Tutorial for the Wunderbot Web-Controlled Camera

By Adam Steiner '05

Idea

Provide a means to control the Wunderbot from a website, anywhere in the world.

Software/Platforms in Use/ Specifics

LabView 7.1
Running on: Wunderbot
Purpose: All Wunderbot functions are now in LabView.

Java v1.5
Running on: LAM, User Computer
Purpose: Fast implementation, Changes made easy.

Microsoft Windows 2000
Running on: Wunderbot, LAM

Unreal Media Streaming Player ActiveX Control
Running on: User Computer
Purpose: Receives video signal and displays to user.
Reason for choosing: Only free streaming software that has barely any lag.

Unreal Media Live Server
Running On: Wunderbot
Purpose: Encoder of Web cam video signal.

Unreal Media Server
Running On: LAM
Purpose: Takes encoded Web cam video and broadcasts to user.

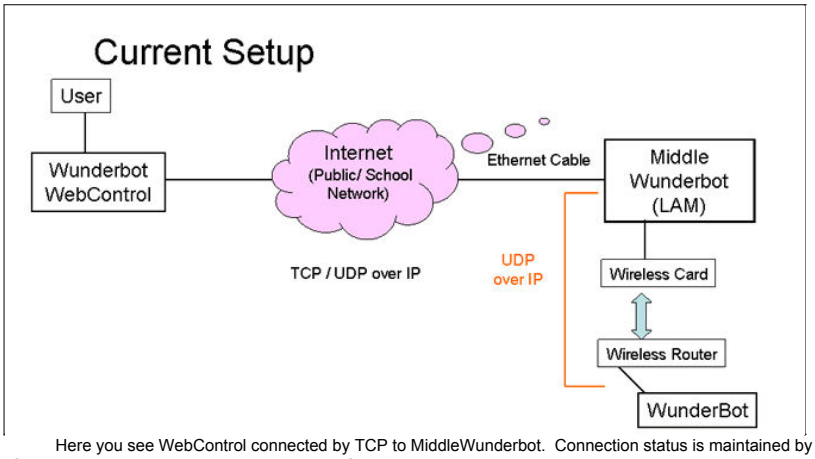
Names and Description:

WunderbotWebControl: Java applet. Interface on user's screen that provides camera movement functions. Also called WebControl
MiddleWunderbot: Java application: Runs on Lam that provides a connection point for WebControl, and reports if Wunderbot is able to be used. Forwards commands through (UDP packet parsing).
UDP.vi: LabView vi. Takes data received by Wunderbot and determines what camera motion requested. Request is transferred to appropriate VIs (See Brian Moran- Bernard for more information about this.)
Wunderbot: Robot. Windows 2000 Operating System

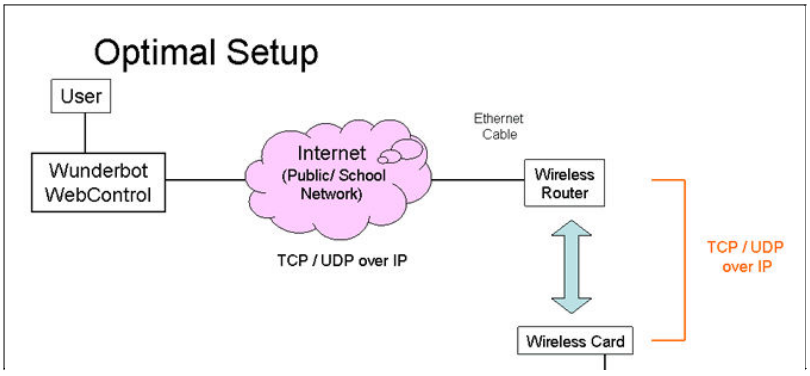
How To Use

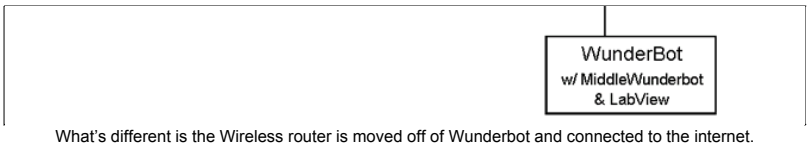
Follow this link . This page checks the user's computer for needed software (Windows 98 or later, Java 1.4.2 or later, Unreal Streaming Player Control). If these conditions are not met, instructions are given on how to meet them. If they are met, then a link appears that takes the user to the next page. The main page contains the embedded Unreal Media Streaming Player ActiveX Control and WunderbotWebControl Java Applet. On loading of WebControl, the user is presented with a prompt for the address of MiddleWunderbot. If connected properly, the video stream begins and camera control is enabled.

