Development of a Wireless Position Tracking System

An Undergraduate Thesis
Presented to
The Department of Electrical and Computer Engineering
The Faculty of Engineering
The University of Manitoba

In Partial Fulfilment
Of the Requirements for the Degree in
Bachelor of Science in Computer Engineering

Submitted By:
Kyle Geske

Date:
March 9, 2000

Abstract

This thesis developed a low cost wireless tracking system utilizing both GPS and Internet technology. Unlike currently available systems, it provides world-wide accessibility to the real-time location data collected by the tracking devices while still providing full freedom of motion for the object being tracked. More specifically, the World Wide Web was used as a distribution medium for the location data and a cellular modem was used to give the tracking device the ability to report its location in a wireless manner. The client and server computer programs designed for this system were written as cross-platform applications in order to allow them to run under both the Windows 98 and Linux operating systems. Through extensive testing, the system was shown to provide position accuracy within 100 meters horizontally and 156 meters vertically, 95% of the time.

Acknowledgements

I would like to take the time to thank the following people: My family and my girlfriend Shannon for providing me with moral support and putting up with my crankiness after pulling thesis oriented all-nighters. My thesis advisor Bob McLeod for accepting me as an advisee, obtaining a GPS receiver for me and not mocking my chosen topic. And finally, my friends, for without you guys I would probably have dropped out of engineering years ago. So thanks, Andrew, Bruce, Bryan, Chris, Harold, Mike and Steve.

Table Of Contents

Table of Figures

Table of Tables

List of Abbreviations and Acronyms

ASCII	American Standard Code for Information Interchange
CMOS	Complementary Metal Oxide Semiconductor
DGPS	Differential Global Positioning System
DOD	Department of Defense
GPS	Global Positioning System
GUI	Graphical User Interface
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IP	Internet Protocol
LAN	Local Area Network
OEM	Original Equipment Manufacturer
PPS	Precise Positioning Service
RAM	Random Access Memory
RFC	Request For Comment
SPS	Standard Positioning Service
SV	Space Vehicle
TAIP	Trimble ASCII Interface Protocol
TCP	Transmission Control Protocol
URL	Universal Resource Locator
UTC	Co-ordinated Universal Time

Chapter 1 - Introduction

1.1 Purpose

The objective of this thesis is to investigate the feasibility of a low cost wireless position tracking system using both GPS, (Global Positioning System), and Internet technology. This thesis introduces the architecture for a low cost wireless position tracking system along with an evaluation of the technology.

1.2 Problem

In the past, the real-time tracking of vehicles, items, and personnel was only a reality for large government agencies in spy novels. Recent developments in technology are now bringing this type of tracking capability to the general public. Real-time tracking systems could be used to monitor the locations of police squad cars and ambulances for faster emergency response time. The location of public transportation vehicles, such as buses could be displayed at bus stop terminals for interested patrons. Overly protective or prying parents could track the location of the family car unbeknownst to their teenager in the driver's seat.

When considering systems of this nature, one has to consider both the location data being used for tracking purposes, as well as the method with which the location data will be made accessible. Currently, most low cost tracking systems do not allow for real-time access to location data. Instead, the onboard tracking device stores the data for later retrieval and analysis. For a tracking system to be truly useful, it should provide a highly accessible real-time data monitoring mechanism. To allow for this mechanism, and still provide full freedom of movement, the system would have to report its location data via a "wireless" communication link.

The system developed in this thesis uses a GPS receiver as a source of location data and a wireless connection to the Internet, (more specifically the World Wide Web), as a distribution medium for this data.

1.3 Scope

This thesis is comprised of 6 chapters. The first chapter presents the objective of the thesis, details what is lacking in currently available position tracking systems and provides some motivation for the thesis. The second chapter provides background and technical information on GPS, the Internet and position tracking systems. The third chapter details the requirements for the system being developed and outlines the basic architecture for the tracking system. In chapter four a detailed description of the tracking system is given. Chapter five provides methods and the results for the experimentation done with the system. And finally, chapter six provides a conclusion to the thesis along with recommendations for future work.

Chapter 2 - Background Information

This thesis relies very heavily on the use of both GPS and Internet technology. This chapter will begin with an overview of the Global Positioning System run by the US Department of Defence, DoD. A brief explanation of the Internet related technologies used in this thesis will also be provided.

2.1 GPS Background Information

For as long as man has been on this earth, he has been searching for accurate navigation and positioning methods; from the first man travelling on foot and using landmarks to determine his position, to a SR-71 Blackbird military aircraft gathering target location data while travelling at supersonic speeds. The United States Department of Defence runs a freely available and highly accurate positioning system called GPS. This section will provide an overview of GPS and the longitude/latitude co-ordinate system used in this thesis.

2.1.1 Longitude/Latitude Co-Ordinate System

The location of any point on a planar surface can be fully described by both a horizontal and a vertical co-ordinate. Had the earth been truly flat and rectangular as it had been perceived in the past, a simple equally spaced grid system could be used to describe any location on earth. However, the earth is neither flat nor rectangular. It is not even a perfect sphere. The fact of the matter is that the earth is an oblate ellipsoid, a slightly egg shaped sphere [ROS99]. A grid-like system of latitudes and longitudes was devised to describe precise locations on earth. Both latitude and longitude are measured in terms of degrees, (°). Figure 2.1 shows a map of the earth including basic longitude and latitude information.

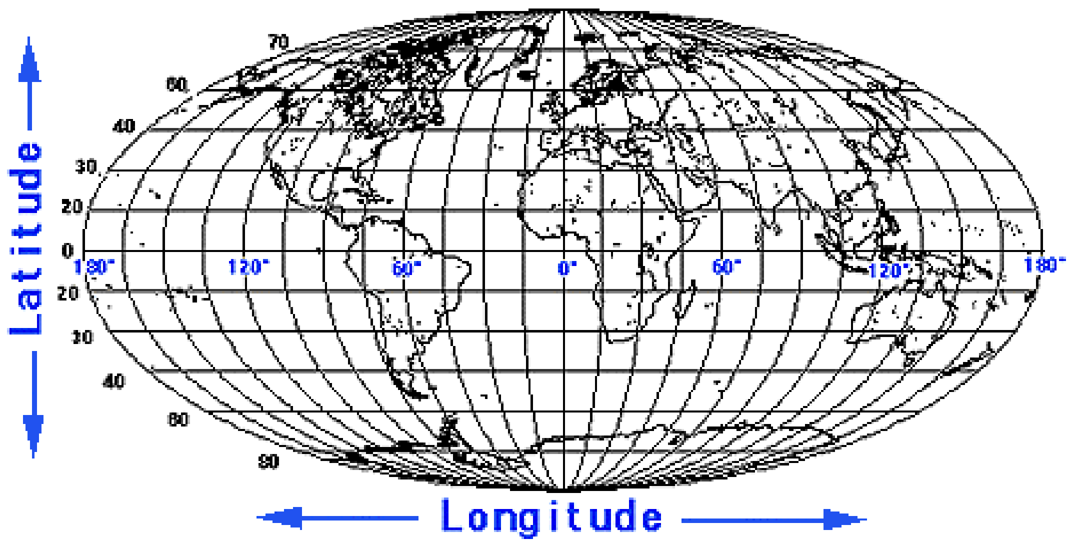


Figure 2.1 - The earth divided into Longitude and Latitude

Latitudes can be described as lines that circle the earth parallel to the equator, the line that equally divides the earth into a Northern and Southern hemisphere. Degrees of latitude are numbered from 0° to 90° North and South. Zero degrees in latitude is the location of the equator. Ninety degrees North and South is the location of the North and South poles respectively. Each degree of latitude is approximately equal to 111 km, this variation is cause by the ellipsoid shape of the earth.

Longitudes can be best described as lines that circle the earth and converge at both the North and South poles. Degrees of longitude are numbered from 0° to 180° East and West. Zero degrees in longitude is located at Greenwich, England. The East and West degrees meet at 180°, and define the International Date Line. At the equator, a degree of longitude is approximately equal to 111 kms. As the latitude increases, longitudinal degrees decrease in distance until they converge at one of the earth's poles.

For increased accuracy, degrees can be sub-divided into minutes, ('), and seconds, ("). Each degree contains 60 minutes, and each minute contains 60 seconds. Seconds can be further divided into tenths, hundredths, and thousands [ROS99]. A longitude and latitude is said to be in decimal degree form if it has not been broken up into minutes and seconds. In decimal degree form, the sign of the number denotes the co-ordinate's direction, (positive for North and East, negative for South and West). The level of accuracy dealt with in this thesis is in the range of 100-200 meters, and therefore, longitude and latitude will be described in degrees, minutes and seconds. An example co-ordinate in this system is the location of Winnipeg, Manitoba, Canada at 49° 54' North and 97° 14' West. Seconds were not used in this location, as they would have overly constrained the location of the city.

To obtain the metric spacing of longitude and latitude at different points on the earth's surface and remain within the accuracy required by this thesis, the WGS84 datum was used as well as the great circle distance formula. The WGS84 datum provides an accurate measurement of the circumference of the earth through its poles.

2.1.2 Global Positioning System

GPS is a satellite based radio navigation system developed and run by the US Department of Defence. The system consists of 24 satellites, or Space Vehicles, (SVs), 5 ground monitoring stations around the world, and a master control station in Colorado. The signals broadcast by the SVs are received and used to calculate the longitude, latitude and height of the receiver. The information provided by the SVs can also be used to provide timing with near atomic clock accuracy. There are two levels of service available from the GPS, the Standard Positioning Service, (SPS), and the Precise Positioning Service, (PPS).

