**Testing  
  
Testing of the Client:**  
The first tests of the client where to make sure the parameter validation implemented worked correctly. This is illustrated in the screen dumps below:

|  |  |
| --- | --- |
| **Input** | **Result** |
| “” |  |
| “ <space><d>” |  |
| “<127.0.0.1>” |  |
| “<form>” |  |

The second part to testing the client was to ensure that the client could connect to a server (both locally and non-locally) and receive data; this is illustrated in the screen dumps and extracts of the log file that the client keeps.

|  |  |
| --- | --- |
| **Input** | **Output** |
| <”127.0.0.1”><”Jamie”><”home”> | **Log Output:** 127.0.0.1 - - [05/Mar/2008:02:13:19 +00:00] “SENT: Jamie home”  127.0.0.1 - - [05/Mar/2008:02:13:19 +00:00] “RECEIVED: OK  “ |
| <”127.0.0.1”><”Jamie”> | **Log Output:** 127.0.0.1 - - [05/Mar/2008:02:16:44 +00:00] “SENT: Jamie”  127.0.0.1 - - [05/Mar/2008:02:16:44 +00:00] “RECEIVED: home  “ |
| <”127.0.0.1”><”aaa”> | **Log Output:** 127.0.0.1 - - [05/Mar/2008:02:21:00 +00:00] “SENT: aaa”  127.0.0.1 - - [05/Mar/2008:02:21:00 +00:00] “RECEIVED: UNKNOWN  “ |
| <”whois.crsnic.net”><”hull.com”> | **Log Output:** whois.crsnic.net - - [05/Mar/2008:02:28:31 +00:00] “SENT: hull.com”  whois.crsnic.net - - [05/Mar/2008:02:28:32 +00:00] “RECEIVED:  Whois Server Version 2.0  Domain names in the .com and .net domains can now be registered  with many different competing registrars. Go to…” |

The third part to testing the client was to test the network error handling:

|  |  |
| --- | --- |
| **Input** | **Output** |
| <”madeupaddress.website.com”><”hi”> | **Log Output:** Local Client - - [05/Mar/2008:02:31:57 +00:00] “ERROR: System.Net.Sockets.SocketException: A connection attempt failed because the connected party did not properly respond after a period of time, or established connection failed because connected host has failed to respond 65.61.199.238:43  at System.Net.Sockets.TcpClient.Connect(String hostname, Int32 port)  at whereis.Whois.ConnectAndReceive(String[] args, String& response) in C:\work\2nd\_year\08241\whereis\whereis\Program.cs:line 70” |

**Testing of the Server:**  
The first part of testing the server was to ensure that more than one connection could be made to the server (telnet was used for all the server tests to make sure there weren’t errors on the client side):

|  |  |
| --- | --- |
| **Input** | **Output** |
| <””> | **Log Output:** 127.0.0.1:1768 - - [05/Mar/2008:02:39:33 +00:00] "NEW CLIENT CONNECTED" OK |

The second part of testing the server involved testing the validation of incoming requests was performed correctly:

|  |  |
| --- | --- |
| **Input** | **Output** |
| <” “> | **Log Output:** 127.0.0.1:1783 - - [05/Mar/2008:02:48:34 +00:00] "NEW CLIENT CONNECTED" OK  127.0.0.1:1783 - - [05/Mar/2008:02:48:36 +00:00] "GET " ERROR: You must supply a name. |
| <”Jamie “> |  |