Data Flow Layout



Here you see WebControl connected by TCP to MiddleWunderbot. Connection status is maintained by TCP. Real time commands are done by UDP. Commands are relayed to Wunderbot through UDP by MiddleWunderbot.

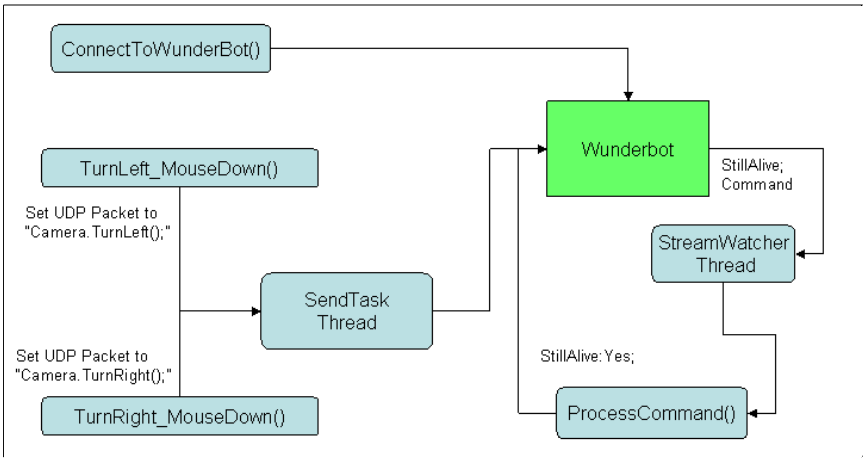




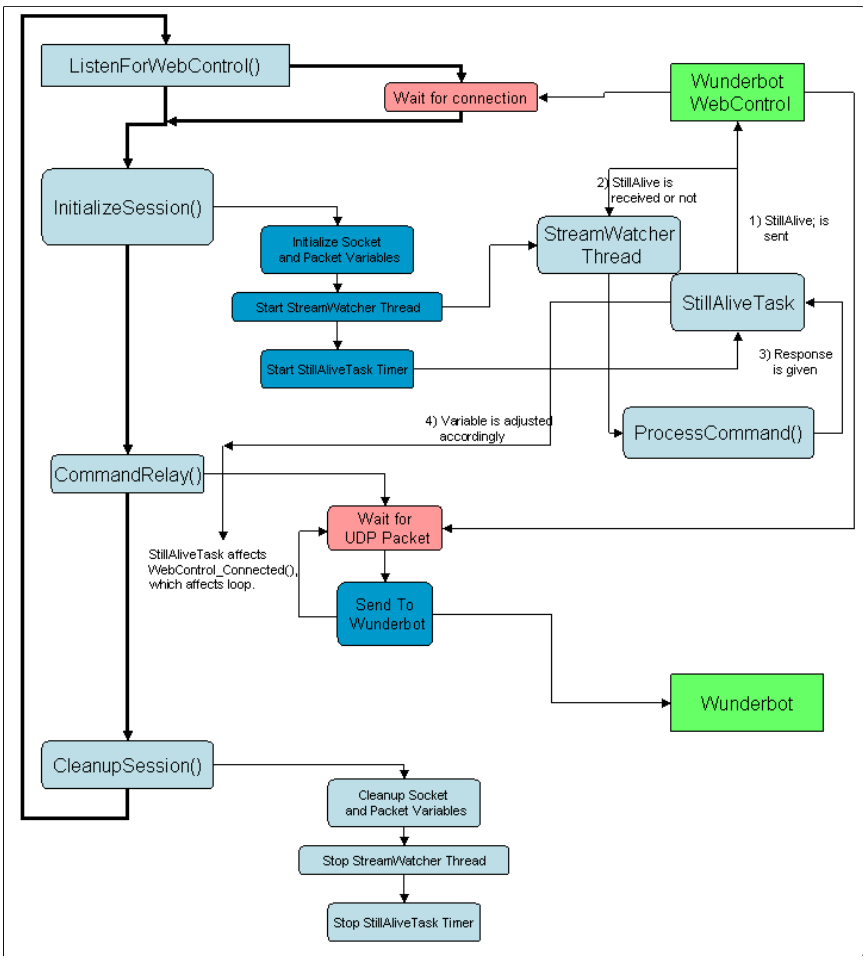
What's different is the Wireless router is moved off of Wunderbot and connected to the internet. Through port forwarding and its firewall, Wunderbot is kept safe from attack and LAM is eliminated from the picture completely. MiddleWunderbot is put on Wunderbot for connection status and parsing. Video and command relaying is done by the wireless router faster and more efficiently. Connection status is maintained by TCP. Real time commands are done by UDP. Commands are relayed to Wunderbot through UDP.