SPS is the service available to the general public at no cost, but at a slight reduction of accuracy. Since GPS is a broadcast system, the US government did not want to give everyone in the world access to a highly accurate positioning system for reasons of national security. Instead of fully denying access to the service, they deliberately introduced small timing and position errors into the public band GPS signals. This is known as Selective Availability, (SA). With SA, SPS provides a predictable positioning accuracy of 100 meters horizontally, 156 meters vertically with time transfer accuracy to UTC within 340 nanoseconds 95% of the time [USN99].

PPS is the highly accurate service available to the US military and other US sanctioned organizations. PPS signals are protected from public view using very sophisticated encryption techniques. As PPS is not restricted by SA it can provide a predictable positioning accuracy of 22 meters horizontally, 27.7 meters vertically with time transfer accuracy to UTC within 100 nanoseconds 95% of the time.

Each of the 24 SVs orbit the earth with a 12 hour period, at approximately 20,200km, at an inclination angle of 55 degrees. This allows for 5 to 8 SVs to be visible from any point on earth at all time. Figure 2.2 provides a basic view of the GPS constellation.

The SVs maintain two way communication with the ground stations, which provide them with precise location and clock correction information. Incorporating this correction information with the data from their on board atomic clocks, the SVs broadcast the current time, (in UTC) as well as their exact location. These broadcasts are synchronized in such a manner that all the SVs transmit the information simultaneously.

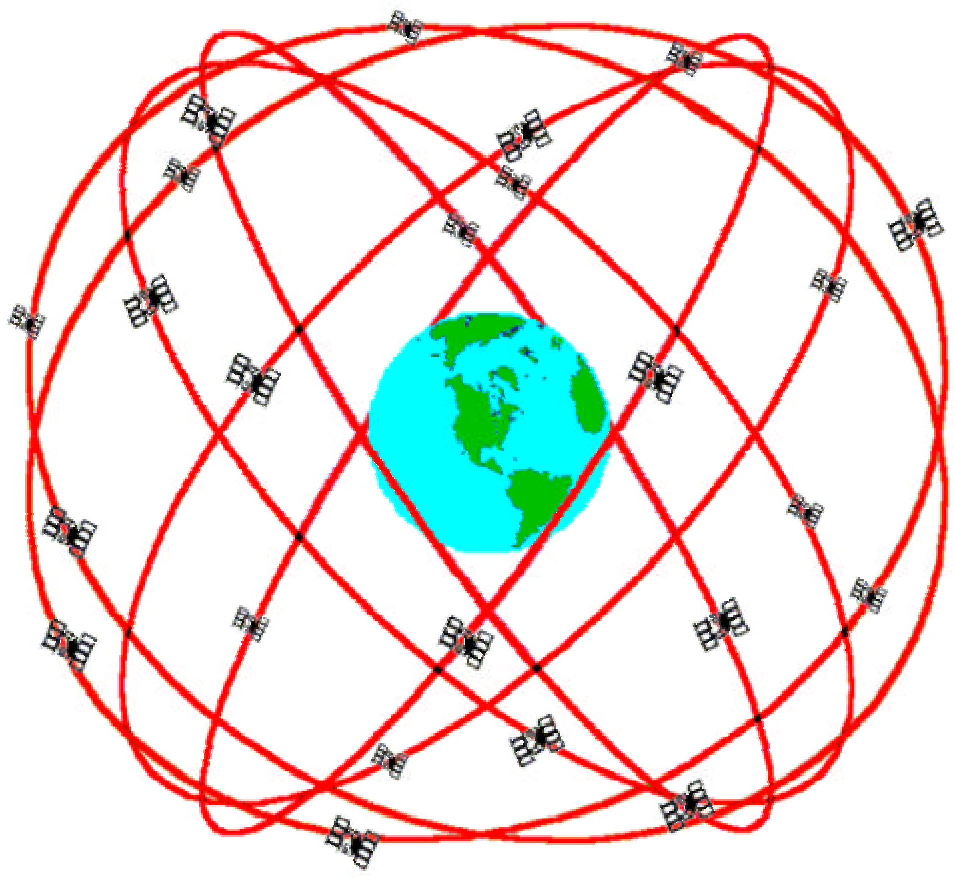


Figure 2.2 - The 24 SV GPS Constellation

GPS receivers are passive devices that use the data sent by the SVs to calculate their location. Contrary to public belief, the GPS receivers do not transmit any information back to the SVs. The GPS receivers use the location and timing information broadcast by a SV to determine their distance away from the satellite. They can determine this distance because they know the time at which the broadcast message was sent, the time at which they received the message and the speed of the radio waves used for the transmission. The method used by the receivers to calculate their location is that of triangulation.

To triangulate the position of a GPS receiver in three dimensions, three reference SVs are needed. Once the GPS receiver has calculated its distance from a SV it knows that it lies somewhere on the surface of a sphere with the SV in its centre. With three reference distances, the receiver knows that it is located at the point at which the three spheres intersect.

This scenario would work perfectly if the GPS receiver could determine its precise distance from each SV. However, this calculation is very dependant on timing, and low cost GPS receivers do not contain atomic clocks, (if they did, they would no longer be very low cost). As we are dealing with signals which are travelling at close to the speed of light (3x10^8 m/s), a small discrepancy in timing will result in a very large error in distance. How does the receiver then compensate for this lack in timing accuracy? It will lock onto the signal of one more reference SV. If three perfect measurements are able to locate a point in 3-dimensional space, then four imperfect measurements can do approximately the same thing. [TRI97] As we know, if the three spheres were accurately calculated they would all meet at a single point. With inaccurate timing, a forth sphere measurement would not intersect with the first three. The GPS receiver acknowledges that this discrepancy is due to errors in its internal clock and will look for a clock correction which would allow all four spheres to intersect at a single point. Not only does this provide an accurate fix on the location of the receiver, it also gives the existing internal clock near atomic clock accuracy.

2.2 Internet Technology Background Information

Communication and the distribution of information are two of the most important aspects of daily life. With the advent of the Internet, the ability to make information instantly accessible to millions world-wide has become a reality. This thesis uses the Internet as a distribution medium for the location data which is gathered by the GPS receiver. The data is made available via the World Wide Web, and therefore, this section will provide a brief overview of the HTTP Protocol used to communicate between Web browsers and Web clients. As the location data is sent to the web server via a TCP/IP connection this protocol suite will also be examined.

2.2.1 The TCP/IP Protocols

The Transmission Control Protocol, (TCP), and the Internet Protocol, (IP), are the main data transfer protocols on the Internet. The combination of TCP and IP provides a reliable and connection oriented data path for networked applications. The general hierarchy of client/server communication using a TCP/IP network is depicted in figure 2.3.

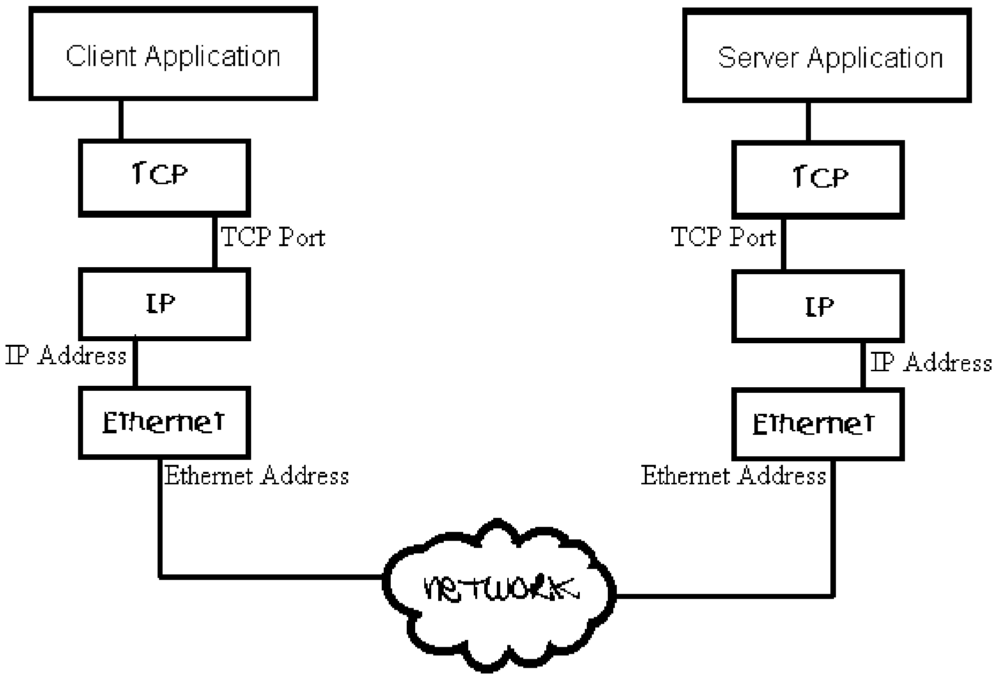


Figure 2.3 - TCP/IP communication

The flow of data from a client application to a server application across a TCP/IP connection is as follows: The client application, which has been assigned a TCP port number by the programmer, sends its data to the TCP module. The application specifies the destination TCP port of the server application, as well as a destination IP Address. To simplify matters it can be assumed that all networked computers have a unique IP address and these addresses correspond to unique Ethernet addresses. The Ethernet is the physical medium through which the TCP/IP connection is transmitted. The TCP module then passes the client's data down to the IP module. Both the TCP and IP modules add extra header information to the data. These headers hold information such as the destination IP address and the destination TCP port number. Again, to simplify matters we must assume that the data magically arrives at its destination and is passed by the

Ethernet module to the IP level. The destination IP module strips off the IP headers and passes the data to the appropriate TCP port. The server application has also been bound to a TCP port number and will next receive the client's data from the TCP module.

