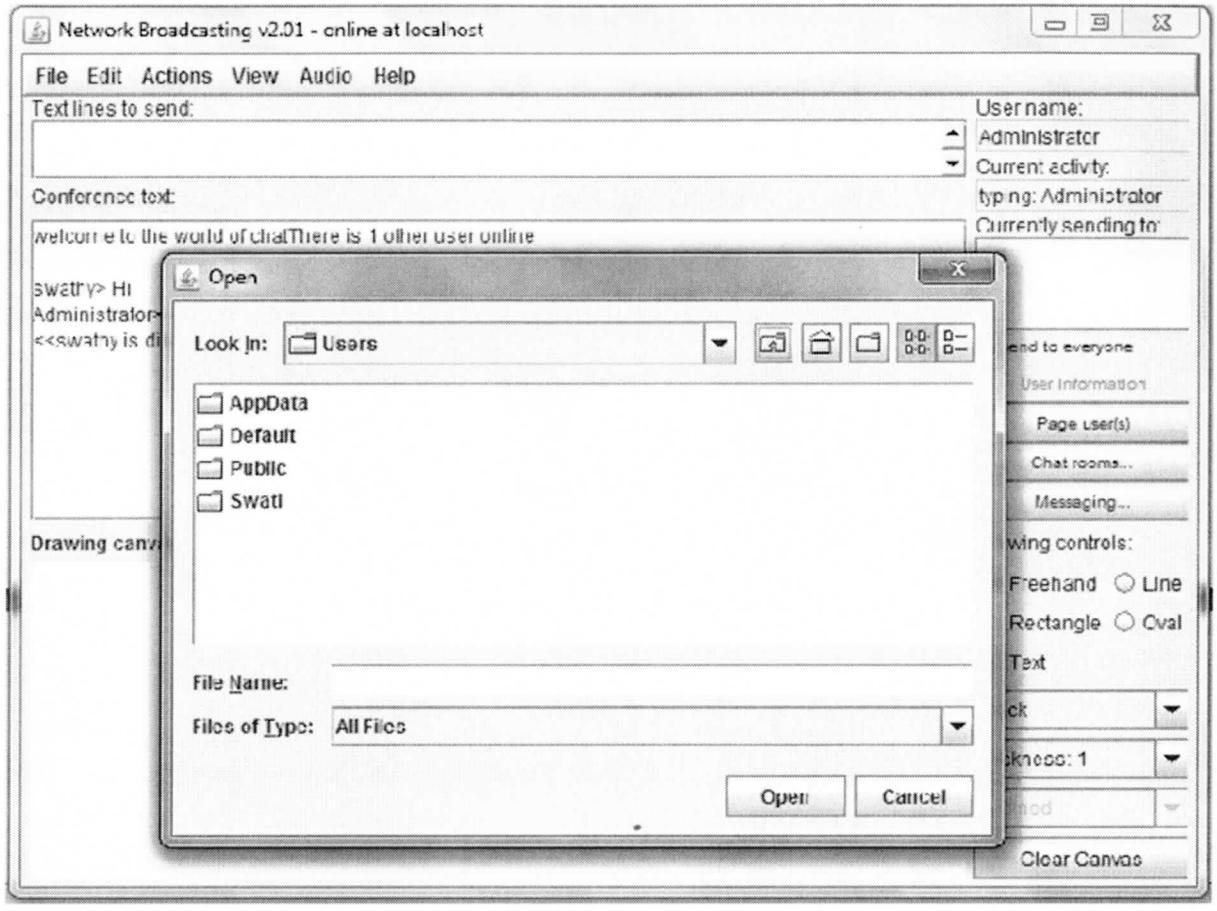
Administrator-Client Conversation



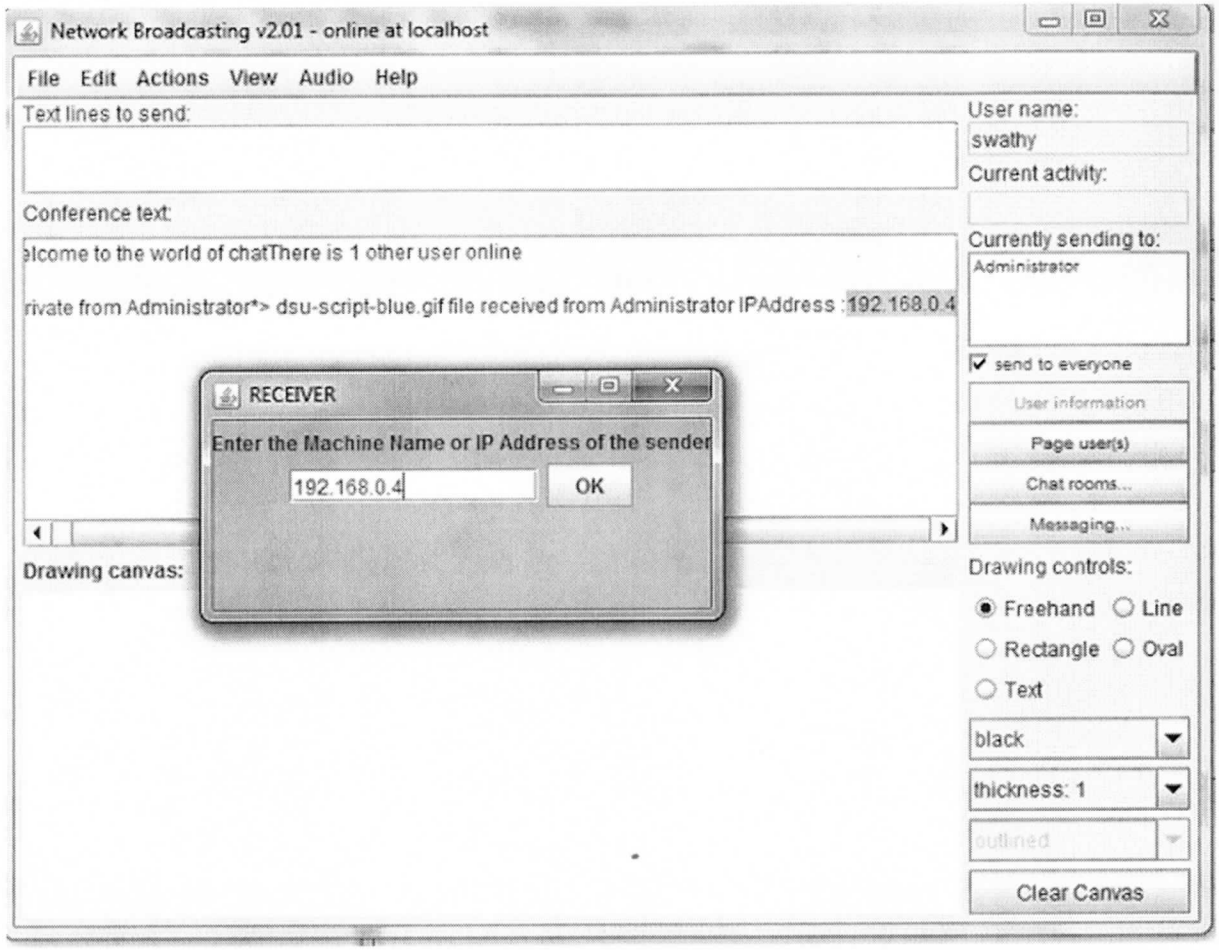
Audio Recorder Screen



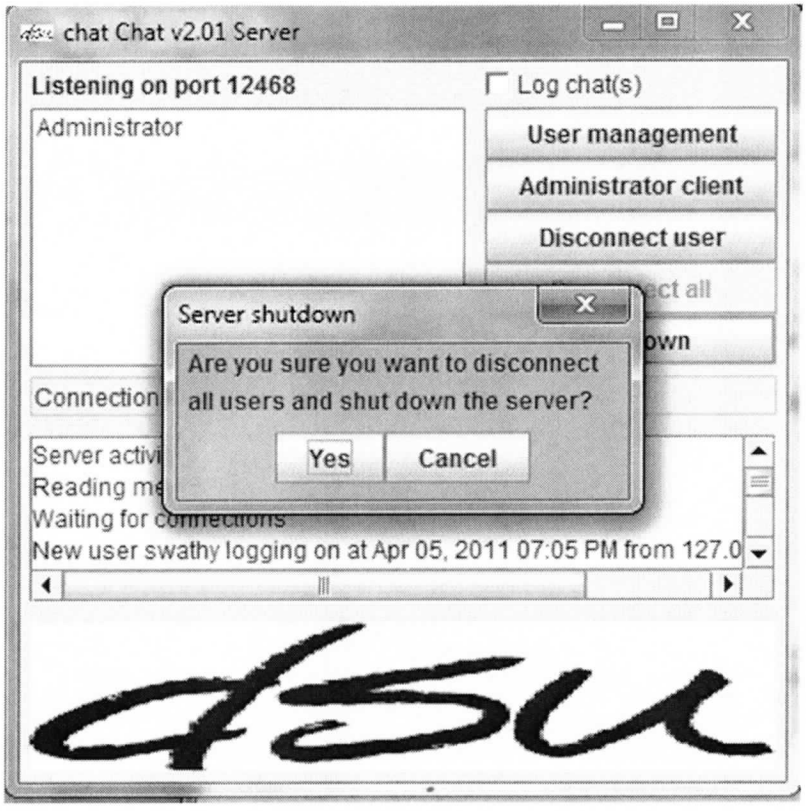
Selecting File to Send



File Received by Client



Server Shutdown Window



APPENDIX B: Program Code

ChatServer.java

```
import java.net.*;
import java.util.*;
import java.text.*;
import java.io.*;
class chatServerShutdown extends Thread
{
  // This gets called when the VM is shutting down.  If the server has
  // already terminated properly (for example, from the administrator
  // pushing the 'shut down' button), then there's nothing to do.  But,
  // if the administrator sends a KILL signal, or presses CTRL-C or
  // something like that, then this thread will shut down the server
  // properly
  private chatServer server;
  public chatServerShutdown(chatServer s)
  {
      server = s;
  }
  public void run()
  {
      if (server.stop)
        // The server has already shut down by itself (i.e. not
        // from an external signal)
        return;
      server.externalShutdown = true;
      server.shutdown();
  }
}
public class chatServer
```

```
        extends Thread
    {
            Calendar cal = Calendar.getInstance();
            protected String name = ((cal.get(Calendar.DATE)) +"_"+
(cal.get(Calendar.MONTH))
+"_"+(cal.get(Calendar.YEAR))+"_"+(cal.get(Calendar.HOUR))+"_"+(cal.get(Calendar.MIN
UTE))+"_"+(cal.get(Calendar.SECOND))).toString();
        protected static String userPasswordFileName = "User.passwords";
        protected String serverLogName = new String(name + ".log") ;
        protected static String messageFileName = "Messages.saved";
        protected static String welcomeMessageName = "WELCOME.TXT";
        protected static String portnumberParam = "-portnumber";
        protected static String usepasswordsParam = "-usepasswords";
        protected static String newusersParam = "-newusers";
        protected static String nographicsParam = "-nographics";
        protected static String chatlogsParam = "-chatlogs";
        protected static int DEFAULTPORT = 12468;
        protected URL myURL;
        protected int port;
        protected chatServerWindow myWindow;
        protected boolean graphics;
        protected boolean requirePasswords;
        protected boolean passwordEncryption;
        protected boolean allowNewUsers;
        protected boolean logChats;
        private Integer userIdCounter = new Integer(1);
        protected ServerSocket myServerSocket;
        protected Socket mySocket;
        protected ThreadGroup myThreadGroup;

        public Vector connections;
```

```
public Vector messages;
public chatChatRoom mainRoom;
public Vector chatRooms;
private FileOutputStream log;
protected chatPasswordEncryptor passwordEncryptor;
protected chatClientSocket administratorClient;
protected int currentConnections = 0;
protected int peakConnections = 0;
protected int totalConnections = 0;
protected boolean stop = false;
protected boolean externalShutdown = false;
protected SimpleDateFormat dateFormatter =
    new SimpleDateFormat("MMM dd, yyyy hh:mm a");
public chatServer(int askport, boolean passwords, boolean newusers,
                  boolean isgraphics, boolean islog)
{
    super("chat Chat server");
    port = askport;
    requirePasswords = passwords;
    allowNewUsers = newusers;
    graphics = isgraphics;
    logChats = islog;
    // Start up the log file
    try {
        System.out.println(serverLogName);
      File logFile = new File(serverLogName);
      log = new FileOutputStream(logFile);
    }
    catch (IOException e) {
      // Oops, no log file
      System.out.println(e);
```

```
    serverOutput("Unable to open " + serverLogName + " file\n");
}
// if user wants graphics, set up simple window
if (graphics)
    {
        myWindow =
            new chatServerWindow(this, "chat Chat v"
                                    + chat.VERSION + " Server");
        myWindow.setSize(400, 400);
        myWindow.setVisible(true);
    }
else
    {
        System.out.println("\nchat server status");
        System.out.println("Listening on port " + port);
        System.out.println("Connections:");
    }
try {
    myServerSocket = new ServerSocket(port);
}
catch (IOException e) {
    serverOutput("Couldn't create server socket\n");
    System.exit(1);
}
connections = new Vector();
messages = new Vector();
chatRooms = new Vector();
myThreadGroup = new ThreadGroup("Clients");
// Set up the object for encrypting passwords
passwordEncryptor = new chatPasswordEncryptor();
if (requirePasswords)
```

```
    // Try to make sure the password file exists
    try {
        new FileOutputStream(userPasswordFileName, true).close();
    }
    catch (IOException f) {}
// Create the initial 'main' chat room
mainRoom = new chatChatRoom("Main", "Administrator", false, null);
try {
    mainRoom.setLogging(logChats);
}
catch (IOException e) {
    serverOutput("Unable to start chat log for room " +
                mainRoom.name + "\n");
}
serverOutput("Reading message file\n");
readMessages();
serverOutput("Waiting for connections\n");
// To catch shutdown events, such as CTRL-C.  Note that this Java
// feature was only introduced as of 1.3, so if you are getting
// compilation errors, you should check your Javac version or comment
// out this bit.
double javaVersion = 0.0;
try {
    javaVersion = new Double(System.getProperty("java.version")
                        .substring(0, 3)).doubleValue();
}
catch (NumberFormatException e) {}
if (javaVersion >= 1.3)
    Runtime.getRuntime()
        .addShutdownHook(new chatServerShutdown(this));
start();
```

```
}
public int checkPassword(String fileName, String userName,
                         String password)
    throws Exception
{
    // Return true if the supplied user name/password combo match
    DataInputStream passwordStream = null;
    // Try to find the user name/password combo in the password file
    try {
      passwordStream =
          new DataInputStream(new FileInputStream(fileName));
     // Read entry by entry.
      while(true)
          {
            String tempUserName = "";
            String tempPassword = "";
            try {
                tempUserName = passwordStream.readUTF();
                tempPassword = passwordStream.readUTF();
            }
            catch (EOFException e) {
                // Reached the end of the file, no match
                //throw new Exception("User does not exist");
                return(chatCommand.ERROR_NOUSER);
            }
            // Do the user name and password match?
            if (tempUserName.equals(userName))
                {
                  if (tempPassword.equals(password))
                      return (chatCommand.LOGIN_SUCESS);
                  else
```

```
                        // The user name exists, but the password
                        // doesn't match
                        return (chatCommand.ERROR_WRONGPASSWORD);
                    }
                }
            }
            catch (IOException e) {
              serverOutput("Error reading password file: " + e.toString() +
                           "\n");
              return (chatCommand.ERROR_NOUSER);
            }
        }
        protected chatChatRoom findChatRoom(String roomName)
        {
            // This will return the chat room object with the corresponding