Flowcharts

WunderbotWebControl



MiddleWunderbot



LabView VI

Code

WunderbotWebControl \ MainApplet.java

```
/*
 * MainApplet.java
 *
 * Created on November 1, 2005, 4:08 PM
 */
import java.net.*;
import java.io.*;
import java.util.*;
import java.lang.StringBuilder;
import netscape.javascript.JSObject;

public class MainApplet extends javax.swing.JApplet {

    public void init() {
        initComponents();
    }

    /** This method is called from within the init() method to
     * initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is
     * always regenerated by the Form Editor.
     */
    // <editor-fold defaultstate="collapsed" desc=" Generated Code ">
    private void initComponents() {
        buttonGroup1 = new javax.swing.ButtonGroup();
        jButton1 = new javax.swing.JButton();
        STATUS_LABEL = new javax.swing.JLabel();
        jPanel1 = new javax.swing.JPanel();
        btnCameraLeft = new javax.swing.JButton();
        btnCameraRight = new javax.swing.JButton();
        jLabel2 = new javax.swing.JLabel();
        txtHostname = new javax.swing.JTextField();
        jButton2 = new javax.swing.JButton();

        getContentPane().setLayout(null);

        jButton1.setText("Connect");
        jButton1.setAlignmentY(0.0F);
        jButton1.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
        jButton1.addMouseListener(new java.awt.event.MouseAdapter() {
            public void mouseClicked(java.awt.event.MouseEvent evt) {
                jButton1MouseClicked(evt);
            }
        });

        getContentPane().add(jButton1);
        jButton1.setBounds(110, 100, 90, 30);

        STATUS_LABEL.setBackground(new java.awt.Color(204, 51, 0));
        STATUS_LABEL.setText("Go");
        getContentPane().add(STATUS_LABEL);
        STATUS_LABEL.setBounds(130, 24, 90, 16);

        jPanel1.setLayout(null);

        jPanel1.setBorder(javax.swing.BorderFactory.createEtchedBorder());
        btnCameraLeft.setText("Left");
        btnCameraLeft.addMouseListener(new java.awt.event.MouseAdapter() {
            public void mousePressed(java.awt.event.MouseEvent evt) {
                btnCameraLeftMousePressed(evt);
            }
            public void mouseReleased(java.awt.event.MouseEvent evt) {
                btnCameraLeftMouseReleased(evt);
            }
        });

        jPanel1.add(btnCameraLeft);
        btnCameraLeft.setBounds(10, 10, 60, 30);

        btnCameraRight.setText("Right");
        btnCameraRight.addMouseListener(new java.awt.event.MouseAdapter() {
            public void mousePressed(java.awt.event.MouseEvent evt) {
                btnCameraRightMousePressed(evt);
            }
            public void mouseReleased(java.awt.event.MouseEvent evt) {
```

```
            btnCameraRightMouseReleased(evt);
        }
    });

    jPanel1.add(btnCameraRight);
    btnCameraRight.setBounds(100, 10, 70, 30);

    getContentPane().add(jPanel1);
    jPanel1.setBounds(20, 40, 180, 50);

    jLabel2.setText("Camera Control");
    getContentPane().add(jLabel2);
    jLabel2.setBounds(20, 20, 110, 20);

    txtHostname.setText("localhost");
    getContentPane().add(txtHostname);
    txtHostname.setBounds(20, 100, 70, 22);

    jButton2.setText("jButton2");
    jButton2.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            jButton2ActionPerformed(evt);
        }
    });

    getContentPane().add(jButton2);
    jButton2.setBounds(130, 140, 81, 25);