The third part of testing the server is to test the network error handling, for this my client had to be used in debug mode, so once connected it could be closed before sending any information:

|  |  |
| --- | --- |
| **Input** | **Output** |
| Connect to the server, then disconnect without sending anything | **Log Output:** 127.0.0.1:1850 - - [05/Mar/2008:03:04:42 +00:00] "NEW CLIENT CONNECTED" OK  127.0.0.1:1850 - - [05/Mar/2008:03:04:57 +00:00] "Could not get a response from the client: System.IO.IOException: Unable to read data from the transport connection: An existing connection was forcibly closed by the remote host. ---> System.Net.Sockets.SocketException: An existing connection was forcibly closed by the remote host  at System.Net.Sockets.Socket.Receive(Byte[] buffer, Int32 offset, Int32 size, SocketFlags socketFlags)  at System.Net.Sockets.NetworkStream.Read(Byte[] buffer, Int32 offset, Int32 size)  --- End of inner exception stack trace ---  at System.Net.Sockets.NetworkStream.Read(Byte[] buffer, Int32 offset, Int32 size)  at System.IO.StreamReader.ReadBuffer()  at System.IO.StreamReader.ReadLine()  at whereisserver.Server.ReceiveData(TcpListener& listener, TcpClient& connectedClient, Database& myDatabase) in C:\work\2nd\_year\08241\whereisserver\whereisserver\Server.cs:line 145"  - - [05/Mar/2008:03:04:57 +00:00] "System.InvalidOperationException: The operation is not allowed on non-connected sockets.  at System.Net.Sockets.TcpClient.GetStream()  at whereisserver.Server.StartListen(TcpListener listener) in C:\work\2nd\_year\08241\whereisserver\whereisserver\Server.cs:line 108" |

The fourth and final part of testing the server was to ensure the database would load and save correctly:

|  |  |
| --- | --- |
| **Input** | **Output** |
| Starting the server with no database present |  |
| Starting the server with a database present |  |
| Database present with key = “aaa” value = “bbb” client sets value of key “aaa” to “ccc”, close server, start server again, client requests value of key “aaa” | **Log Output:** 127.0.0.1:1865 - - [05/Mar/2008:03:12:40 +00:00] "NEW CLIENT CONNECTED" OK  127.0.0.1:1865 - - [05/Mar/2008:03:12:42 +00:00] "GET aaa" OK |

**Aspects of Design:**  
I made use of different classes to structure my programs correctly, e.g. a server class and a database class. I designed a logging system to keep track of what the client and server were doing and to keep error reports on network handling. I made use of interfaces to help with my design, i.e. when implementing the corresponding classes I had to adhere to my initial design. I made extensive use of methods to make my code more readable and concise. I also implemented multi-threading in the design of my server to allow multiple connections.  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
 **Appendices**  
**Appendix I: Server Listing**  
 Appendix I.i: Program.cs  
 Appendix I.ii: Server.cs  
 Appendix I.iii: Database.cs  
**Appendix II: Client Listing**  
 Appendix I.i: Program.cs  
 Appendix I.ii: Form1.cs   
**Appendix III: LoggingSystem listing**  
 Appendix I.i: Class1.cs   
  
**Appendix I:**  
  
**Appendix I.i:  
  
Program.cs**using System;

using System.IO;

using System.Net;

using System.Net.Sockets;

using System.Text;

using System.Threading;

namespace whereisserver

{

public class Program

{

static void Main(string[] args)

{

Server myServer = new Server();

}

}

}  
  
**Appendix I.ii:**  
  
**Server.cs**using System;

using System.IO;

using System.Net;

using System.Net.Sockets;

using System.Text;

using System.Threading;

using System.Collections;

using Logging;

namespace whereisserver

{

public interface IServer

{

/// <summary>

/// Calls WaitForAndAcceptConnection and ReceiveDataAndDisconnect using the listener it receives

/// <param name="listener">A listener</param>

/// </summary>

/// <returns>

/// Nothing

/// </returns>

void StartListen(TcpListener listener);

/// <summary>

/// Waits for a connection and accepts it

/// </summary>

/// <param name="listener">Reference to a TcpListener</param>

/// <returns>

/// The TcpClient who is connected

/// </returns>

TcpClient WaitForAndAcceptConnection(ref TcpListener listener);