            // name
            chatChatRoom tempRoom = null;
            chatChatRoom returnRoom = null;
            // Is it the main chat room?
            if (mainRoom.name.equals(roomName))
            {
              returnRoom = mainRoom;
            }
            else
              {
                  for (int count = 0; count < chatRooms.size(); count ++)
                    {
                        tempRoom = (chatChatRoom)
                          chatRooms.elementAt(count);
                        System.out.println("temp :"+tempRoom);
                        if (tempRoom.name.equals(roomName))
```

```
                    {
                        returnRoom = tempRoom;
                        break;
                    }
                }
            }
        return (returnRoom);
}
public boolean isUserAllowed(chatChatRoom room,
                    chatClientSocket client, String password)
{
    // Return true if a user is allowed to enter a chat room.  False
    // otherwise.
    System.out.println("room.priv"+room.priv);
    // Private room?
    if (room.priv)
      {
        boolean invited = false;
        // Make sure that this user was either invited, or supplied
        // the correct password.  Check the password first
        if (!room.password.equals(password))
          {
            // No correct password supplied.  Check the list
            // of invitees
            for (int count2 = 0;
                count2 < room.invitedUsers.size(); count2 ++)
                {
                    Integer Id = (Integer) room.invitedUsers
                       .elementAt(count2);
                    System.out.println("ID"+Id);
                    System.out.println("Id.intValue()"+Id.intValue());
```

```
                        System.out.println("client.user.id"+client.user.id);
                        if (Id.intValue() == client.user.id)
                          {
                              invited = true;
                              break;
                          }
                    }
            if (!invited)
              {
                        try {
                          client.sendServerMessage(
                           "Not invited to/incorrect password " +
                           "for the private room " +
                           room.name);
                        }
                        catch (IOException e) {
                          disconnect(client, false);
                          return (false);
                        }
                        return (false);
                    }
              }
        }
    // Make sure the user has not been banned from the room
    for (int count1 = 0; count1 < room.bannedUserNames.size(); count1 ++)
      {
          if (room.bannedUserNames.elementAt(count1)
            .equals(client.user.name))
            {
                // This user has been banned from this room.  Send
                // them a message and quit
```

```
                    try {
                        client.sendBanUser(client.user.id, room.name);
                    }
                    catch (IOException e) {
                        disconnect(client, false);
                        return (false);
                    }
                    return (false);
                }
            }
        // The user is allowed
        return (true);
    }
    public synchronized void disconnect(chatClientSocket who,
                                        boolean notify)
    {
        int count;
        chatChatRoom chatRoom;
        if (notify)
            {
                try {
                    // Try to let the user know they're being disconnected
                    who.sendDisconnect(who.user.id, "You are being " +
                                       "disconnected.  Goodbye.");
                }
                catch (IOException e) {}
            }
        // Shut down the client socket
        who.shutdown();

        // Remove the user from their chat room.
```

```
try {
   who.leaveChatRoom();
}
catch (IOException e) {}
// Remove the user from our list of connections
synchronized (connections)
   {
         connections.removeElement(who);
         connections.trimToSize();
         currentConnections = connections.size();
   }
serverOutput("User " + who.user.name + " disconnected at "
            + dateFormatter.format(new Date()) + "\n");
serverOutput("There are " + currentConnections +
            " users connected\n");
// Tell all the other clients to ditch this user
for (count = 0; count < currentConnections; count ++)
   {
         chatClientSocket other = (chatClientSocket)
            connections.elementAt(count);
         try {
            other.sendDisconnect(who.user.id, "");
         }
         catch (IOException e) {}
   }
try {
   sleep(250);
}
catch (InterruptedException I) {}

if (graphics)
```

```
        {
            synchronized (myWindow.userList) {
              // Remove the user name from the list widget.  We do this
              // loop to make sure it hasn't already been removed,
              // since disconnect() can get called multiple times for
              // one disconnection.
              for (count = 0; count < myWindow.userList.getItemCount();
                    count ++)
                  {
                    if (myWindow.userList.getItem(
                                count).equals(who.user.name))
                        {
                          myWindow.userList.remove(who.user.name);
                          break;
                        }
                  }
            }
            myWindow.updateStats();
            if (currentConnections <= 1)
              myWindow.disconnectAll.setEnabled(false);
            if (currentConnections <= 0)
              myWindow.disconnect.setEnabled(false);
        }
    return;
}
public synchronized void disconnectAll(boolean notify)
{
    int count;

    // Loop backwards through all of the current connections
    for(count = (currentConnections - 1); count >= 0; count --)
```

```
            {
                chatClientSocket temp =
                  (chatClientSocket) connections.elementAt(count);
                if (temp == null)
                  continue;
                disconnect(temp, notify);
            }
        return;
}
 public void run()
 {
        while (!stop)
            {
                try {
                  mySocket = myServerSocket.accept();
                }
                catch (IOException e) {
                  serverOutput("Socket error\n");
                  try {
                        myServerSocket.close();
                  }
                  catch (IOException f) {
                        serverOutput("Couldn't close socket\n");
                  }
                  System.exit(1);
                }
                if (mySocket == null)
                  {
                        serverOutput("Server tried to start up NULL "
                                    + "socket\n");
                        try {
```

```
                    myServerSocket.close();
                }
                catch (IOException g) {
                    serverOutput("Couldn't close socket\n");
                }
                System.exit(1);
            }
            chatClientSocket cs =
                new chatClientSocket(this, mySocket, myThreadGroup);
        }
    }