}// </editor-fold>


// Button Press Events
// When the user presses down on a button, the command is placed into the data packet.
//  SendCommands is made true, which tells SendTask TimerTask to send the packet at .1 second intervals
// When the user releases the button, SendCommands is made false
private void btnCameraRightMousePressed(java.awt.event.MouseEvent evt) {
    if(!Wunderbot_Connected){return;}
    udpCommandPacket.setData("Camera.TurnRight;".getBytes());
    SendCommands = true;
}
    private void btnCameraRightMouseReleased(java.awt.event.MouseEvent evt) {
        SendCommands = false;
}

private void btnCameraLeftMouseReleased(java.awt.event.MouseEvent evt) {
    SendCommands = false;
}
    private void btnCameraLeftMousePressed(java.awt.event.MouseEvent evt) {
        if(!Wunderbot_Connected){return;}
        udpCommandPacket.setData("Camera.TurnLeft;".getBytes());
        SendCommands = true;
}
        private void jButton1MouseClicked(java.awt.event.MouseEvent evt) {
            ConnectToWunderbot(txtHostname.getText());
}


// Variables declaration - do not modify
private javax.swing.JLabel STATUS_LABEL;
private javax.swing.JButton btnCameraLeft;
private javax.swing.JButton btnCameraRight;
private javax.swing.ButtonGroup buttonGroup1;
private javax.swing.JButton jButton1;
private javax.swing.JButton jButton2;
private javax.swing.JLabel jLabel2;
private javax.swing.JPanel jPanel1;
private javax.swing.JPanel panCamera;
private javax.swing.JTextField txtHostname;
// End of variables declaration

    // Variable in Use
private Socket Wunderbot;
private BufferedReader Wunderbot_In = null;
private BufferedWriter Wunderbot_Out = null;
private DatagramSocket udpWunderbot = null;
private DatagramPacket udpCommandPacket = null;
private Timer j = new Timer();
private boolean SendCommands = false;
private byte byteBuffer[] = new byte[512];
private boolean doStreamWatcher = true;
private boolean deBugging = false;
// For some reason, Socket.isConnected() fails to work. So we work around it with a boolean
private boolean Wunderbot_Connected = false;
```

```
// StreamWatcher watches the input stream of the Wunderbot Socket. It gathers data until
// a semicolon is received, signifying a command.  ProcessCommand is then called to handle it.
private class StreamWatcher extends Thread {
    char inChar[] = new char[512]; int charlength;
    int loc = 0, curChar = 0;
    public void run() {
        try{
            StringBuilder line = new StringBuilder(512);// StringBuilder is a more efficient way to handle s
            while(doStreamWatcher && Wunderbot_Connected){
                charlength = Wunderbot_In.read(inChar,0,512);
                for(loc=0;loc<charlength;loc++){// Go through received data and see what it is
                    line.append(inChar[loc]);
                    if(inChar[loc] == ';'){// Commands are separated by semi-colons
                        ProcessCommand(line);
                        line.setLength(0);line.setLength(512); // Clear out line
                    }
                }
            }
        } catch (SocketException se) {
            se.printStackTrace();
            CloseConnection();
        } catch (Exception e) {
            e.printStackTrace();
            System.exit(-1);
        }
    }
}
private StreamWatcher StreamWatcherT;

// Function that handles commands received.  Currently, there's just "StillAlive;" but more can be added.