/// <summary>

/// Receives data from the client

/// <param name="listener">A reference to a listener</param>

/// <param name="client">A reference to the connected client</param>

/// <param name="myDatabase">A reference to the database</param>

/// <returns>

/// An empty string if successful, else an error message

/// </returns>

/// </summary>

//string ReceiveData(ref TcpListener listener, ref TcpClient client,

// ref Database myDatabase);

string ReceiveData(ref TcpListener listener, ref TcpClient client);

/// <summary>

/// Parses the string from the client and decides whether it's a name or name and location

/// <param="inString">String to parse</param>

/// <param name="outName">The name</param>

/// <param name="outLocation">The location</param>

/// </summary>

/// <returns>

/// An int, 1 = name, 2 = name and location, -1 = an error occurred

/// </returns>

int ParseString(string inString, out string outName, out string outLocation);

/// <summary>

/// Validates the string

/// </summary>

/// <param="inString">String to validate</param>

/// <returns>

/// An empty string if successful, else an error message

/// </returns>

string ValidateString(string inString);

}

public class Server : IServer

{

private Database myDatabase = new Database();

private LoggingSystem log = new LoggingSystem();

public Server()

{

TcpListener listener = new TcpListener(43);

string reply = "";

// Load the database first

reply = myDatabase.LoadDatabase();

if (reply != "") // Something went wrong

{

Console.WriteLine(reply);

}

listener.Start();

StartListen(listener);

}

public void StartListen(TcpListener listener)

{

TcpClient client = null;

bool running = true;

string reply = "";

Console.WriteLine("New thread open for someone to connect:");

while (running)

{

try

{

client = WaitForAndAcceptConnection(ref listener);

client = listener.AcceptTcpClient();

log.WriteToLog("NEW CLIENT CONNECTED", ".//log.txt",

"OK", client.Client.RemoteEndPoint.ToString());

Console.WriteLine(client.Client.RemoteEndPoint.ToString() +

" Connected");

// Now someone has connected create a new thread allowing someone else to connect

ThreadStart th = delegate { StartListen(listener); }; // delegate allows parameter

// passing

new Thread(th).Start();

//reply = ReceiveData(ref listener, ref client, ref myDatabase);

reply = ReceiveData(ref listener, ref client);

StreamWriter sw = new StreamWriter(client.GetStream());

sw.WriteLine(reply); // Send the reply back to the connected client

sw.Flush();

client.Close();

}

catch (System.Exception e)

{

Console.WriteLine("ERROR: Check the log for details.");

log.WriteToLog(e.ToString(), ".//log.txt", "", "");

}

}

}

public TcpClient WaitForAndAcceptConnection(ref TcpListener listener)

{

TcpClient connectedClient = null; // So getstream becomes available

bool connected = false;

while (!connected)

{

connected = listener.Pending();

}

return connectedClient;

}

//public string ReceiveData(ref TcpListener listener, ref TcpClient connectedClient,

// ref Database myDatabase)

public string ReceiveData(ref TcpListener listener, ref TcpClient connectedClient)

{

string error = "";

string name = "";

string location = "";

int intError = 0;

StreamReader sr = new StreamReader(connectedClient.GetStream());

string response = "";

bool boolResponse = false;

try

{

response = sr.ReadLine(); // Get the input from the client

Console.WriteLine("RECEIVED: " + connectedClient.Client.RemoteEndPoint.ToString()

+ " " + "'" + response + "'");

error = ValidateString(response); // Make sure the input is in the correct format

if (error != "") // Something went wrong

{

Console.WriteLine("SENT: " + connectedClient.Client.RemoteEndPoint.ToString() +

" " + "'" + error + "'");

return error;

}

intError = ParseString(response, out name, out location); // Check whether name/name+location

if (intError == -1) // Something went wrong

{

error = "ERROR: Use <name> or <name><space><location>";

Console.WriteLine("SENT: " + connectedClient.Client.RemoteEndPoint.ToString() +

" " + "'" + error + "'");

return error;

}

if (location == "") // No location specified, only name specified

{

boolResponse = myDatabase.DoesPersonExist(name); // Makes logging easier

if (boolResponse)

{

error = "OK";

log.WriteToLog("GET " + name, ".//log.txt",

error, connectedClient.Client.RemoteEndPoint.ToString());

error = myDatabase.GetLocation(name); // For returning the location or "Error:..."