    protected int getUserId()
    {
        // Returns a number to be used as a user Id
        int tmp;
        synchronized (userIdCounter)
            {
                tmp = userIdCounter.intValue();
                userIdCounter = new Integer(tmp + 1);
            }
        return (tmp);
    }
    protected void createNewUser(String userName, String encryptedPassword)
        throws Exception
    {
        // Create a new user acount.
        new chatUserTool().createUser(userName, encryptedPassword);
    }
    protected void serverOutput(String message)
    {
        if (graphics)
```

```
            myWindow.logWindow.append(message);
        else
            System.out.print(message);
        // Write it to the log file
        if (log != null)
            {
                try {
                    byte[] messagebytes = message.getBytes();
                    log.write(messagebytes);
                }
                catch (IOException F) {
                    if (graphics)
                        myWindow.logWindow
                            .append("Unable to write to log file\n");
                    else
                        System.out.print("Unable to write to log file\n");
                }
            }
        return;
    }
    protected void readMessages()
    {
        String tempFor = "";
        String tempFrom = "";
        String tempMessage = "";
        DataInputStream messageStream = null;
        try {
            messageStream =
                new DataInputStream(new FileInputStream(messageFileName));

            // Read entry by entry.
```

```
        while(true)
            {
              try {
                    tempFor = messageStream.readUTF();
                    tempFrom = messageStream.readUTF();
                    tempMessage = messageStream.readUTF();
              }
              catch (EOFException e) {
                    // Reached the end of the file
                    break;
              }
              messages.addElement(new chatMessage(tempFor, tempFrom,
                                                  tempMessage));
            }
        messageStream.close();
      }
      catch (IOException E) {}
      return;
}
protected void saveMessages()
{
      DataOutputStream messageStream = null;
      try {
        messageStream =
              new DataOutputStream(new FileOutputStream(messageFileName));
        for (int count = 0; count < messages.size(); count ++)
            {
              chatMessage tempMessage =
                    (chatMessage) messages.elementAt(count);

              messageStream.writeUTF(tempMessage.messageFor);
```

```
                messageStream.writeUTF(tempMessage.messageFrom);
                messageStream.writeUTF(tempMessage.text);
            }
        messageStream.close();
    }
    catch (IOException E) {
        serverOutput("Error writing the messages file\n");
    }
    return;
}
protected void shutdown()
{
    serverOutput("Server shutting down...\n");
    if (currentConnections > 0)
        {
            serverOutput("Disconnecting users\n");
            // Loop through all of the users, sending them the message
            // that the server is shutting down
            for (int count = 0; count < currentConnections; count ++)
                {
                    chatClientSocket who = (chatClientSocket)
                        connections.elementAt(count);
                    try {
                        who.sendDisconnect(who.user.id,
                                "The server is shutting down.  " +
                                "Goodbye.");
                    }
                    catch (IOException e) {}
                }
            // Make sure
            disconnectAll(true);
```

```
    }
else
    serverOutput("No users connected\n");
// Print some stats
serverOutput("Peak connections this session: "
            + peakConnections + "\n");
serverOutput("Total connections this session: "
            + totalConnections + "\n");
serverOutput("Saving user messages\n");
saveMessages();
serverOutput("Closing log file\n");
try {
    log.close();
}
catch (IOException F) {
    serverOutput("Unable to close server log file\n");
}
if (graphics)
    myWindow.dispose();
stop = true;
// If we aren't using the GUI window, we should provide some
// visual feedback that the server has terminated.
if (!graphics)
    {
        System.out.println("");
        System.out.println("chat server shutdown complete");
    }
// This function can be called by the chatServerShutdown thread
// when the server gets killed by an external signal.  If so, it
// sets the externalShutdown flag, and we shouldn't call the
// System.exit() function
```

```
        if (!externalShutdown)
            System.exit(0);
    }
    private static void usage()
    {
        System.out.println("\nchat Chat server usage:");
        System.out.println("java chatServer [" +
                           portnumberParam + " number] [" +
                           usepasswordsParam + "] [" +
                           newusersParam + "] [" +
                           nographicsParam + "] [" +
                           chatlogsParam + "]");
        return;
    }
    public static void main(String[] args)
    {
        int usePort = DEFAULTPORT;
        boolean reqPass = false;
        boolean allowNew = false;
        boolean useGraphics = true;
        boolean logChats = false;
        chatServer server;
        // Parse the arguments
        for (int count = 0; count < args.length; count ++)
          {
              if (args[count].equals(portnumberParam))
                {
                    if (++count < args.length)
                      {
                          try {
                              usePort = Integer.parseInt(args[count]);
```

```
                }
                catch (Exception E) {
                    System.out.println("\nchatServer: "
                            + "illegal port number "
                            + args[count]);
                    System.out.println("Type 'java "
                            + "chatServer -help' "
                            + "for usage information");
                    System.exit(1);
                }
            }
        }
    else if (args[count].equals(usepasswordsParam))
        reqPass = true;
    else if (args[count].equals(newusersParam))
        allowNew = true;
    else if (args[count].equals(nographicsParam))
        useGraphics = false;
    else if (args[count].equals(chatlogsParam))
        logChats = true;
    else if (args[count].equals("-help"))
        {
            usage();
            System.exit(1);
        }
    else
        {
            System.out.println("\nchatServer: unknown "
                    + "argument " + args[count]);
            System.out.println("Type 'java chatServer -help' "
                    + "for usage information");
```

```
                    System.exit(1);
                }
            }
        // Start the server
        server = new chatServer(usePort, reqPass, allowNew, useGraphics,
                                logChats);
        // Get a URL to describe the invocation directory
        try {
            server.myURL = new URL("file", "localhost", "./");
        }
        catch (Exception E) {
            System.out.println(E);
            System.exit(1);
        }
        return;
    }
}
```