Although this is a very simplistic view of a TCP/IP connection, it provides enough detail to explain how an application layer protocol, like HTTP, can transfer data from source to destination across a TCP/IP network.

2.2.2 The HTTP Protocol

The HyperText Transfer Protocol (HTTP), is an application layer protocol used to transfer information on the World Wide Web using a TCP/IP network. A subset of the HTTP protocol was implemented in this thesis for the data distribution system.

HTTP is a request and response oriented protocol. In terms of its use on the World Wide Web, a Web browser sends an HTTP formatted request to a Web server requesting web page content. The Web server, in turn, returns an HTTP formatted response, which includes a success or error code and possibly, the requested web page content. Much like the description of TCP/IP, this is a necessarily simplistic view of the HTTP protocol. A sample request and response is depicted in table 2.1. The line numbers were added to facilitate further explanation.

Web Browser HTTP Request
1) GET /index.html HTTP/1.1
2) Accept: */*
3) Accept-Language: en-ca
4) Accept-Encoding: gzip,deflate
5) User-Agent:Mozilla/4.0 (compatible;MSIE 5.01;Windows 98)
6) Host: 207.161.231.161
7) Connection: Keep-Alive
Web Server HTTP Response
1) HTTP/1.1 200 OK
2) Date: Mon, 21 Feb 2000 05:38:37 GMT
3) Server: Apache/1.3.9 (Unix)
4) Keep-Alive: timeout=15, max=100
5) Connection: Keep-Alive
6) Transfer-Encoding: chunked
7) Content-Type: text/html

Table 2.1 - HTTP Request and Response

In order to create a Web server that can effectively communicate with standard Web browsers, a server must meet minimal compliance with the HTTP RFC, (a standards document written by the Internet Engineering Task Force). For minimum compliance, the bolded sections of table 2.1 must be considered. The requested content would follow line 7 in the server's response.

The first line of the HTTP request indicates the browser's request to "GET" the file, "index.html", using the HTTP protocol version 1.1. With this information alone, the server can now make a reply to the browser.

The first line of the response indicates that the server is also using version 1.1 of the protocol. The next line of importance is line 5, the connection status. Keep-Alive signifies that the TCP/IP connection, through which the request and reply was sent, will remain open for further request. This is subject to the timeouts in line 4. If the server wishes to terminate the TCP/IP connection after its response it can return a connection type of *closed*. The final, and perhaps the most important line, is line 7. In this line the server specifies the type of content information it is returning. In our example, the server specifies a content-type of *text/html.*

All of the TCP/IP and HTTP interactions are performed unseen to the user, who merely typed a webpage URL into a web browser and waited for the content to appear.

2.3 Summary

This chapter introduced some of the technology systems used in this Thesis. The first section dealt with the Global Positioning System and the Longitude/Latitude Co-ordinate system. The second section provided some background information on the TCP/IP and HTTP protocols. An understanding of the information presented in this chapter will lead to a better overall understanding of the system requirements outlined in chapter 3 and the detailed system description in chapter 4.

Chapter 3 - System Requirements and Architecture

This chapter discusses the requirements for the position tracking system which will be developed. Also presented is the outline for the basic architecture of the system. The position tracking system developed in this thesis consists of three major sections: 1) Data acquisition, 2) Data processing, 3) Data distribution. Each of these topics will be investigated in this chapter and fully detailed in chapter four.

3.1 System Requirements

The requirements for the tracking system are as follows:

- Wireless data reporting and world-wide accessibility to data.
- Support for multiple data-reporting clients.
- Low cost.
- Portable and easy to use client and server software.
- Position accuracy to 100 meters horizontally and 156 meters vertically, 95% of the time.

3.2 System Architecture

The system developed in this thesis deals with the acquisition, processing and distribution of accurate location data. A GPS receiver connected to a portable computer will provide the location data for the system. The data acquisition is to be performed by a client application running on this portable computer. The distribution and analysis of the data will be performed by both the client application and a server application running on an Internet connected computer. The client and server application will communicate via a cellular modem connection to the Internet. Since a single server application can track multiple client systems, an attempt will be made to keep the cost of the client systems to a minimum.

3.2.1 Data Acquisition

The location data will be collected from an OEM GPS receiver module. The chosen receiver must include support for computer interfacing as well as a simple data exchange protocol. It must also be able to provide the vertical and horizontal accuracy required by the system. To keep this system affordable, the portable computer to be used for acquisition must itself be quite low cost, and consequently, not very powerful. This being the case, the operating system under which the client application is run should be one which will not add much processing and memory overhead to the computing tasks of the client.

3.2.2 Data Processing

In order to prevent the client and server application from being overloaded, the task of data processing must be distributed between both applications. The client application will communicate with the GPS receiver, process the communication protocol data and send raw longitude and latitude information to the server via a cellular modem. The server will further process the data by linking the co-ordinates to readable location names, if the location name is known. The server will also archive the client's location information for later retrieval.

3.2.3 Data Distribution

The server application will communicate with both the outside world, and with the client. The outside world being any user who wishes to access the stored location data. The server itself will perform the function of a web server and will also communicate with the client application via a proprietary protocol over a TCP/IP network. Since only one host computer is needed to run the server application, it will not be restricted by cost or performance issues.

3.4 Summary

This chapter discussed the requirements of the position tracking system to be developed in this thesis. It also presented the basic architecture of the system. It was seen that the system could be separated into the acquisition, processing and distribution of data. Now that the requirements for these topics have been defined, they will be explained in further detail in chapter 4.

Chapter 4 - Detailed System Description

This chapter provides a detailed description of the position tracking system. As in the previous chapter, the description will be broken down into the acquisition, processing and distribution of the location data. Also included is a section detailing some of the problems that occurred during development and how those problems were solved. Both the hardware and software aspects of the thesis will be described in the following sections.

4.1 Data Acquisition

The data acquisition system is comprised of a GPS receiver module, a client application and a portable computer equipped with a cellular modem. Each of these components and how they interact will be discussed in this section.

4.1.1 GPS Receiver Module

The GPS receiver used in this thesis is the ACE II GPS module, manufactured and distributed by Trimble Navigation Limited. The module itself is approximately equal in size to a credit card, (82.6mm x 46.5mm x 11.5mm), and can be seen in figure 4.1.

The ACE II is a SPS receiver that can track up to 8 SVs simultaneously. The ACE II provides the standard SPS accuracy for Longitude, Latitude and time. The typical location acquisition time claimed by Trimble for the receiver from a cold start, (no battery backup knowledge of its location, time, or SV positions), is less than 130 seconds, 90% of the time. However, through experimentation, acquisition times of 2 to 12 minutes have been observed. To decrease the acquisition time, a 3.6-volt lithium backup battery can be added to allow RAM storage of time, last known location and possible SV locations.

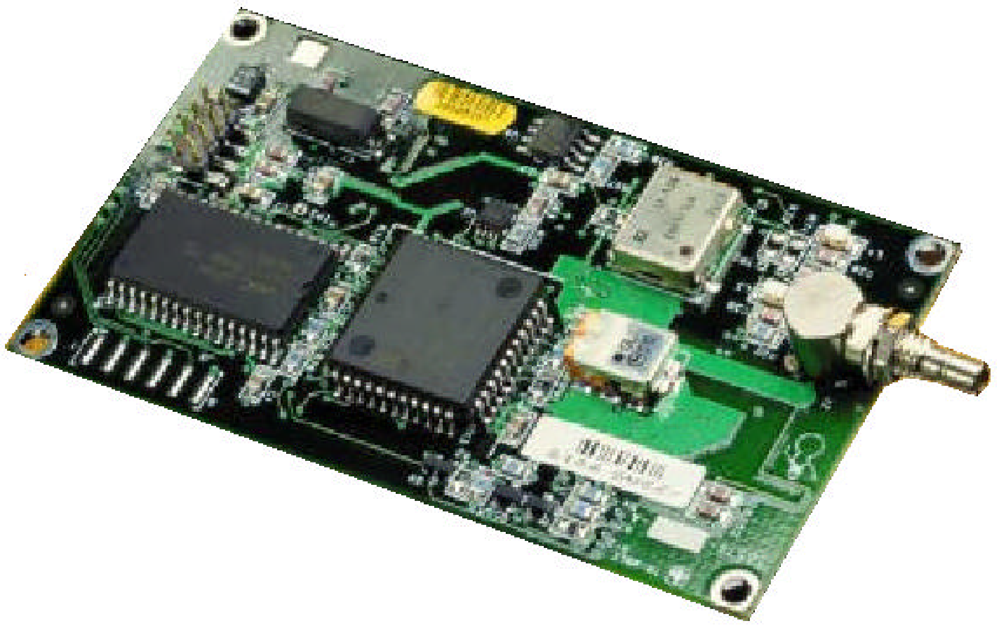
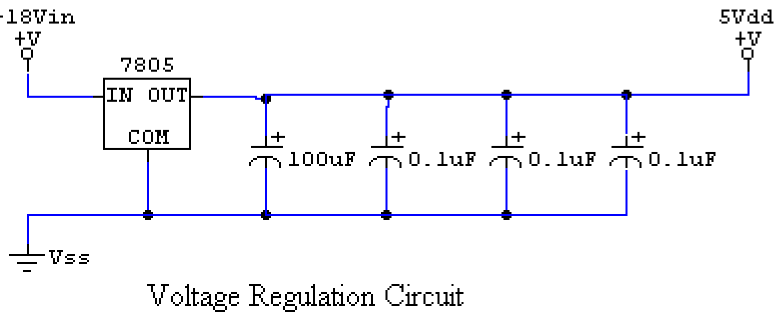


Figure 4.1 - The ACE II OEM GPS Receiver

TAIP, an ASCII based protocol developed by Trimble facilitates communication with the receiver over a serial connection. Using the TAIP protocol, the device used in this thesis was configured to report location data on a set time interval. This time interval can be configured within the client application. More information on TAIP can be found in Appendix A.