Console.WriteLine("SENT: " + connectedClient.Client.RemoteEndPoint.ToString() +

" " + "'" + error + "'");

return error;

}

else // Person doesn't exist

{

error = "UNKNOWN";

log.WriteToLog("GET " + name, ".//log.txt",

error, connectedClient.Client.RemoteEndPoint.ToString());

Console.WriteLine("SENT: " + connectedClient.Client.RemoteEndPoint.ToString() +

" " + "'" + error + "'");

return error;

}

}

else // name and location both specified

{

error = myDatabase.ValidateName(name);

if (error != "") // Something went wrong

{

Console.WriteLine("SENT: " + connectedClient.Client.RemoteEndPoint.ToString() +

" " + "'" + error + "'"); ;

return error;

}

error = myDatabase.ValidateLocation(location);

if (error != "") // Something went wrong

{

Console.WriteLine("SENT: " + connectedClient.Client.RemoteEndPoint.ToString() +

" " + "'" + error + "'");

return error;

}

boolResponse = myDatabase.DoesPersonExist(name);

if (boolResponse) // Person exists so update their location

{

error = myDatabase.SetLocation(name, location);

error = "OK";

log.WriteToLog("SET " + name + " " + location, ".//log.txt",

error, connectedClient.Client.RemoteEndPoint.ToString());

Console.WriteLine("SENT: " + connectedClient.Client.RemoteEndPoint.ToString() +

" " + "'" + error + "'");

}

else // Person doesn't exist so add them and their location

{

myDatabase.AddPerson(name, location);

error = "OK";

log.WriteToLog("SET " + name + " " + location, ".//log.txt",

error, connectedClient.Client.RemoteEndPoint.ToString());

Console.WriteLine("SENT: " + connectedClient.Client.RemoteEndPoint.ToString() +

" " + "'" + error + "'");

}

}

return error;

}

catch (System.Exception e)

{

error = "Could not get a response from the client, check the logs for more details";

log.WriteToLog("Could not get a response from the client: " + e.ToString(), ".//log.txt",

"", connectedClient.Client.RemoteEndPoint.ToString());

Console.WriteLine(error);

return error;

}

}

public int ParseString(string inString, out string outName, out string outLocation)

{

string name = "";

string location = "";

int lengthOfInString = inString.Length;

int endPosition = 0;

int counter = -1; // for keeping track of position within the string

foreach (char c in inString)

{

counter++;

if (c == ' ') // If there's a space then a name and a location has been given

{

endPosition = lengthOfInString - counter; // Gives the endpoint for the name

name = inString.Remove(counter); // Remove the location part

int tempLength = 0;

tempLength = lengthOfInString - name.Length;

// Below counter+1 is there because otherwise the space would be counted

// Below tempLength-1 is there because otherwise outOfBounds (see line above)

location = inString.Substring((counter + 1), (tempLength - 1)); // Removes the name part

outName = name;

outLocation = location;

return 2; // Both name and location

}

}

// If here then only a name has been specified

outName = inString;

outLocation = location;

return 1; // Only a name

}

public string ValidateString(string inString)

{

// The string shouldn't have more than 1 space because max is

// <name><space><location>

int counter = 0; // Keep track of spaces

string reply = "";

foreach (char c in inString)

{

if (c == ' ')

{

counter++;

}

}

if (counter > 1)

{

reply = "ERROR: <name> or <name><space><location>.";

return reply;

}

return reply; // If here everything went OK

}

}

}

**Appendix I.iii:**  
  
**Database.cs**using System;

using System.IO;

using System.Net;

using System.Text;

using System.Runtime.Serialization.Formatters.Binary;

using System.Collections;

namespace whereisserver

{

public interface IDatabase

{

/// <summary>

/// Adds a person and corresponding location to the database calls ValidateName and