Chat.java

```
import java.awt.*;
import java.net.*;
import java.io.*;
//public class chat extends Object implements Runnable
public class chat implements Runnable
{
    public static final String VERSION = "2.01";
    public static String usernameParam    = "-username";
    public static String passwordParam    = "-password";
    public static String servernameParam  = "-servername";
```

```
public static String portnumberParam   = "-portnumber";
public static String chatroomParam     = "-chatroom";
public static String widthParam        = "-xsize";
public static String heightParam       = "-ysize";
public static String nopasswordsParam  = "-nopasswords";
public static String locksettingsParam = "-locksettings";
public static String autoconnectParam  = "-autoconnect";
public static String hidecanvasParam   = "-hidecanvas";
private chatWindow window;
private URL myURL = null;
private String name = "";
private String password = "";
private String host = "";
private String port = "";
private String room = "";
private int windowWidth = 0;
private int windowHeight = 0;
private boolean requirePasswords = true;
private boolean lockSettings = false;
private boolean autoConnect = false;
private boolean showCanvas = true;
private void usage()
{
    System.out.println("\nBroadcast usage:");
    System.out.println("java Broadcast [" +
                       usernameParam + " name] [" +
                       passwordParam + " password] [" +
                       servernameParam + " host] [" +
                       portnumberParam + " port] [" +
                       chatroomParam + " room] [" +
                       widthParam + " number] [" +
```

```
                    heightParam + " number] [" +
                    nopasswordsParam + "] [" +
                    locksettingsParam + "] [" +
                    autoconnectParam + "] [" +
                    hidecanvasParam + "]");
        return;
}
private boolean parseArgs(String[] args)
{
        // Loop through any command line arguments
        for (int count = 0; count < args.length; count ++)
          {
              if (args[count].equals(usernameParam))
                {
                    if (++count < args.length)
                      name = args[count];
                }
              else if (args[count].equals(passwordParam))
                {
                    if (++count < args.length)
                      password = args[count];
                }
              else if (args[count].equals(servernameParam))
                {
                    if (++count < args.length)
                      host = args[count];
                }
              else if (args[count].equals(portnumberParam))
                {
                    if (++count < args.length)
                      port = args[count];
```

```
    }
  else if (args[count].equals(chatroomParam))
    {
        if (++count < args.length)
          room = args[count];
    }
  else if (args[count].equals(widthParam))
    {
        if (++count < args.length)
          windowWidth = Integer.parseInt(args[count]);
    }
  else if (args[count].equals(heightParam))
    {
        if (++count < args.length)
          windowHeight = Integer.parseInt(args[count]);
    }
  else if (args[count].equals(nopasswordsParam))
    requirePasswords = false;

  else if (args[count].equals(locksettingsParam))
    lockSettings = true;
  else if (args[count].equals(autoconnectParam))
    autoConnect = true;
  else if (args[count].equals(hidecanvasParam))
    showCanvas = false;
  else if (args[count].equals("-help"))
    {
        usage();
        return (false);
    }
  else
```

```
                {
                    System.out.println("\ncast: unknown argument "
                                + args[count]);
                    System.out.println("Type 'java cast -help' for "
                                + "usage information");
                    return (false);
                }
            }
        return (true);
}
public static void main(String[] args)
{
        chat firstinstance = new chat(args);
        firstinstance.run();
        return;
}
public chat(String[] args)
{
        // Get a URL to describe the invocation directory
        try {
            myURL = new URL("file", "localhost", "./");
                System.out.println( " The URL IS ::" +myURL);
        }
        catch (Exception E) {
            System.out.println(E);
            System.exit(1);
        }
        // Parse our args.  Only continue if successful
        if (!parseArgs(args))
            System.exit(1);
        // If "username" is blank, that's OK.  However, if the server and/or
```

```
// port are blank, we'll supply some default ones here
if ((host == null) || host.equals(""))
   host = "localhost";
if ((port == null) || port.equals(""))
   port = "12468";
// Open the window
window = new chatWindow(name, password, host, port, showCanvas,
                        myURL);
// Set the window width and height, if applicable
Dimension tmpSize = window.getSize();
if (windowWidth > 0)
   tmpSize.width = windowWidth;
if (windowHeight > 0)
   tmpSize.height = windowHeight;
window.setSize(tmpSize);
// Make the pretty icon
window.setIcon();
// Should the window prompt users for passwords automatically?
window.requirePassword = requirePasswords;

// Should the user name, server name, and port name be locked
// against user changes?
window.lockSettings = lockSettings;
// Show the window
window.show();
// Are we supposed to attempt an automatic connection?
if (autoConnect)
   window.connect();
// Is the user supposed to be placed in an initial chat room?
if (!room.equals(""))
   if (window.theClient != null)
```

```
                try {
                    window.theClient.sendEnterRoom(room, false, "");
                }
                catch (IOException e) {
                    window.theClient.lostConnection();
                    return;
                }
        // Done
        return;
    }
    public void run()
    {
        // Nothing to do here.
        return;
    }
}
```