Serial communication between the receiver and the portable computer is established via a standard RS-232 connection. Since different machines can have different internal signal levels, RS-232 was created as a standard interface for inter-device communication [KINS91]. With ordinary CMOS technology, a logical one is represented by +5 volts and logical zero with 0 volts. In RS-232, logical one is represented by a negative voltage in the range of -3 to -25 volts and logical zero by a positive voltage in the range of +3 to +25. The ACE II input/output pins for serial communication were designed for CMOS logic levels and the portable computers serial port was designed for RS-232 levels, therefore, a small conversion circuit was needed. A MAX232 chip by MAXIM was utilized for the voltage level conversion.

Another small circuit was needed for the power supply of the receiver. The receiver requires a power supply of +5 volts DC with an accuracy of 5% and that can supply a maximum current of 175mA. This supply was implemented with a 7805 voltage regulator and some filtering capacitors. The voltage regulator takes in voltage from +8 volts to +18 Volts DC and outputs a +5 volt DC signal. A standard 9-volt DC wall adapter was used as a supply for the circuit during development. The circuit was designed in such a way that a standard 9-volt battery could be used to power the device for field trials. The schematics for the power circuit and the RS-232 conversion circuit can be seen in figure 4.2. The RX and TX signals in the conversion circuit represent the receive and transmit communication lines.



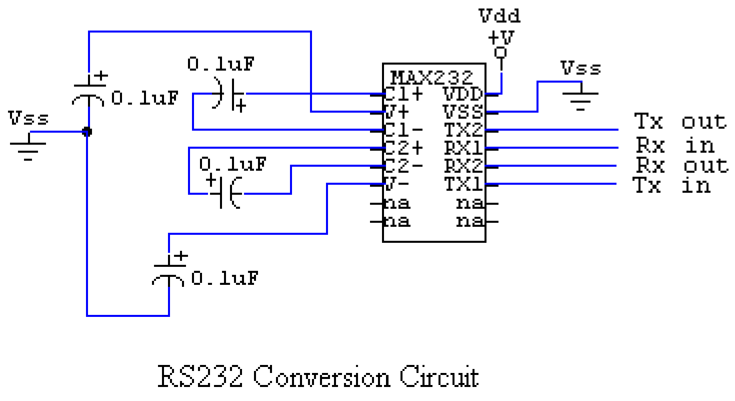


Figure 4.2 - Power Supply and RS-323 Regulation Circuits

The cost of the ACE II GPS receiver and its associated Mighty Mouse Antenna was $410. This cost, however, would decrease significantly for production size orders of the device.

4.1.2 Portable Computer System

The development system for this thesis was a desktop Pentium II 350 with 128 Megabytes of RAM, running Windows 98 as an operating system. This system provided a solid development and testing atmosphere. Although it is not the state of the art in current computer technology, a portable system of this calibre would not come at a low cost. A more appropriate portable system would be as follows: A laptop computer powered by a low end Pentium processor with 16 Megabytes of Ram and a on-board cellular modem running the Linux operating system.

Linux is a Unix type operating system and was chosen for four reasons:

1) It is totally free to use.

2) It comes with built-in support for TCP/IP and serial communication.

3) It adds little overhead to processing and system memory, and can be run on older, less powerful machines.

4) Computer programs written with care can run on both Windows and Linux based system.

A used laptop computer, as described above, can be purchased for a couple of hundred dollars, if not for less.

4.1.3 Client Software

As mentioned in the previous section, the client software must be written in such a manner as to allow it to run under both Windows 98, (the development system), and Linux, (the production system). Two possible computer-programming languages were researched as possible cross-platform development solutions. The two languages were Java from Sun Microsystems, and ANSI C++.

Sun Microsystems cites Java as *the* platform independent programming language of choice for today's programmers. Java includes support functions for TCP/IP and serial communication. As Java is an interpreted language, a Java Interpreter must be present to process the Java program in order for it to run. The interpretation of a Java program tends to be quite CPU and memory intensive.

Dennis Ritchie created the C programming language at Bell Telephone Laboratories in 1972. The language was created to aid in the design of the UNIX operating system. The language was standardised in 1983 by ANSI and there now exists ANSI compliant C compilers for almost every computer platform. The C++ language was created to add extra functionality to C as well as to address certain problems inherent to the C language.

Given its strong ties to the UNIX/Linux operating system, as well as the overhead of Java interpretation, C++ was chosen for client development. Special care had to be taken to provide full compatibility for Windows and Linux. Cross platform programming libraries were developed for TCP/IP and serial communication. These libraries along with the source code for the rest of the client can be found in Appendix B.

The client application has two basic tasks; to collect GPS data via a serial connection, and to report location data to the server via a TCP/IP connection. The client can also request the server to tag co-ordinates within a certain radius with a site specific name.

Using the cross platform serial routines, along with the TAIP protocol, the client can configure the GPS receiver to output its current location on a set schedule. This schedule also defines when the client will report location data to the server. The client processes the reported data and uses the TCP/IP routines to send the data to the server. The processing of data, as well as the application layer protocol used for client/server communication, will be discussed in the following section.

4.2 Data Processing

The client and the server applications share the task of processing the GPS data. The client processes the TAIP messages sent by the GPS receiver and then communicates the results to the server. The server further processes and stores the information before making it available for viewing.

4.2.1 Client - Receiver Communication Processing

The TAIP messages that the client receives from the GPS receiver must be processed in order to extract the longitude and latitude information. The co-ordinates will be sent to the server in decimal degree form. The server must then convert the decimal degree information into degrees, minutes and seconds. For example, the server could receive a value +37.39438 for latitude. The minutes would be determined by 0.39438 x 60 = 23.6628 minutes. The seconds would be determined by 0.6628 x 60 = 39.768 seconds.

4.2.2 Client - Server Communication Processing

The client can send two distinct types of messages to the server, a location history message and a location logging message. The location history message is the message the client sends each time the GPS receiver reports its location. The location history message contains a latitude, longitude, and the time at which the data was captured. The location logging message is the message which the client sends when the user wants to tag the current co-ordinates with a site specific name. The location logging message contains a latitude, longitude, a site specific location name and the radius of validity in meters. The format of the messages used for client/server communication can be seen in table 4.1.

##ACCCDDDDDEEEEFFFFF{G}	
##	Indicates the start of a new message
A	The message type indicator. This can be either an 'H' for location history data or an 'L' for location logging data.
CCCDDDDD	The latitude in decimal degree form with a decimal point between the Cs and Ds. (eg. +4985983)
EEEEFFFFF	The longitude in decimal degree form with a decimal point between the Es and Fs. (eg. -09713393)
G	For messages of type H this is a time string. For messages of type L this is a location and a radius of validity.
{x}	Signifies that x may occur any number of times

Table 4.1 - The Structure of Client/Server Messages

4.2.3 Server Information Processing

Once the server has received the location information from the client, it must store this information into a file for future retrieval. The server also has a file containing site specific location information. This location file contains a longitude, latitude, location name and a radius of validity. If a user requests a co-ordinate via the web server, the server must calculate if it falls within any of the co-ordinate ranges in the location file. In order to do this, the server calculates the distance between the requested point and each of the co-ordinates in the location file using the greater circle distance formula. If the calculated length is shorter than that of the specified radius then the locations match and the information returned to the user will be tagged with the location name. The greater circle distance formula determines how many degrees apart two longitude/latitude co-ordinates are. The WGS84 Datum defines the circumference of the earth to be 40007.863 km and therefore, one degree is equal to the circumference divided by 360 degrees. The following is a simple example of this distance calculation using the co-ordinates for Winnipeg, Manitoba, Canada and Sydney, Australia.