/// ValidateLocation, calls SaveDatabase

/// </summary>

/// <param="inName">The person's name</param>

/// <param="inLocation">The person's location</param>

/// <returns>

/// An empty string if successful, else an error message

/// </returns>

string AddPerson(string inName, string inLocation);

/// <summary>

/// Validates the person's name

/// <param="inName">The person's name</param>

/// </summary>

/// <returns>

/// An empty string if successful, else an error message

/// </returns>

string ValidateName(string inName);

/// <summary>

/// Validates the person's location

/// <param="inLocation">The person's location</param>

/// </summary>

/// <returns>

/// An empty string if successful, else an error message

/// </returns>

string ValidateLocation(string inLocation);

/// <summary>

/// Get the person's location

/// <param="inName">The person's name to lookup to get their corresponding location</param>

/// </summary>

/// <returns>

/// The person's location if successful else an error message

/// </returns>

string GetLocation(string inName);

/// <summary>

/// Set the person's location, calls SaveDatabase

/// <param="inName">The person's name</param>

/// <param="inLocation">The person's location</param>

/// </summary>

/// <returns>

/// An empty string if successful, else an error message

/// </returns>

string SetLocation(string inName, string inLocation);

/// <summary>

/// Check to see whether the person already exists in the database

/// <param="inName">Name to check</param>

/// </summary>

/// <returns>

/// True if the person exists, else false

/// </returns>

bool DoesPersonExist(string inName);

/// <summary>

/// Loads the database from a file into a hashtable

/// </summary>

/// <returns>

/// An empty string if everything went ok, else an error message

/// </returns>

string LoadDatabase();

/// <summary>

/// Saves the database from a hashtable into a file

/// </summary>

/// <returns>

/// An empty string if everything went ok, else an error message

/// </returns>

string SaveDatabase();

}

public class Database : IDatabase

{

private Hashtable myDatabase = new Hashtable();

public string AddPerson(string inName, string inLocation)

{

string reply;

reply = ValidateName(inName);

if (reply != "") // Something went wrong

{

return reply; // No point continuing

}

reply = ValidateLocation(inLocation);

if (reply != "") // Something went wrong

{

return reply;

}

// Convert the name to uppercase to avoid further validation routines

myDatabase.Add(inName.ToUpper(), inLocation);

// Update the database

reply = this.SaveDatabase();

return reply;

}

public string ValidateName(string inName)

{

string reply = "";

if (inName.StartsWith(" "))

{

reply = "ERROR: The name cannot start with a white space.";

return reply;

}

if (inName == "")

{

reply = "ERROR: You must supply a name.";

return reply;

}

return reply; // Everything went OK if here

}

public string ValidateLocation(string inLocation)

{

string reply = "";

if(inLocation.StartsWith(" "))

{

reply = "ERROR: The location cannot start with a white space.";

return reply;

}

if(inLocation == "")

{

reply = "ERROR: You must supply a location.";

return reply;

}

return reply; // Everything went OK if here

}

public string GetLocation(string inName)

{

string reply = "";

bool keyPresent = false;

keyPresent = myDatabase.ContainsKey(inName.ToUpper());

if (!keyPresent) // The person isn't in the database

{

reply = "ERROR: no entries found.";

return reply;

}

inName = inName.ToUpper();

reply = myDatabase[inName].ToString(); // Get the location corresponding to the name supplied

return reply; // If here everything went OK

}

public string SetLocation(string inName, string inLocation)

{

string reply = "";

myDatabase[inName] = inLocation; // Set/Update the location

// Update the database

reply = this.SaveDatabase();

return reply;

}

public bool DoesPersonExist(string inName)

{

bool response = false;

response = myDatabase.ContainsKey(inName.ToUpper());

return response;

}

public string LoadDatabase()

{

string reply = "";

FileInfo fileToCheck = new FileInfo(".//database.txt");

// Does the database exist to load?

if (!fileToCheck.Exists)