chatUser.java

```
public class chatUser
{
    // This object keeps all the relevant information about a user for either
    // the server or the client
    protected int id;
    protected String name;
    protected String password;
    protected String additional;
    private String chatroomName;      // Used by the client
    private chatChatRoom chatroom; // Used by the server
    chatUser(chatServer server)
```

```
{
    // This constructor will be used by the server, since it automatically
    // assigns a new user Id
    System.out.println("This is to assign ID in chatUser");
    id = server.getUserId();
    System.out.println("This is to assign ID in chatUser"+id);
    name = "newuser" + id;
    password = "";
    additional = "";
    chatroomName = "";
    chatroom = null;
}
chatUser(int i, String nm, String pw, String add)
{
    // This constructor will be used by the client, with information
    // supplied by the server
    System.out.println("This is to assign ID in chatUser i"+i);
    System.out.println("This is to assign ID in chatUser nm"+nm);
    System.out.println("This is to assign ID in chatUser pw"+pw);
    System.out.println("This is to assign ID in chatUser add"+add);
    id = i;
    name = nm;
    password = pw;
    additional = add;
    chatroomName = "";
    chatroom = null;
}
public void setChatRoom(chatChatRoom newRoom)
{
    // Used by the server to set the user's chat room
    chatroom = newRoom;
```

```
        System.out.println( "In setChatRoom " +chatroom);
    }
    public void setChatRoomName(String newRoomName)
    {
        // Used by the client to set the user's chat room name
        System.out.println( "room name :" +newRoomName);
        chatroomName = newRoomName;
    }
    public String getChatRoomName()
    {
        if (chatroom != null)
        {
          System.out.println( "THe chatroom ::" +chatroom.name);
           return (chatroom.name);
        }
        else
        {
          System.out.println( "THe else part"+chatroomName);
           return (chatroomName);
        }
    }
    public chatChatRoom getChatRoom()
    {
        System.out.println("THe getchatroom :" +chatroom);
        return (chatroom);
    }
}
```

chatCommand.java

```
/*
 Here is the list of commands, with brief summary information
 NAME                ARGS
 ----                ----
 setproto        version
 noop
 ping
 connect      name
 userinfo        user_id name password encrypted additional
 servermess    message
 disconnect    user_id message
 roomlist        #rooms [name creator private invited #users username...] ...
 invite from_id roomname invite_id
 enterroom     from_id          roomname private password encrypted
 bootuser      from_id roomname boot_id
 banuser       from_id  roomname ban_id
 allowuser     from_id roomname allow_id
 activity      from_id type #for [for_id...]
 chattext        from_id private colour data #for [for_id...]
 line            from_id colour x0 y0 x1 y1 thick #for [for_id...]
 rect            from_id colour x0 y0 width height thick fill #for [for_id...]
 oval            from_id colour x0 y0 width height thick fill #for [for_id...]
 drawtext        from_id colour x y type attr size text #for [for_id...]
 drawpicture   from_id x y length data #for [for_id...]
 clearcanv       from_id #for [for_id...]
 pageuser        from_id #for [for_id...]
 instantmess   from_id for_id message
 leavemess     from_id for_name message
 readmess        from_id
```

```
  storedmess    number [sender_name message]...
  error         from_id errorcode #for [for_id...]
*/


public class chatCommand
{
   // The list of command type ids
   public static final short SETPROTO   = 1;
   public static final short NOOP       = 2;
   public static final short PING       = 3;
   public static final short CONNECT    = 4;
   public static final short USERINFO   = 5;
   public static final short SERVERMESS  = 6;
   public static final short DISCONNECT  = 7;
   public static final short ROOMLIST    = 8;
   public static final short INVITE      = 9;
   public static final short ENTERROOM   = 10;
   public static final short BOOTUSER    = 11;
   public static final short BANUSER     = 12;
   public static final short ALLOWUSER   = 13;
   public static final short ACTIVITY    = 14;
   public static final short CHATTEXT    = 15;
   public static final short LINE        = 16;
   public static final short RECT        = 17;
   public static final short OVAL        = 18;
   public static final short DRAWTEXT    = 19;
   public static final short DRAWPICTURE = 20;
   public static final short CLEARCANV   = 21;
   public static final short PAGEUSER    = 22;
   public static final short INSTANTMESS = 23;
```

```
    public static final short LEAVEMESS   = 24;
    public static final short READMESS    = 25;
    public static final short STOREDMESS  = 26;
    public static final short ERROR       = 27;

    // Activity subtypes
    public static final short ACTIVITY_TYPING  = 1;
    public static final short ACTIVITY_DRAWING = 2;

    // Error subtypes
    public static final short ERROR_NOPAGE  = 1;
    public static final short ERROR_NOSOUND = 2;

    //Checking Login Error Type

    public static final int ERROR_NOUSER = 1;
    public static final int ERROR_WRONGPASSWORD = 2;
    public static final int LOGIN_SUCESS = 3;
}
```