private void ProcessCommand(StringBuilder command){
    if(command.indexOf("StillAlive")!=1){ if(deBugging){System.out.print("I'm not dead yet!\n");}new SendKee
}

// run() is executed every 1/10th of a second, and sends out udpCommandPacket, only if its enabled (Button :
private class SendTask extends TimerTask {
    public void run() {
        if(SendCommands && Wunderbot_Connected){
            try {
                udpWunderbot.send(udpCommandPacket);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
}

// SendKeepAlive is a separate thread that sends alive status to MiddleWunderbot when requested.
// Failure to responds makes MiddleWunderbot think WebControl has disconnected, and closes it current sessio
private class SendKeepAlive extends Thread{
    public void run(){
        try{
            if(Wunderbot_Connected){
            Wunderbot_Out.write("StillAlive:Yes;");
            Wunderbot_Out.flush();
            }
        } catch (SocketException se) {// Possible lost of connection, therefore, properly close it.
            se.printStackTrace();
            CloseConnection();
        }catch(Exception e){e.printStackTrace();}
    }
};

// Function that executes a JavaScript function on the webpage to start the video stream at the applet's dis
private void InitUMediaPlayer(String IP,String Port,String Alias)  {
    JSObject.getWindow(this).eval("InitUPlayer(\"" + IP + "\",\"" + Port + "\",\"" + Alias + "\"");
}

// Opens connection to MiddleWunderbot, sets appropriate streams, and turns on the .1 second interval SendTa
private void ConnectToWunderbot(String hostname) {
    try {
        Wunderbot = new Socket(InetAddress.getByName(hostname),4400);
        STATUS_LABEL.setText("Connected");
        Wunderbot_Connected = true;

        // Set Stream variables for socket, so read and wirte can occur
        Wunderbot_In = new BufferedReader(new InputStreamReader(Wunderbot.getInputStream()));
        Wunderbot_Out = new BufferedWriter(new OutputStreamWriter(Wunderbot.getOutputStream()));
        // Enable StreamWatcher for commands
        StreamWatcherT = new StreamWatcher();
        StreamWatcherT.start();

        // Set Datagram Socket and Packet
```

```
            udpWunderbot = new DatagramSocket();//new InetSocketAddress(hostname,4402));
            udpWunderbot.connect(InetAddress.getByName(hostname),4402);
            udpCommandPacket = new DatagramPacket(byteBuffer,512);

            // run video feed
            InitUMediaPlayer(hostname,"4119","wundervision");

            // Set timer to send commands at 100ms (1/10th of a sec)
            j.scheduleAtFixedRate(new SendTask(),0,100);
        } catch (Exception e) {
            e.printStackTrace();
            System.exit(-1);
        }
    }

    // Closes connection to MiddleWunderbot and cleans itself up
    private void CloseConnection(){
        try {
            doStreamWatcher = false;
            SendCommands = false;
            Wunderbot_In.close();
            Wunderbot_Out.close();
            Wunderbot.close();

            udpWunderbot.close();
            udpWunderbot = null;
            StreamWatcherT = null;
            Wunderbot_Connected = false;
            STATUS_LABEL.setText("Closed");
            j.cancel();
            Wunderbot_In = null;
            Wunderbot_Out = null;
            System.gc();
        } catch (Exception e) {
            e.printStackTrace();
            System.exit(-1);
        }
    }
}
```