Winnipeg:

Winnipeg Latitude = LatW = 49.9 in decimal degrees

Winnipeg Longitude = LonW = -97.13 in decimal degrees

Sydney, Australia:

Sydney Latitude = LatS = -33.92 in decimal degrees

Sydney Longitude = LonS = 151.28 in decimal degrees

Difference in Latitude = dlat = latW - latS

Difference in Longitude = dlon = lonW - lonS

Meters per Degree = 40007.863km/360° = 111132.95 meters/degree

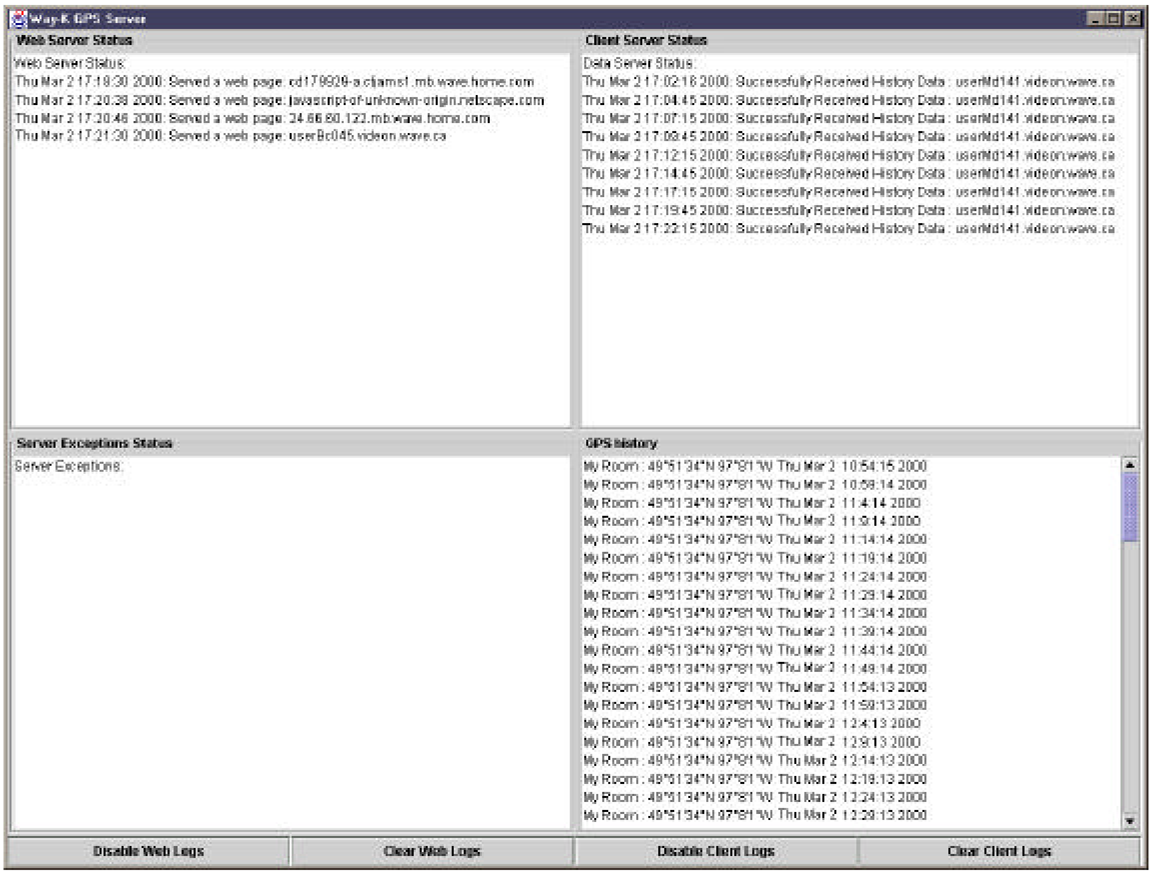
Degrees = 2*ASIN(((SIN((dlat)/2)^2)+COS(latW)*COS(latS)*(SIN((dlon)/2))^2)^-2)

Degrees = 128.57°

Distance = degrees * 111132.95 meters/degree = 14288.36 km

4.3 Data Distribution

The distribution of the GPS data is carried out by the server application. The server can be separated into a front-end GUI, (Graphical User Interface), shown in figure 4.3, an HTTP web server and a data upload server. The data upload server was discussed in sections 4.2.2 and 4.2.3. Each portion of the application is run within a separate thread. Using threads within a program allows separate sub-sections of a program to run independently and concurrently. Much like the client, the server was written as a multi-platform program. Unlike the client, the server was written entirely in the Java programming language. This choice was made due to the simplicity of thread usage in the Java language and because the server's host computer was not constrained by cost or performance issues. In order to simplify the testing of the system, the server application was designed to track a single client agent in the field. However, due to the object oriented nature of the Java language and the way in which the server was designed, multi-client functionality could be easily added. The Java source code for the server can be found in Appendix C.



```
Web Server Status:
Thu Mar 2 17:18:30 2000: Served a web page: cd179929-a.cjams1.mb.wave.home.com
Thu Mar 2 17:20:38 2000: Served a web page: javascript-of-unknown-origin.netscape.com
Thu Mar 2 17:20:46 2000: Served a web page: 24.66.80.122.mb.wave.home.com
Thu Mar 2 17:21:30 2000: Served a web page: userBc045.videon.wave.ca
```

```
Data Server Status:
Thu Mar 2 17:02:16 2000: Successfully Received History Data : userMd141.videon.wave.ca
Thu Mar 2 17:04:45 2000: Successfully Received History Data : userMd141.videon.wave.ca
Thu Mar 2 17:07:15 2000: Successfully Received History Data : userMd141.videon.wave.ca
Thu Mar 2 17:09:45 2000: Successfully Received History Data : userMd141.videon.wave.ca
Thu Mar 2 17:12:15 2000: Successfully Received History Data : userMd141.videon.wave.ca
Thu Mar 2 17:14:45 2000: Successfully Received History Data : userMd141.videon.wave.ca
Thu Mar 2 17:17:15 2000: Successfully Received History Data : userMd141.videon.wave.ca
Thu Mar 2 17:19:45 2000: Successfully Received History Data : userMd141.videon.wave.ca
Thu Mar 2 17:22:15 2000: Successfully Received History Data : userMd141.videon.wave.ca
```

```
Server Exceptions:
```

```
My Room : 49°51'34"N 97°8'1"W Thu Mar 2 10:54:15 2000
My Room : 49°51'34"N 97°8'1"W Thu Mar 2 10:59:14 2000
My Room : 49°51'34"N 97°8'1"W Thu Mar 2 11:4:14 2000
My Room : 49°51'34"N 97°8'1"W Thu Mar 2 11:9:14 2000
My Room : 49°51'34"N 97°8'1"W Thu Mar 2 11:14:14 2000
My Room : 49°51'34"N 97°8'1"W Thu Mar 2 11:19:14 2000
My Room : 49°51'34"N 97°8'1"W Thu Mar 2 11:24:14 2000
My Room : 49°51'34"N 97°8'1"W Thu Mar 2 11:29:14 2000
My Room : 49°51'34"N 97°8'1"W Thu Mar 2 11:34:14 2000
My Room : 49°51'34"N 97°8'1"W Thu Mar 2 11:39:14 2000
My Room : 49°51'34"N 97°8'1"W Thu Mar 2 11:44:14 2000
My Room : 49°51'34"N 97°8'1"W Thu Mar 2 11:49:14 2000
My Room : 49°51'34"N 97°8'1"W Thu Mar 2 11:54:13 2000
My Room : 49°51'34"N 97°8'1"W Thu Mar 2 11:59:13 2000
My Room : 49°51'34"N 97°8'1"W Thu Mar 2 12:4:13 2000
My Room : 49°51'34"N 97°8'1"W Thu Mar 2 12:9:13 2000
My Room : 49°51'34"N 97°8'1"W Thu Mar 2 12:14:13 2000
My Room : 49°51'34"N 97°8'1"W Thu Mar 2 12:19:13 2000
My Room : 49°51'34"N 97°8'1"W Thu Mar 2 12:24:13 2000
My Room : 49°51'34"N 97°8'1"W Thu Mar 2 12:29:13 2000
```

Figure 4.3 - Server Application Screenshot

4.3.1 Front-end Server GUI

The goal of the front-end GUI for the server was to provide the user (i.e. the administrator of the server) with feedback regarding the status of the web server and the data server, the location history of the client and any communication errors that have occurred. The GUI, as seen in Figure 4.3, is separated into four text-based information areas and a button panel.

The web server status area provides information on when the web pages were accessed and by which Internet address. The client server, (or data server), status area provides information on when the client uploaded either history or location logging information. The Internet address of the client is also shown in the client server status area. The GPS history status area displays the last N locations of the client and the times at which the locations were recorded. The number of locations displayed can be configured within the server application. The final display area is the server exceptions status area: In this area the user is given information on any communication problems that have occurred. These communication problems can range from TCP/IP errors to parsing error due to invalid client data. The button panel allows the user to clear or freeze the data or web server status areas.

4.3.2 Back-end HTTP Server

The location data collected by the client application is made accessible via the World Wide Web by means of a web server thread. The web server binds itself to the TCP/IP port 80, the standard port for web servers, and waits for requests from web browsers. When a web browser connects to the web server, a new and separate thread is spawned in order to deal with the request. This is done in order to allow multiple web browsers to simultaneously access the web server. The new request thread processes the browser's HTTP request and sends back an HTTP response along with the location information of the GPS client. Much like the GUI, the server responds with the last N locations of the client and the times at which the locations were recorded. Again, the value of N can be configured within the server. Once the response is sent the request thread is terminated.

4.4 Other Development Issues

This section will provide a brief overview of some of the major problems that occurred during the development of the tracking system. Also described will be the solutions used to overcome these problems.

4.4.3 Thread Synchronization Issues

The main issue that surfaced while programming the multi-threaded server was that of object synchronization. In order to design the server with multi-client expandability in mind, the issue of multiple clients simultaneously uploading data had to be considered. Since all the data would be stored within a single database, an object locking mechanism was included to prevent file contention. This same locking system was also used to prevent contention issues with other internal server objects that could be written to simultaneously by various threads.

4.4.4 Multi-Platform Compliance

A fairly large portion of development time was spent modifying the client application so that it would run in an identical manner under both Windows 98 and Linux. In most cases, the cause of encountered problems was either the absence of a specific C++ function for one of the operating systems or a function that would operate in a different manner for each operating system. These problems were eventually resolved by considerable research and development.

4.5 Summary

This chapter provided a detailed description of the position tracking system developed in this thesis. The core technologies used in the acquisition, processing and distribution of data were explained. A brief overview of some development problems and their associated solutions was discussed. Also, the choice of certain technologies was explained in reference to the system requirements defined in chapter 3.

Chapter 5 - Experimentation

This chapter will provide an overview of the tests which were performed using the tracking system. These tests were done in order to confirm that the system was operating according to the requirements of this thesis. Both the procedure and the results of the tests will be detailed within the following sections.