chatMessage.java

```
public class chatMessage
{
    public String messageFor;
    public String messageFrom;
    public String text;
    public chatMessage(String whoFor, String whoFrom, String info)
    {
        messageFor = whoFor;
```

```
        messageFrom = whoFrom;
        text = info;
    }
}
```

chatRoomInfo.java

```
import java.util.*;
public class chatRoomInfo
{
    String name;
    String creatorName;
    boolean priv;
    boolean invited;
    boolean roomOwner;
    Vector userNames;
    chatRoomInfo()
    {
        name = "";
        creatorName = "";
        priv = false;
        invited = false;
        roomOwner = false;
        userNames = new Vector();
    }
}
```

chatCreateRoom.java

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
public class chatCreateRoom
   extends JDialog
   implements ActionListener, ItemListener, KeyListener, WindowListener
{
   protected chatWindow parentWindow;
   protected JLabel roomNameLabel;
   protected JTextField roomName;
   protected Checkbox priv;
   protected JLabel passwordLabel;
   protected JPasswordField password;                    //changed here
from textfield
   protected JLabel passwordWarningLabel1;
   protected JLabel passwordWarningLabel2;
   protected JButton ok;
   protected JButton cancel;

   protected GridBagLayout myLayout;
   protected GridBagConstraints myConstraints;
   chatCreateRoom(JFrame parent)
   {
        super(parent, "Create a chat room", true);
        parentWindow = (chatWindow) parent;
        // Make all of the widgets
        myLayout = new GridBagLayout();
        myConstraints = new GridBagConstraints();
```

```
Container con=getContentPane();
con.setLayout(myLayout);
myConstraints.insets.top = 0; myConstraints.insets.bottom = 0;
myConstraints.insets.right = 5; myConstraints.insets.left = 5;
myConstraints.anchor = myConstraints.WEST;
myConstraints.weightx = 1.0; myConstraints.weighty = 1.0;

roomNameLabel = new JLabel("Room name:");
roomNameLabel.setFont(parentWindow.smallFont);
myConstraints.gridx = 0; myConstraints.gridy = 0;
myConstraints.gridheight = 1; myConstraints.gridwidth = 2;
myConstraints.fill = myConstraints.NONE;
myLayout.setConstraints(roomNameLabel, myConstraints);
con.add(roomNameLabel);

roomName = new JTextField(30);
roomName.setFont(parentWindow.smallFont);
roomName.addKeyListener(this);
roomName.setEditable(true);
roomName.setEnabled(true);
myConstraints.gridx = 0; myConstraints.gridy = 1;
myConstraints.gridheight = 1; myConstraints.gridwidth = 2;
myConstraints.fill = myConstraints.BOTH;
myLayout.setConstraints(roomName, myConstraints);
con.add(roomName);

priv = new Checkbox("room is private", false);
priv.setFont(parentWindow.smallFont);
priv.setEnabled(true);
priv.addItemListener(this);
myConstraints.gridx = 0; myConstraints.gridy = 2;
```

```
myConstraints.gridheight = 1; myConstraints.gridwidth = 2;
myConstraints.fill = myConstraints.NONE;
myConstraints.insets.top = 5; myConstraints.insets.bottom = 0;
myLayout.setConstraints(priv, myConstraints);
con.add(priv);

passwordLabel = new JLabel("Password:");
passwordLabel.setFont(parentWindow.smallFont);
passwordLabel.setEnabled(priv.getState());
myConstraints.gridx = 0; myConstraints.gridy = 3;
myConstraints.gridheight = 1; myConstraints.gridwidth = 2;
myConstraints.fill = myConstraints.NONE;
myConstraints.insets.top = 0; myConstraints.insets.bottom = 0;
myLayout.setConstraints(passwordLabel, myConstraints);
con.add(passwordLabel);

password = new JPasswordField();
password.setFont(parentWindow.smallFont);
password.addKeyListener(this);
password.setEditable(true);
password.setEnabled(priv.getState());
password.setEchoChar(new String("*").charAt(0));
myConstraints.gridx = 0; myConstraints.gridy = 4;
myConstraints.gridheight = 1; myConstraints.gridwidth = 2;
myConstraints.fill = myConstraints.BOTH;
myLayout.setConstraints(password, myConstraints);
con.add(password);

passwordWarningLabel1 =
   new JLabel("Warning: Your Java client cannot encrypt your");
passwordWarningLabel1
```

```
   .setVisible(!parentWindow.passwordEncryptor.canEncrypt);
passwordWarningLabel1.setFont(chatWindow.XsmallFont);
passwordWarningLabel1.setEnabled(priv.getState());
myConstraints.gridx = 0; myConstraints.gridy = 5;
myConstraints.gridheight = 1; myConstraints.gridwidth = 2;
myConstraints.fill = myConstraints.NONE;
myLayout.setConstraints(passwordWarningLabel1, myConstraints);
con.add(passwordWarningLabel1);

passwordWarningLabel2 =
   new JLabel("passwords.  They will be sent as plain text.");
passwordWarningLabel2
   .setVisible(!parentWindow.passwordEncryptor.canEncrypt);
passwordWarningLabel2.setFont(chatWindow.XsmallFont);
passwordWarningLabel2.setEnabled(priv.getState());
myConstraints.gridx = 0; myConstraints.gridy = 6;
myConstraints.gridheight = 1; myConstraints.gridwidth = 2;
myLayout.setConstraints(passwordWarningLabel2, myConstraints);
con.add(passwordWarningLabel2);

myConstraints.insets.top = 5; myConstraints.insets.bottom = 5;

ok = new JButton("Ok");
ok.setFont(parentWindow.smallFont);
ok.addActionListener(this);
ok.addKeyListener(this);
ok.setEnabled(false);
myConstraints.gridx = 0; myConstraints.gridy = 7;
myConstraints.gridheight = 1; myConstraints.gridwidth = 1;
myConstraints.fill = myConstraints.NONE;
myConstraints.anchor = myConstraints.EAST;
```