MiddleWunderbot \ Main.java

```
/*
 * Main.java
 *
 * Created on November 21, 2005, 2:42 PM
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */

package MiddleWunderbot;
import java.net.*;
import java.util.*;
import java.io.*;
/*  MiddleWunderbot
 *      A command-line Java Program that facilites command relays from WebControl to the WunderBot.
 */

public class Main {
    final int WebControl_PORT = 4400;
    final int CommandRelay_WebControl_PORT = 4402;
    final int CommandRelay_Wunderbot_PORT = 3330;
    final int VideoRelay_PORT = 4404;
    final int WaitForWebControl = 0;

    // Ip Addresses - These are important as they say where everything is
    // These must be maintained as the Wunderbot moves around (network address, speaking)
    final String WirelessNetIP = "127.0.0.1";//"192.168.0.3";
    final String PublicNetIP = "192.168.1.101";//"192.168.0.3";
    final String WunderBotIP = "127.0.0.1";//"192.168.0.2";


    //Socket and Stream Variables
    private InetAddress LoopbackAddr;
    private ServerSocket serverSocket;
    private Socket WebControl;
    private BufferedReader WebControl_StreamIn = null;
    private BufferedWriter WebControl_StreamOut = null;
    private DatagramPacket CommandPacket;
```

```
private DatagramSocket udpWebControl, udpWunderbot;


// Thread booleans
private boolean deBugging = true;
private boolean doCommandsRelay = true;
private boolean StillAlive_Sent = false;
private boolean StillAlive_Received = false;
private boolean StillAlive_Wait = false;
private boolean RunMe = true;
private boolean WebControl_Connected = false;
private boolean doStreamWatcher = true;

// On Main class initialization, this is called. It creates a loopback address object to "127.0.0.1"
public void Main() {
    try{
        LoopbackAddr = InetAddress.getLocalHost();
    }catch (Exception e){
        e.printStackTrace();
        System.exit(-1);
    }
}

/* TimerTask that is called every 5 seconds to see if WebControl is still operational.  It can operate three
 * 1st Time: StillAlive Command is sent to WebControl
 * 2nd Time: Checks to see if Response was received.  If not, then wait till next interval to check
 * 3rd Time: If a response is still not received, then connection is assumed to be lost, and the proper var
 */
private class StillAliveTask extends TimerTask{
    public void run() {
        try {
            if(!WebControl_Connected){return;}
            if(StillAlive_Wait && !StillAlive_Received){// If this is the first time since sending StillAliv
                //2nd time call
                WebControl_StreamOut.write(";");// By sending semicolon, no command is sent, yet the connect
                StillAlive_Wait=false;
                return;
            }
            if(StillAlive_Sent){// if you get here, then its been 10 seconds since you last asked
                //3rd time call
                if(!StillAlive_Received){WebControl_Connected = false;}
                StillAlive_Sent = false;
                StillAlive_Received = false;
                return;
            }
            //1st time call
            StillAlive_Sent = true;
            StillAlive_Received = false;
            StillAlive_Wait = true;
            WebControl_StreamOut.write("StillAlive;");
            WebControl_StreamOut.flush();

            System.out.print("\tStill Alive?\n");
        }catch (SocketException se) {
            se.printStackTrace();
            WebControl_Connected = false;
            StillAlive_Sent = false;
            StillAlive_Received = false;
        }catch (Exception e) {
            e.printStackTrace();
            System.exit(-1);
        }
    }
}
private Timer j = new Timer();

// Function that creates a ServerSocket and waits for a connection request
// from the WebControl
private void ListenForWebControl() {
    try {
        // Bind a Listening Socket onto the port
        serverSocket = new ServerSocket(WebControl_PORT,1);
        System.out.print("\tWaiting for WebControl on port " + WebControl_PORT + "\n");

        // Listen for a connection for a specified time period
        serverSocket.setSoTimeout(WaitForWebControl);
        WebControl = serverSocket.accept();

        // Connection is made
        WebControl_Connected = true;
        System.out.print("\tConnected to " + WebControl.getInetAddress().toString() + "\n");
        serverSocket.close();// Close the listener

        // Assign Stream variables
```

```
            WebControl_StreamIn = new BufferedReader(new InputStreamReader(WebControl.getInputStream()));
            WebControl_StreamOut = new BufferedWriter(new OutputStreamWriter(WebControl.getOutputStream()));

        } catch (SocketTimeoutException ste){
            // Caught if serverSocket timeouts.  just get out of it
//            System.out.print("Socket timed out, Ending Execution.");
//            System.exit(2);

        }catch (IOException ex) {
            ex.printStackTrace();
            System.exit(-1);
        }
    }

    // Future Implementation
    // Function that takes user's credentials and authenticates them.  Currently it just says "Yes, User is Auth
    private boolean AuthenticateUser(){return true;}

    // Watches incoming TCP stream from WebControl, waits for commands from WebControl, and passes them off to I
    private class StreamWatcher extends Thread {
        char inChar[] = new char[512]; int charlength;