5.1 Description Of Experiments

A series of field trials and assessments were used to test the tracking system for compliance with the requirements of this thesis. Not all of the requirements defined in section 3.1 could be accessed. The following are the requirements from section 3.1, restated:

- Wireless data reporting and world-wide accessibility to data.
- Support for multiple data-reporting clients.
- Low cost.
- Portable and easy to use client and server software.
- Position accuracy to 100 meters horizontally and 156 meters vertically, 95% of the time.

5.1.1 Requirements That Could Not Be Tested

The ability to test the wireless data reporting relied on the availability of a laptop with a cellular modem. Although a cellular modem was not available for testing purposes, section 5.1.3 will detail how the tracking system was still testable. Also, since there was only one client system built, it was impossible to test the server's support for multiple clients. Chapter 4 details that the server was written for single client tracking but designed with multi-client expansion in mind. Finally, an actual cost evaluation for the system would be very hard to determine for a number of reasons. The first being that if the system were to be developed commercially the cost would be determined by evaluating bulk costs for the various components. The bulk cost of components varies according to the number of components purchased. Second, a fairly large portion of the cost would be determined by the cost of the client and server host computers. In most cases, cheap or used laptops could be used for the client system and most companies or casual users would have existing Internet connected computers on which the server could run unobtrusively. This being said, steps were taken throughout the thesis to ensure a low cost system.

5.1.2 Remaining Tests and Results

Evaluating the ease of use of the server and client applications is a very subjective procedure. However, extreme care was taken during the programming and design phases of development in order to ensure a simple yet powerful interface for both applications. The testing phase of the thesis lead to further refinements in the interfaces for both the client and the server. The portability of the programs was fairly simple to evaluate. Due to the fact that the server was written using the Java language it can be run on any operating system that supports the Java language. In order to test the client program's portability between Windows 98 and Linux, the remaining system tests were run under both operating systems.

In order to evaluate the accuracy of the tracking system the following test was devised. The test comprised of leaving the client at a fixed location for a period of 500 minutes. The GPS device was configured to report its location once every 5 minutes resulting in 100 recorded locations. Since the accuracy of the system must be 100 meters horizontally and 156 meters vertically, the maximum linear error, (using pythagoras' theorem of right angle triangles), is approximately 185 meters. At the beginning of the 500 minute test, the client was used to tag the current location with a specific name on the server with a radius of validity of 185 meters. After the 100 locations were logged the location data was observed via the web server. This test was run twice using the Windows 98 client and twice using the Linux client. During the first two Windows 98 test runs, 97 of the 100 locations were reported to be within 185 meters of the initial location. After the first Linux test run, 98 locations were within 185 meters of the initial location. The second Linux test run yielded 97 locations within the required accuracy range.

The general usability of the system and further accuracy tests were conducted by uploading location names for various locations around town. These locations were revisited 3 or 4 days later in order to see if system recognized the location co-ordinates and tagged them appropriately on the web data. No problems were encountered during this phase of testing which was conducted with both the Windows 98 and Linux client.

5.1.3 Testing Without A Cellular Modem

Initially, the thought of testing a wireless system with a wireless means of communication seemed impossible if the system was to be kept mobile. However, the way in which the client and server applications were designed suggested a simple solution. The TCP/IP protocols can be used in an identical manner independent of underlying physical communication medium, whether it be a wireless modem connection to the Internet or a physical connection to an Internet connected network or LAN. The property of TCP/IP that was exploited for the solution is the ability of a TCP/IP capable computer system to establish connections with themselves without the presence of a physical communication medium. This property allows the server and client applications to communicate by running both programs simultaneously on the same laptop computer. Using the same property, a web browser run on the laptop could also connect to the server application to download location data. This solution along with the ability to run the GPS hardware off a 9-volt battery gave the system full freedom of motion for the testing phase of the thesis.

5.2 Summary

This chapter detailed the results and methods of the tests that were performed using the tracking system. The system was shown to be compliant with the requirements and expectation of this thesis. This section also showed how various problems that arose during testing were overcome.

Chapter 6 - Conclusion and Recommendations

This thesis was motivated by the need for a low cost wireless tracking system for the general public. Unlike currently available systems, the system designed in this thesis provides world-wide accessibility to the real-time location data collected by the tracking device, while still providing full freedom of motion for the object being tracked.

As mentioned in chapter one, the system could be used to decrease emergency vehicle response time, provide clients of public transportation with accurate arrival and departure times, and perhaps, for somewhat less ethical purposes, such as parental tracking of unaware teenagers. Since the client component of the system was designed to be cost effective and is relatively small in size, the entire system could be directly embedded into future laptop computers to provide a variety of services, including tracking stolen laptops.

It was demonstrated that the system can provide the accuracy required by this thesis and that it functioned in a predictable manner. Although the testable requirements of the thesis were satisfied, there is still room for system improvement. The following are some recommendations for future work with the tracking system:

- A system upgrade to provide location data with greater accuracy. In order to increase the accuracy while still using SPS GPS data, the system can communicate with a GPS receiver placed at a known fixed location. The information gathered by this receiver could be used to provide enhanced timing correction information. This is known as DGPS or Differential GPS, and when used with SPS data it can provide accuracy within 1-10 meters.

- No complex computer security issues were addressed in this thesis. Any rogue client with the system password could upload false location data to the server. Further, the web server provides no way of restricting access to the location data. Authentication and encryption procedures could be implemented to solve these problems.
- As mentioned in earlier chapters, the system created in this thesis can presently only track a single client agent. By building more client devices and expanding the server program the system could be converted into a fully functional multi-client tracking system.

References

[ROS99] Rosenberg, Matt. "*Latitude and Longitude*." About.com Inc. 28 Nov. 1999. <http://geography.about.com/education/geography/library/weekly/aa031197.htm>

[USN99] **"***GPS System Description*." United States Naval Observatory. 12 Nov. 1999. <ftp://tycho.usno.navy.mil/pub/gps/gpssy.txt>

[TRI97] "*GPS Tutorial*." Trimble Navigation Limited. 1997 <http://www.trimble.com/gps/index.htm>

[KINS91] Kinsner, W. "*Microprocessor Interfacing Laboratory Notes."* University of Manitoba. Sep. 1991.

[ACE98] "*ACE II GPS. System Designer Reference Manual.*" Trimble Navigation Limited. Jun. 19998.

Appendix A - TAIP Protocol Information

The follow protocol information was taken from [ACE98].

Message Format

>ABB{C}[;ID=DDDD][;*FF]<

>	Start of message delimiter
A	Message qualifier
BB	A two character message identifier
C	Data String
DDDD	Optional four character vehicle ID
FF	Optional two character checksum
<	End of message delimiter
{x}	Signifies that x can occur any number of times
[x]	Signified that x may optionally occur once

Table A.1 - TAIP Message Format

Message Qualifiers

Qualifier	*Action*
Q	Query for a single sentence (sent to GPS sensor)
R	Response to a query or a scheduled report (from the sensor)
F	Schedule reporting frequency interval in seconds
S	Enables equipment to be initialized, and sets various message types
D	Specify a minimum distance travelled and a minimum and maximum time interval for the next report

Table A.2 - TAIP Message Qualifiers

Sample PV Message

The Position/Velocity Solution (PV) message is one of the more commonly used TAIP messages and most sensors using TAIP are set by default to output the PV message once every 5 seconds.

The following analysis of a typical PV message is provided to further explain the TAIP message protocol.

>RPV15714+3739438-1220384601512612;ID=1234;*7F<

ID	*Meaning*
>	Start of Message Delimiter
R	Response Qualifier
PV	PV message Identifier
15714	GPS Time of Day
+3739438	Latitude
-12203846	Longitude
015	Speed
126	Heading
1	Source of Data
2	Age of Data
;ID=1234	Vehicle ID
;*7F	Checksum
<	End of Message Delimiter

Table A.3 - Sample TAIP PV Message

Appendix B - Client Source Code (C++) and Resource Files

GPSclient.cpp

```
/*
 * GPSclient.cpp
 *
 * Writen by: Kyle Andrew McGrath Geske
 *
 * Created as a requirement of my Undergraduate thesis in
 * Computer Engineering for the University of Manitoba.
 *
 * Description: The file contains the program's mainline.
 *
 * Copyright Kyle Geske 1999/2000
 *
 */


#include <iostream.h>
#include <string.h>

#include "init.h"
#include "GPSclient.h"
#include "ops.h"

void main(int argc, char **argv)
{
  ++argv,--argc; // Remove the 0th Arg.

  cout << "Way-K GPS Client. Version " << VER_HIGH << "." << VER_LOW << endl;

  if ((argc > 0) && (!strcmp(argv[0],"-v")))
  {
    config_it(true); // Configure (Sockets, Serial Ports, I/O, Log Files, GPS)
  }
  else
  {
    config_it(false); // Configure (Sockets, Serial Ports, I/O, Log Files, GPS)
  }

  ops(); // Begin normal operations.

  unconfig_it(); // Shutdown everything. (Sockets, Serial Ports, I/O, Log File)

}
```