```
myConstraints.insets.right = 0; myConstraints.insets.left = 5;
myLayout.setConstraints(ok, myConstraints);
con.add(ok);

cancel = new JButton("Cancel");
cancel.setFont(parentWindow.smallFont);
cancel.addActionListener(this);
cancel.addKeyListener(this);
cancel.setEnabled(true);
myConstraints.gridx = 1; myConstraints.gridy = 7;
myConstraints.gridheight = 1; myConstraints.gridwidth = 1;
myConstraints.fill = myConstraints.NONE;
myConstraints.anchor = myConstraints.WEST;
myConstraints.insets.right = 5; myConstraints.insets.left = 0;
myLayout.setConstraints(cancel, myConstraints);
con.add(cancel);

// register to receive the various events
addKeyListener(this);
addWindowListener(this);

// show the window and get going
setSize(200, 200);
pack();
setLocation((((((parentWindow.getBounds()).width)
              - ((getSize()).width)) / 2)
             + ((parentWindow.getLocation()).x)),
            (((((parentWindow.getBounds()).height)
              - ((getSize()).height)) / 2)
             + ((parentWindow.getLocation()).y)));
```

```
        setVisible(true);
        roomName.requestFocus();
    }

    protected void goCreate()
    {
        if (!roomName.getText().equals(""))
            {
                String pword = "";
                if (priv.getState())
                    pword = parentWindow.passwordEncryptor
                            .encryptPassword(password.getText());

                parentWindow.canvas.clear();

                try {
                    parentWindow.theClient
                            .sendEnterRoom(roomName.getText(), priv.getState(),
                                    pword);
                }
                catch (IOException e) {
                    parentWindow.theClient.lostConnection();
                    return;
                }
                parentWindow.currentRoom.name = roomName.getText();
                parentWindow.roomOwner(true);
                return;
            }
    }
    public void actionPerformed(ActionEvent E)
    {
```

```
        if (E.getSource() == ok)
          {
               goCreate();
               dispose();
               return;
          }
        if (E.getSource() == cancel)
          {
               dispose();
               return;
          }
}
public void itemStateChanged(ItemEvent E)
{
        if (E.getSource() == priv)
          {
               boolean state = priv.getState();
               passwordLabel.setEnabled(state);
               password.setEnabled(state);
               passwordWarningLabel1.setEnabled(state);
               passwordWarningLabel2.setEnabled(state);
               return;
          }
}
public void keyPressed(KeyEvent E)
{
}
public void keyReleased(KeyEvent E)
{
        if (E.getKeyCode() == E.VK_ENTER)
          {
```

```
            if (E.getSource() == cancel)
              {
                  dispose();
                  return;
              }
            else
              {
                  goCreate();
                  dispose();
                  return;
              }
        }
      else if (E.getSource() == roomName)
        {
            if (roomName.getText().equals(""))
              ok.setEnabled(false);
            else
              ok.setEnabled(true);
            return;
        }
}
public void keyTyped(KeyEvent E)
{
}
public void windowActivated(WindowEvent E)
{
}
public void windowClosed(WindowEvent E)
{
}
public void windowClosing(WindowEvent E)
```

```
    {
        dispose();
        return;
    }
    public void windowDeactivated(WindowEvent E)
    {
    }
    public void windowDeiconified(WindowEvent E)
    {
    }
    public void windowIconified(WindowEvent E)
    {
    }
    public void windowOpened(WindowEvent E)
    {
    }
}
```

ChatMessagingDialog:

```
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import java.io.*;
import javax.swing.*;


public class chatMessagingDialog
    extends JDialog
    implements ActionListener, ItemListener, KeyListener, WindowListener
```

```
{
   protected chatWindow parentWindow;

   protected JButton readMessages;
   protected JLabel instantForLabel;
   protected java.awt.List allUsersList;
   protected JLabel saveForLabel;
   protected JTextField saveFor;
   protected JLabel messageTextLabel;
   protected JTextArea messageText;
   protected JButton ok;
   protected JButton cancel;
   protected JPanel p1;
   protected JPanel p2;
   protected GridBagLayout myLayout;
   protected GridBagConstraints myConstraints;


   public chatMessagingDialog(chatWindow parent)
   {
        super(parent, "Messaging", false);

        parentWindow = parent;
        myLayout = new GridBagLayout();
        myConstraints = new GridBagConstraints();
Container con=getContentPane();
        con.setLayout(myLayout);

        myConstraints.insets = new Insets(0, 5, 0, 5);

        p1 = new JPanel();
```

```
p1.setLayout(myLayout);

readMessages = new JButton("Read saved messages");
readMessages.addActionListener(this);
readMessages.addKeyListener(this);
myConstraints.gridx = 0; myConstraints.gridy = 0;
myConstraints.gridheight = 1; myConstraints.gridwidth = 1;
myConstraints.fill = myConstraints.BOTH;
myConstraints.anchor = myConstraints.WEST;
myConstraints.insets.top = 5; myConstraints.insets.bottom = 0;
myLayout.setConstraints(readMessages, myConstraints);
p1.add(readMessages);

instantForLabel = new JLabel("Send instant message to:");
myConstraints.gridx = 0; myConstraints.gridy = 1;
myConstraints.gridheight = 1; myConstraints.gridwidth = 1;
myConstraints.fill = myConstraints.NONE;
myConstraints.anchor = myConstraints.WEST;
myConstraints.insets.top = 0; myConstraints.insets.bottom = 0;
myLayout.setConstraints(instantForLabel, myConstraints);
p1.add(instantForLabel);

allUsersList = new java.awt.List(5);
allUsersList.setFont(parentWindow.smallFont);
allUsersList.addItemListener(this);
allUsersList.addKeyListener(this);
allUsersList.setMultipleMode(false);
myConstraints.gridx = 0; myConstraints.gridy = 2;
myConstraints.weightx = 1.0; myConstraints.weighty = 1.0;
myConstraints.fill = myConstraints.BOTH;
myConstraints.anchor = myConstraints.WEST;
```

```
myLayout.setConstraints(allUsersList, myConstraints);
p1.add(allUsersList);

saveForLabel = new JLabel("- OR - Save message for user name:");
myConstraints.gridx = 0; myConstraints.gridy = 3;
myConstraints.gridheight = 1; myConstraints.gridwidth = 1;
myConstraints.fill = myConstraints.NONE;
myConstraints.anchor = myConstraints.WEST;
myLayout.setConstraints(saveForLabel, myConstraints);
p1.add(saveForLabel);

saveFor = new JTextField(20);
saveFor.addKeyListener(this);
myConstraints.gridx = 0; myConstraints.gridy = 4;
myConstraints.gridheight = 1; myConstraints.gridwidth = 1;
myConstraints.fill = myConstraints.BOTH;
myConstraints.anchor = myConstraints.WEST;
myLayout.setConstraints(saveFor, myConstraints);
p1.add(saveFor);

messageTextLabel = new JLabel("Message to send:");
myConstraints.gridx = 0; myConstraints.gridy = 5;
myConstraints.gridheight = 1; myConstraints.gridwidth = 1;
myConstraints.fill = myConstraints.NONE;
myConstraints.anchor = myConstraints.WEST;
myLayout.setConstraints(messageTextLabel, myConstraints);
p1.add(messageTextLabel);

messageText =
   new JTextArea("", 2, 20 );//,
TextArea.SCROLLBARS_VERTICAL_ONLY);
```