        int loc = 0, curChar = 0;
        public void run() {
            try{
                StringBuilder line = new StringBuilder(512);// StringBuilder is a more efficient way to handle s
                while(doStreamWatcher && WebControl_Connected){
                    charlength = WebControl_StreamIn.read(inChar,0,512);
                    for(loc=0;loc<charlength;loc++){// Go through received data and see what it is
                        line.append(inChar[loc]);
                        if(inChar[loc] == ';'){// Commands are separated by semi-colons
                            ProcessCommand(line);
                            line.setLength(0);line.setLength(512); // Clear out line
                        }
                    }
                }
            } catch (SocketException se) {
                se.printStackTrace();
                WebControl_Connected = false;
            } catch (Exception e) {
                e.printStackTrace();
                System.exit(-1);
            }
        }
    };
    private StreamWatcher StreamWatcherT;

    // Function that handles commands received.  Currently, there's just "StillAlive:Yes;" but more can be addec
    private void ProcessCommand(StringBuilder command){
        if(command.indexOf("StillAlive:Yes")!=1){StillAlive_Received = true;return;}
    }

    // Function that waits fro a packet from WebControl, then passes them off to Wunderbot.
    public void CommandsRelay(){
        System.out.print("Listening for Udp Packets\n");
        // Loop of main thread
        while(WebControl_Connected) {
            try{
                udpWebControl.receive(CommandPacket);
                if(deBugging){System.out.print("  Relaying " + new String(CommandPacket.getData(),CommandPacket.
                udpWunderbot.send(CommandPacket);
            } catch(SocketTimeoutException ste){} catch(IOException ioe) {}
        }
    }
    // function that initializes variables used in relaying, and starts StillAliveTask and StreamWatcher Thread
    private void InitializeSession() throws Exception{
        // Start StreamWatcher
        doStreamWatcher = true;
        StreamWatcherT = new StreamWatcher();
        StreamWatcherT.start();

        // Start StillAliveTask
        j.scheduleAtFixedRate(new StillAliveTask(),0,5000);


        // initalizes udp sockets and packets
        byte byteBuffer[] = new byte[512];
        udpWebControl = new DatagramSocket(new InetSocketAddress(PublicNetIP,CommandRelay_WebControl_PORT));
        udpWebControl.setSoTimeout(1000);// sets timeout to 1 second.  the receive function will only block for
        udpWunderbot = new DatagramSocket(new InetSocketAddress(WirelessNetIP,CommandRelay_Wunderbot_PORT));// C
        udpWunderbot.connect(InetAddress.getByName(WunderBotIP),CommandRelay_Wunderbot_PORT);// Set destination
        CommandPacket = new DatagramPacket(byteBuffer,512);
    }

    // Function that cleans up the variables used in relaying, as well as ending the StreamWatcher Thread and St
```

```
    private void CleanUpSession() throws Exception{
        // Clean up udp variables
        udpWebControl.close();
        udpWunderbot.close();
        udpWebControl = null;
        udpWunderbot = null;
        CommandPacket = null;

        // End StreamWatcher
        doStreamWatcher = false;
        try{
            WebControl_StreamIn.close();
            WebControl_StreamOut.close();
            WebControl.close();
        }catch(SocketException se){}
        StreamWatcherT = null;
        // End StillAliveTask
        j.cancel();

    }

    // Main execution. First listens for connection from WebCOntrol, Authenticates (Future Implementation), enab
    // StreamWatcher thread and goes to loop with Command Relay function.
    // Execution leaves CommandRelay thread when connection to WebControl is lost.  After that, clean up of vari
    // Once that's done, the whole thing loops around so that another user can connect.
    public void Execute(){
        System.out.print("MiddleWunderBot\n\tFacilitates relay of commands to Wunderbot\n");
        //GetPublicIPInterface();
        try{
            while(RunMe){
                ListenForWebControl();//  Wait for connection from WebControl
                if(!AuthenticateUser()){// If user is unauthenticated
                    // disconnect WebControl, then wait again
                    WebControl.close();
                    System.out.print("WebControl Disconnected\n");
                    continue;
                }

                InitializeSession();
                CommandsRelay();

                CleanUpSession();

                System.out.print("Session Ended\n");
                System.gc();// call the Garbage Collector to free resources
            }
        } catch (Exception e) {
            e.printStackTrace();
            System.exit(-1);
        }

    }

    // All programs have a starting point of execution.  This creates an instance of itself and executes inside.
    // See Java Tutorials for understanding of object-oriented programming practice
    public static void main(String[] args) {
        new Main().Execute();//(int) Integer.valueOf(args[0]));
    }
}
```

Further Reading

The Java Homepage http://java.sun.com
The Java API Documentation 1.5.05 http://java.sun.com/j2se/1.5.0/docs/api/
Java Tutorial http://java.sun.com/docs/books/tutorial/
Unreal Media Website http://www.umedia.net