{

reply = "ERROR: The database could not be loaded because it doesn't exist," +

" if this is the first time running the server this error can be ignored.";

}

else

{

try

{

FileStream file = new FileStream(".//database.txt", FileMode.Open, FileAccess.Read);

BinaryFormatter bf = new BinaryFormatter();

myDatabase = (Hashtable)bf.Deserialize(file);

file.Close();

}

catch (System.Exception e)

{

reply = "ERROR: Whilst loading the database this error occurred: " + e.ToString();

}

}

return reply;

}

public string SaveDatabase()

{

string reply = "";

try

{

FileStream file = new FileStream(".//database.txt", FileMode.OpenOrCreate,

FileAccess.Write);

BinaryFormatter bf = new BinaryFormatter();

bf.Serialize(file, myDatabase);

file.Close();

}

catch (System.Exception e)

{

reply = "ERROR: Whilst saving the database this error occurred: " + e.ToString();

}

return reply;

}

}

}

**Appendix II:  
  
Appendix II.i:**using System;

using System.Net.Sockets;

using System.IO;

using System.Windows.Forms;

using System.Threading;

using Logging;

namespace whereis

{

public class Whois

{

// Parameters passed in will populate these variables

private string inName;

private string inLocation;

private string inServer;

public string WriteToLog(string message, string status, string server) // Write to the log file

{

LoggingSystem log = new LoggingSystem();

string reply = log.WriteToLog(message, ".\\log.txt", status, server);

return reply;

}

public void PopulateLocation(string[] inArgs) // Populate the location parameter

{

inLocation = inArgs[2];

}

public void PopulateServer(string[] inArgs) // Populate the server parameter

{

inServer = inArgs[0];

}

public void PopulateName(string[] inArgs) // Populate the name parameter

{

inName = inArgs[1];

}

public string ConnectAndReceive(string[] args, out string response) // Connect to the server

// and receive data from the server

{

string errorReply = "";

response = "";

// Console validation - using ui

if (args.Length < 2)

{

errorReply = "You didn't specify a name.";  
 return errorReply;

}

// Form validation - using gui

if (args[0] == "")

{

errorReply = "You didn't specify a server.";

return errorReply;

}

if (args[1] == "")

{

errorReply = "You didn't specify a name.";

return errorReply;

}

// If here then passed the validation tests

TcpClient client = new TcpClient(); // Initialise a new client

client.SendTimeout = 1000; // Set the timeout to 1 second (100ms)

try

{

client.Connect(args[0], 43); // [0] == the server

StreamWriter sw = new StreamWriter(client.GetStream());

StreamReader sr = new StreamReader(client.GetStream());

// Find out if they specified a location and act accordingly

if (args.Length > 2) // So don't go out of bounds when using the console ui

{

if (args[2] != "") // Then they specified a name and location

{

sw.WriteLine(args[1] = args[1] + " " + args[2]); // [1] == person's name [2] == location

WriteToLog("SENT: " + args[1], "", args[0]); // Keep a log of client's actions

}

else // Location specified was ""

{

sw.WriteLine(args[1]); // [1] == person's name

WriteToLog("SENT: " + args[1], "", args[0]);

}

}

else // Specified a name and not a location

{

sw.WriteLine(args[1]); // [1] == person's name

WriteToLog("SENT: " + args[1], "", args[0]);

}

sw.Flush();

response = sr.ReadToEnd();

WriteToLog("RECEIVED: " + response, "", args[0]);

}

catch (Exception e)

{

errorReply = "An error occurred whilst connecting to the server, check the log for further details.";

WriteToLog("ERROR: " + e.ToString(), "", "Local Client");

return errorReply;

}

//client.Close(); // disconnect from server - don't do this, this is the servers job

return errorReply; // Everything went ok if here

}

static void Main(string[] args)

{

Whois a = new Whois(); // Setup a new instance

int argumentLength;

int minimumArgumentsAllowed = 1;

int maximumArgumentsAllowed = 3;

string response;

string reply;