GPSclient.h

```
/*
 * GPSclient.h
 *
 * Writen by: Kyle Andrew McGrath Geske
 *
 * Created as a requirement of my Undergraduate thesis in
 * Computer Engineering for the University of Manitoba.
 *
 * Description: Header files for GPSclient.cpp
 *
 * Copyright Kyle Geske 1999/2000
 *
 */

#ifndef WAYK_GPSCLIENT_H
#define WAYK_GPSCLIENT_H

#define VER_HIGH 0
#define VER_LOW 1

#endif // WAYK_GPSCLIENT_H
```

init.cpp

```
/*
 * init.cpp
 *
 * Writen by: Kyle Andrew McGrath Geske
 *
 * Created as a requirement of my Undergraduate thesis in
 * Computer Engineering for the University of Manitoba.
 *
 * Description: This file contains functions used to
 * parse the configuration file in order to properly
 * configure the various sub-systems used in the
 * program.
 *
 * Copyright Kyle Geske 1999/2000
 *
 */

// Include Files

#include <iostream.h>
#include <string.h>

#include "init.h"
#include "file_io.h"
#include "net.h"
#include "serial.h"
#include "gps_comm.h"

// Local Static Variables and Structures

static struct config_Data config[CONFIG_ITEMS];
static char config_File[10] = "data.cfg";

// Functions

void config_it(bool verbose)
{
  set_file_verbose(verbose);
  set_filename(config_File,CONFIG_FILE);

  if (open_file('i',CONFIG_FILE))
  {
    config_from_file();
    close_file(CONFIG_FILE);
  }

  set_filename(config_value("ERRORFILE"),ERROR_FILE);
  set_filename(config_value("LOGFILE"),LOG_FILE);
  open_file('a',ERROR_FILE);
  open_file('a',LOG_FILE);

  LOG_REPORT("GPS Client Startup. (operations log)");
  ERROR_REPORT("GPS Client Startup. (error log)");

  wayk_Net_Init();
  Serial_Init(config_value("COMPORT"),config_value("COMSET"));
  Gps_init(config_value("LNTIME"),config_value("PVTIME"),config_value("IDNUM"),
verbose);
  cout.flush();
}

void config_from_file()
{
  struct config_Data *config_Ptr = config;

  for(int i = 0; i < CONFIG_ITEMS; i++)
  {

    strcpy(config_Ptr->key,read_file('!',CONFIG_FILE));
    strcpy(config_Ptr->value,read_file('!',CONFIG_FILE));
    if (!strcmp(config_Ptr->value,"END"))
      break;
    config_Ptr++;
  }

}

const char* config_value(const char *key)
{
  for(int i=0; i<CONFIG_ITEMS; i++)
  {
    if (!strcmp(config[i].key,key))
      return config[i].value;
  }

  return "";
}


void unconfig_it()
{
  LOG_REPORT("GPS Client Shutdown. (operations log)");
  ERROR_REPORT("GPS Client Shutdown. (error log)");

  close_file(ERROR_FILE);
  close_file(LOG_FILE);

  wayk_Net_Cleanup();

  Serial_Cleanup();
}
```

init.h

```
/*
 * init.h
 *
 * Writen by: Kyle Andrew McGrath Geske
 *
 * Created as a requirement of my Undergraduate thesis in
 * Computer Engineering for the University of Manitoba.
 *
 * Description: Header files for init.cpp
 *
 * Copyright Kyle Geske 1999/2000
 *
 */

#ifndef WAYK_INIT_H
#define WAYK_INIT_H

#define CONFIG_ITEMS 11

struct config_Data
{
  char key[50];
  char value[50];
};

void config_it(bool verbose);
void unconfig_it();
const char* config_value(const char *key);
void config_from_file();

#endif
```

ops.cpp

```
/*
 * ops.cpp
 *
 * Writen by: Kyle Andrew McGrath Geske
 *
 * Created as a requirement of my Undergraduate thesis in
 * Computer Engineering for the University of Manitoba.
 *
 * Description: This file contains functions to control
 * the main operations of the program.
 *
 * Copyright Kyle Geske 1999/2000
 *
 */

#if defined(WAYK_OS_WIN32)
  #include <winsock2.h> // Windows Network Include File
#elif defined(WAYK_OS_LINUX)
  #include <arpa/inet.h>  // Linux Netowrk Include File
  extern "C"
    { #include "kbhit.h" }
#endif

// Shared Include Files

#include <iostream.h>
#include <stdio.h>
#include <conio.h>
#include <Windows.h>

#include "ops.h"
#include "gps_comm.h"
#include "net.h"
#include "time.h"
#include "file_io.h"

// Functions

void ops(void)
{
  bool done = false;
  char choice;


  while(!done)
  {
    pollGPS();

    choice = getch();

    switch(choice)
    {
      case 'q' : done = true; break;
      case 'l' : log_it(); break;
      default  : cout << "Invalid Key! [" << choice << "]" << endl;
    }
  }
}
```

```
void pollGPS(void)
{

  boolean full;
  char message[160];
  char *sendData;

  bool allgood = true;

  while(!kbhit())
  {

    bool processed = false;
    full = Receive_Gps_Message(message);
    if (full)
    {
      if (message[2] == 'L')
      {
        sendData = (char *)processLN(message,(char *)getTime(),'H');
        processed = true;
      }
      else if (message[2] == 'P')
      {
        sendData = (char *)processPV(message,(char *)getTime(),'H');
        processed = true;
      }

      if (processed)
      {
        SOCKET sd = Establish_Sock_Connection(get_Current_Hostname(),31337);
        if (Valid_Socket(sd))
        {
          Send_Sock_Message("letmein\n",sd);
          Send_Sock_Message(sendData,sd);
          Shutdown_Sock_Connection(sd);
        }
        else
        {
          ERROR_REPORT("Socket connection could not be made. History data
lost");
        }
      }

    } // if full
  }
}

void log_it(void)
{

  char message[160];
  char name[80];
  int radius;
  char *sendData;
  char *ptr;
  bool full = false;
  bool processed = false;

  Set_Gps_Paused(true);

  cout.flush();
  cout << "Name of Location:" << flush;
  cin.getline(name,80);
  cout << "Radius of Validity:" << flush;
  cin >> radius;
  ptr = name;
  ptr += strlen(name);

  sprintf(ptr,"@%d",radius);

  Send_Gps_Message("QLN");

  while (!full)
  {

    full = Receive_Gps_Message(message);

    if (full)
    {
      if (message[2] == 'L')
      {
        sendData = (char *)processLN(message,name,'L');
        processed = true;
      }
      else if (message[2] == 'P')
      {
        sendData = (char *)processPV(message,name,'L');
        processed = true;
      }
    }
  }

  if (processed)
  {
    SOCKET sd = Establish_Sock_Connection(get_Current_Hostname(),31337);
    if (Valid_Socket(sd))
    {
      Send_Sock_Message("letmein\n",sd);
      Send_Sock_Message(sendData,sd);
      Shutdown_Sock_Connection(sd);
    }
    else
    {
      ERROR_REPORT("Socket connection could not be made. History data lost");
    }
  }

  Set_Gps_Paused(false);
}

char* processPV(char *message, char *addon, char type)
{
  char *ptr;
  char latitude[10];
  char longitude[10];
  static char sendData[80];

  ptr = message;
  ptr += 9;
  strncpy(latitude,ptr,8);
  latitude[8] = 0;
  ptr += 8;
  strncpy(longitude,ptr,9);
  longitude[9] = 0;
  // #HN+4985983-09713393Thr Feb 17 2:25:8 2000
  sprintf(sendData,"##%c%s%s%s\n",type,latitude,longitude,addon);
  cout << "Num SVs: ? - "<< sendData << flush;
  return sendData;
}

char* processLN(char *message, char *addon, char type)
{
  char *ptr;
  char latitude[10];
  char longitude[10];
  char numSV[3];
  static char sendData[80];

  ptr = message;
  ptr += 12;
  strncpy(latitude,ptr,8);
  latitude[8] = 0;
  ptr += 10;
  strncpy(longitude,ptr,9);
  longitude[9] = 0;
  ptr += 33;
  strncpy(numSV,ptr,2);
  numSV[2] = 0;
  // #HN+4985983-09713393Thr Feb 17 2:25:8 2000
  sprintf(sendData,"##%c%s%s%s\n",type,latitude,longitude,addon);
  cout << "Num SVs: " << numSV  << " - "<< sendData << flush;
  return sendData;
}
```

ops.h

```
/*
 * ops.h
 *
 * Writen by: Kyle Andrew McGrath Geske
 *
 * Created as a requirement of my Undergraduate thesis in
 * Computer Engineering for the University of Manitoba.
 *
 * Description: Header files for ops.cpp
 *
 * Copyright Kyle Geske 1999/2000
 *
 */

#ifndef WAYK_OPS_H
#define WAYK_OPS_H

#if defined (WAYK_OS_LINUX)
  #define getch() getchar()
#endif

void ops(void);
void pollGPS(void);
void log_it(void);
char* processLN(char *message, char *addon, char type);
char* processPV(char *message, char *addon, char type);

#endif
```