```
messageText.addKeyListener(this);
myConstraints.gridx = 0; myConstraints.gridy = 6;
myConstraints.gridheight = 1; myConstraints.gridwidth = 1;
myConstraints.fill = myConstraints.BOTH;
myConstraints.anchor = myConstraints.WEST;
myLayout.setConstraints(messageText, myConstraints);
p1.add(messageText);

myConstraints.gridx = 0; myConstraints.gridy = 0;
myConstraints.gridheight = 1; myConstraints.gridwidth = 1;
myConstraints.fill = myConstraints.BOTH;
myConstraints.anchor = myConstraints.CENTER;
myConstraints.insets.top = 0; myConstraints.insets.bottom = 0;
myConstraints.insets.left = 0; myConstraints.insets.right = 0;
myLayout.setConstraints(p1, myConstraints);
con.add(p1);

p2 = new JPanel();
p2.setLayout(myLayout);

ok = new JButton("Ok");
ok.addActionListener(this);
ok.addKeyListener(this);
myConstraints.gridx = 0; myConstraints.gridy = 0;
myConstraints.gridheight = 1; myConstraints.gridwidth = 1;
myConstraints.fill = myConstraints.NONE;
myConstraints.anchor = myConstraints.EAST;
myConstraints.insets.top = 5; myConstraints.insets.bottom = 5;
myConstraints.insets.left = 5; myConstraints.insets.right = 0;
myLayout.setConstraints(ok, myConstraints);
p2.add(ok);
```

```
cancel = new JButton("Cancel");
cancel.addActionListener(this);
cancel.addKeyListener(this);
myConstraints.gridx = 1; myConstraints.gridy = 0;
myConstraints.gridheight = 1; myConstraints.gridwidth = 1;
myConstraints.fill = myConstraints.NONE;
myConstraints.anchor = myConstraints.WEST;
myConstraints.insets.top = 5; myConstraints.insets.bottom = 5;
myConstraints.insets.left = 0; myConstraints.insets.right = 5;
myLayout.setConstraints(cancel, myConstraints);
p2.add(cancel);

myConstraints.gridx = 0; myConstraints.gridy = 1;
myConstraints.gridheight = 1; myConstraints.gridwidth = 1;
myConstraints.fill = myConstraints.BOTH;
myConstraints.anchor = myConstraints.CENTER;
myConstraints.insets.top = 0; myConstraints.insets.bottom = 0;
myConstraints.insets.left = 0; myConstraints.insets.right = 0;
myLayout.setConstraints(p2, myConstraints);
con.add(p2);

setSize(600,400);
pack();
setResizable(false);
setLocation((((((parentWindow.getBounds()).width) -
              ((getSize()).width)) / 2)
             + ((parentWindow.getLocation()).x)),
            (((((parentWindow.getBounds()).height) -
              ((getSize()).height)) / 2)
             + ((parentWindow.getLocation()).y)));
```

```
        addKeyListener(this);
        addWindowListener(this);
        setVisible(true);
        updateLists();
        allUsersList.requestFocus();
    }

    public void updateLists()
    {
        // Update our lists.  This is called when something changes, such
        // as a user logs on/off

        if (allUsersList.getItemCount() > 0)
          allUsersList.removeAll();

        Vector usersVector = parentWindow.theClient.userList;

        // Add all of the connected users to the 'all users' list
        for (int count = 0; count < usersVector.size(); count ++)
          {
              chatUser user = (chatUser) usersVector.elementAt(count);
              allUsersList.add(user.name);
          }

        return;
    }

    private void sendMessage()
    {
        String instantUser = allUsersList.getSelectedItem();
```

```
String saveUser = saveFor.getText();

if ((instantUser == null) && saveUser.equals(""))
  {
      new chatInfoDialog(parentWindow, "Need recipient", true,
                         "You must specify a recipient for "
                         + "the message!");
      return;
  }

if (parentWindow.connected != true)
  {
      new chatInfoDialog(parentWindow, "Not connected", true,
                         "Must be connected first!");
      return;
  }

// If the user has typed a name, leave the message on the server
// for that user
if (!saveUser.equals(""))
  try {
      // Send the message to the server
      parentWindow.theClient.sendLeaveMess(saveUser,
                                           messageText.getText());
  }
  catch (IOException e) {
      parentWindow.theClient.lostConnection();
      return;
  }
else
  {
```

```
            // Send an instant message.  First we need to find the user
            // id that matches the name that's selected

            Vector usersVector = parentWindow.theClient.userList;

            for (int count = 0; count < usersVector.size(); count ++)
              {
                  chatUser user =
                    (chatUser) usersVector.elementAt(count);

                  if (user.name.equals(instantUser))
                    {
                        try {
                          parentWindow.theClient
                                .sendInstantMess(user.id,
                                        messageText.getText());
                        }
                        catch (IOException e) {
                          parentWindow.theClient.lostConnection();
                          return;
                        }
                        break;
                    }
              }
        }

    return;
}


public void actionPerformed(ActionEvent E)
```

```
{
    if (E.getSource() == readMessages)
      {
          try {
            parentWindow.theClient.sendReadMess();
          }
          catch (IOException e) {
            parentWindow.theClient.lostConnection();
          }
          dispose();
          return;
      }

    else if (E.getSource() == ok)
      {
          sendMessage();
          dispose();
          return;
      }

    else if (E.getSource() == cancel)
      {
          dispose();
          return;
      }
}

public void itemStateChanged(ItemEvent E)
{
    if (E.getSource() == allUsersList)
      {
```

```
                // If a user name has been selected, empty out the saveFor
                // field
                if (allUsersList.getSelectedItem() != null)
                   saveFor.setText("");
                return;
            }
}

public void keyPressed(KeyEvent E)
{
}

public void keyReleased(KeyEvent E)
{
    if (E.getKeyCode() == E.VK_ENTER)
        {
            if (E.getSource() == readMessages)
               {
                    try {
                       parentWindow.theClient.sendReadMess();
                    }
                    catch (IOException e) {
                       parentWindow.theClient.lostConnection();
                    }
                    dispose();
                    return;
               }

            else if ((E.getSource() == ok) ||
               (E.getSource() == saveFor) ||
               (E.getSource() == messageText))
```

```
                {
                    sendMessage();
                    dispose();
                    return;
                }

            else if (E.getSource() == cancel)
                {
                    dispose();
                    return;
                }
        }

    else if (E.getSource() == saveFor)
        {
            // The user is typing a name, so make sure no names are
            // selected in the allUsersList
            int items = allUsersList.getRows();
            for (int count = 0; count < items; count ++)
               allUsersList.deselect(count);
        }
}

public void keyTyped(KeyEvent E)
{
}

public void windowActivated(WindowEvent E)
{
}
```

```
    public void windowClosed(WindowEvent E)
    {
    }

    public void windowClosing(WindowEvent E)
    {
        dispose();
        return;
    }

    public void windowDeactivated(WindowEvent E)
    {
    }

    public void windowDeiconified(WindowEvent E)
    {
    }

    public void windowIconified(WindowEvent E)
    {
    }

    public void windowOpened(WindowEvent E)
    {
    }
}
```