// Validate console arguments been passed in

argumentLength = args.GetLength(0);

if ((argumentLength < minimumArgumentsAllowed) ||

(argumentLength > maximumArgumentsAllowed)) // Too few or too many arguments

{

Console.WriteLine("You have specified an incorrect number of arguments.");

Console.WriteLine("Try whereis.exe <server ip/name><space><name>");

Console.WriteLine("or");

Console.WriteLine("whereis.exe <server ip/name><space><name><space><location>");

Console.ReadLine(); // Don't exit until user has hit Enter

}

else // Arguments number correct so populate the corresponding variables

{

if (argumentLength == 2) // name and server specified

{

a.PopulateName(args);

a.PopulateServer(args);

}

if (argumentLength == 3) // name, server, and location specified

{

a.PopulateName(args);

a.PopulateServer(args);

a.PopulateLocation(args);

}

if (args[0] == "form") // Chosen to use the gui

{

Application.Run(new Form1());

}

else // using the console as the ui

{

reply = a.ConnectAndReceive(args, out response);

if (reply != "") // Something went wrong

{

Console.WriteLine(reply);

}

else // Everything went ok

{

Console.WriteLine(response);

}

Console.WriteLine("Press 'Enter' to quit.");

Console.ReadLine(); // Don't quit unless the user has pressed a key

}

}

}

}

}  
  
**Appendix II.ii:**using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Text;

using System.Windows.Forms;

using System.Net.Sockets;

using System.IO;

namespace whereis

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

private void Form1\_Load(object sender, EventArgs e)

{

this.Activate(); // Bring the form to the front

}

private void buttonSend\_Click(object sender, EventArgs e)

{

Whois a = new Whois();

string[] args = new string[3];

string reply;

string response;

args[0] = textBoxServer.Text;

args[1] = textBoxInputName.Text;

args[2] = textBoxInputLocation.Text;

reply = a.ConnectAndReceive(args, out response); // Send the arguments to the server

if (reply != "") // Something went wrong

{

MessageBox.Show(reply, "Whereis", MessageBoxButtons.OK, MessageBoxIcon.Exclamation);

}

else

{

richTextBoxResponse.Text = response; // Response from the server

}

}

}

} **Appendix III:  
  
Appendix III.i:**using System;

using System.Collections.Generic;

using System.Text;

using System.IO;

namespace Logging

{

public interface ILoggingSystem

{

/// <Summary>

/// Writes to a log file

/// <param name="message">The message to write to the log file</param>

/// <param name="file">The file to write to</param>

/// <param name="status">The status, e.g. SET</param>

/// <param name="server">The server that is sending or receiving</param>

/// </Summary>

/// <Returns>

/// An empty string if successful, else an error message

/// </Returns>

string WriteToLog(string message, string file, string status, string server);

}

public class LoggingSystem : ILoggingSystem

{

public string WriteToLog(string message, string fileName, string status, string server)

{

string reply = "";

try

{

FileInfo file = new FileInfo(fileName);

DateTime date = DateTime.Now; // Get the current date and time

string format = date.ToString("dd/MMM/yyyy:hh:mm:ss K"); // format the current date/time

string formattedMessage = server + " - - [" + format + "]" + " \"" + message + "\""

+ " " + status; // combine the date/time format with the actual message and format

// that

if (!file.Exists) // If the file doesn't exist, create it, else append

{

using (StreamWriter sw = file.CreateText())

{

sw.WriteLine(formattedMessage);

}

}

else

{

using (StreamWriter sw = file.AppendText())

{

sw.WriteLine(formattedMessage);

}

}

}

catch (System.Exception e)

{

//reply = Convert.ToString(e); // Won't be very useful to the user, and can't write the

// error to a log file because this occurred when trying to write to a file.

reply = "There was a problem whilst trying to write to the log file\nplease make sure " +

"the file: " + fileName + " is not already open";

return reply;

}

return reply; // If here then everything went OK

}

}

}