gps_comm.cpp

```
/*
 * gps_comm.cpp
 *
 * Writen by: Kyle Andrew McGrath Geske
 *
 * Created as a requirement of my Undergraduate thesis in
 * Computer Engineering for the University of Manitoba.
 *
 * Description: This file contains functions for ACE II GPS
 * receiver communication. Using the TAIP protocol these
 * function utilize the serial communications library
 * found in serial_linux.cpp and serial_win.cpp.
 *
 * Copyright Kyle Geske 1999/2000
 *
 */

// Include Files

#include <string.h>
#include <stdio.h>
#include <iostream.h>
#include <Windows.h>

#include "serial.h"
#include "gps_comm.h"

// Local Static Variables

static bool verbose;
static char idNum[5];
static bool idset = false;
static char LN[5];
static char PV[5];

// Function Declaration

void wait_4_lock(void);

// Functions

void Gps_init(const char* LNtime, const char* PVtime, const char* id,const bool
verb)
{
  char message[80];

  strcpy(idNum,(char *)id);
  strcpy(LN,(char *)LNtime);
  strcpy(PV,(char *)PVtime);
  verbose = verb;

  Send_Gps_Message("SRM;ID_FLAG=T;CS_FLAG=F;CR_FLAG=F;*6E");  // Turn off
Checksums and include a manually generated checksum
  Receive_Gps_Message(message);

  sprintf(message,"FLN0000");  // Disables LN Messages
  Send_Gps_Message(message);
  sprintf(message,"FPV0000");  // Disable PV Messages
  Send_Gps_Message(message);
  sprintf(message,"SID%s",idNum);  // Set the ID #
  Send_Gps_Message(message);
  Receive_Gps_Message(message);
  idset = true;

  wait_4_lock();  // Wait for a 4 SV lock

  sprintf(message,"FLN%s",LN); // Set the LN message reporting time.
  Send_Gps_Message(message);

  sprintf(message,"FPV%s",PV); // Set the PV message reporting time.
  Send_Gps_Message(message);

}

void Send_Gps_Message(const char* message)
{
  const char *ptr = fmt_taip(message);
  int temp = Send_Serial_Message(ptr,strlen(ptr));
  if (verbose)
    cout << "Sending[" << temp << "]: " << ptr << flush;
}

bool Receive_Gps_Message(char* message)
{
  char ch[1];
  int bytes;
  char *ptr = message;
  bool full = false;

  do
  {
    bytes = Receive_Serial_Message(ch,1);
    if (bytes)
      full = true;
    ptr += sprintf(ptr,"%s",ch);
  }
  while(bytes);

  if (verbose && full)
    cout << "Receiving: " << message << endl;

  return full;
}

bool Query_Gps_Message(const char* query, char* message)
{
  bool full;
  Send_Gps_Message(query);
  full = Receive_Gps_Message(message);
  return full;
}

const char* fmt_taip (const char* msg)
{
  static char buffer[80];

  char *ptr;

  ptr = buffer;
  ptr += sprintf (ptr, ">%s", msg);
  if (idset)
```

```
    ptr += sprintf(ptr,";ID=%s",idNum);
  sprintf (ptr, "<\n");

  return buffer;
}

void wait_4_lock(void)
{
  bool full,locked = false;
  char message[80];

  while(!locked)
  {
    Send_Gps_Message("QST");
    full = Receive_Gps_Message(message);
    cout.flush();
    if (full)
    {
      if ((message[5] == '0') && (message[6] == '0'))
        locked = true;
      else
        cout << message[5] << message[6] << endl;
    }
  }

}

void Set_Gps_Paused(bool pause)
{

  char message[160];

  Receive_Gps_Message(message); //dummy
  if (pause)
  {
    sprintf(message,"FLN0000");  // Disables LN Messages
    Send_Gps_Message(message);
    sprintf(message,"FPV0000");  // Disable PV Messages
    Send_Gps_Message(message);
  }
  else
  {
    sprintf(message,"FLN%s",LN); // Set the LN message reporting time.
    Send_Gps_Message(message);
    sprintf(message,"FPV%s",PV); // Set the PV message reporting time.
    Send_Gps_Message(message);
  }
  Receive_Gps_Message(message); //dummy
}
```

gps_comm.h

```
/*
 * gps_comm.h
 *
 * Writen by: Kyle Andrew McGrath Geske
 *
 * Created as a requirement of my Undergraduate thesis in
 * Computer Engineering for the University of Manitoba.
 *
 * Description: Header files for gps_comm.cpp
 *
 * Copyright Kyle Geske 1999/2000
 *
 */

#ifndef WAYK_GPS_COMM_H
#define WAYK_GPS_COMM_H

void Gps_init(const char* LNtime, const char* PVtime, const char* idNum, const
bool verbose);
void Send_Gps_Message(const char* message);
bool Query_Gps_Message(const char* query, char* message);
bool Receive_Gps_Message(char* message);
const char* fmt_taip (const char* msg);
void Set_Gps_Paused(bool pause);


#endif // WAYK_GPS_COMM_H
```

net.cpp

```
/*
 * net.cpp
 *
 * Writen by: Kyle Andrew McGrath Geske
 *
 * Created as a requirement of my Undergraduate thesis in
 * Computer Engineering for the University of Manitoba.
 *
 * Description: This file contains cross-platform
 * TCP/IP functions for the WIN32 and Linux platforms.
 *
 * Copyright Kyle Geske 1999/2000
 *
 */


#if defined(WAYK_OS_WIN32) // Windows Include Files
  #include <winsock2.h>
  #include <iostream.h>
#elif defined(WAYK_OS_LINUX) // Linux Include Files
  #include <sys/socket.h>
  #include <netdb.h>
  #include <netinet/in.h>
  #include <arpa/inet.h>
  #include <unistd.h>
  #include <errno.h>
#endif

// Shared Include Files

#include "net.h"
#include "net_error.h"
#include "file_io.h"

// Constant Declaration

const int kBufferSize = 1024;

// Functions

#if defined(WAYK_OS_WIN32)

#define WAYK_NET_WINSOCK_VER_MAJOR 2
#define WAYK_NET_WINSOCK_VER_MINOR 0

void Winsock_Init()
{

    WSADATA wsaData;

    int rc = WSAStartup(MAKEWORD(WAYK_NET_WINSOCK_VER_MAJOR,
                                 WAYK_NET_WINSOCK_VER_MINOR), &wsaData);
    if (rc != 0)
        WAYK_NET_ERROR("WSAStartup");

    if ((LOBYTE(wsaData.wVersion) != WAYK_NET_WINSOCK_VER_MAJOR) ||
        (HIBYTE(wsaData.wVersion) != WAYK_NET_WINSOCK_VER_MINOR))
    {
        WSACleanup();
        WAYK_NET_ERROR("Bad Winsock version");
    }

}


void Winsock_Cleanup()
{
    int rc = WSACleanup();
  if (rc == SOCKET_ERROR)
    WAYK_NET_ERROR("WSACleanup");
}

#endif // WAYK_OS_WIN32


// Look up Address (DNS lookup if it's not dotted IP)

u_long LookupAddress(const char* pcHost)
{
    u_long nRemoteAddr = inet_addr(pcHost);
    #if defined(WAYK_OS_WIN32)
    if (nRemoteAddr == INADDR_NONE)
    #elif defined(WAYK_OS_LINUX)
    if (nRemoteAddr < 0)
    #endif
    {
        // pcHost isn't a dotted IP, so resolve it through DNS
        hostent* pHE = gethostbyname(pcHost);
        if (pHE == 0) {
      #if defined(WAYK_OS_WIN32)
            return INADDR_NONE;
      #elif defined(WAYK_OS_LINUX)
      return 0;
      #endif
        }
        nRemoteAddr = *((u_long*)pHE->h_addr_list[0]);
    }

    return nRemoteAddr;
}

// Get the host name of this PC.

const char* get_Current_Hostname()
{
  static char hostname[256];

  gethostname(hostname,256);

  return hostname;
}

// Do we have a valid socket?

bool Valid_Socket(SOCKET sd)
{

  if (sd
  #if defined(WAYK_OS_WIN32)
  != INVALID_SOCKET)
  #elif defined(WAYK_OS_LINUX)
  > 0)
  #endif
    return true;
  else
    return false;

}


// Establish Connection

SOCKET Establish_Sock_Connection(const char* hostAddress, PortType Port)
{

  u_long nRemoteAddr = LookupAddress(hostAddress);
    // Create a stream socket
    SOCKET sd = socket(AF_INET, SOCK_STREAM, 0);
    #if defined(WAYK_OS_WIN32)
    if (sd != INVALID_SOCKET)
    #elif defined(WAYK_OS_LINUX)
    if (sd > 0)
    #endif
    {
        sockaddr_in sinRemote;
        sinRemote.sin_family = AF_INET;
        sinRemote.sin_addr.s_addr = nRemoteAddr;
        sinRemote.sin_port = htons(Port);
        if (connect(sd, (sockaddr*)&sinRemote, sizeof(sockaddr_in))
        #if defined(WAYK_OS_WIN32)
       == SOCKET_ERROR)
        {
            sd = INVALID_SOCKET;
     #elif defined(WAYK_OS_LINUX)
       < 0)
     {
       sd = -1;
     #endif
        }
    }

    return sd;

}

// Set up a listener

SOCKET Setup_Sock_Listener(const char* hostAddress, int nPort)
{
  u_long nInterfaceAddr = LookupAddress(hostAddress);
  cout << "Host address : " << hostAddress << endl;
  cout << "Host IP : " << nInterfaceAddr << endl;
  if (nInterfaceAddr != INADDR_NONE)
  {
    SOCKET sd = socket(AF_INET, SOCK_STREAM, 0);
    if (sd
    #if defined(WAYK_OS_WIN32)
      != INVALID_SOCKET)
    #elif defined(WAYK_OS_LINUX)
      > 0)
    #endif
    {
      cout << "Created socket" << endl;
      sockaddr_in sinInterface;
      sinInterface.sin_family = AF_INET;
```

```
        sinInterface.sin_addr.s_addr = nInterfaceAddr;
        sinInterface.sin_port = htons(nPort);
        if (bind(sd, (sockaddr*)&sinInterface, sizeof(sockaddr_in))
          #if defined(WAYK_OS_WIN32)
            != SOCKET_ERROR)
          #elif defined(WAYK_OS_LINUX)
            == 0)
          #endif
        {
          cout << "bind complete. port:" << nPort << " address : " <<
inet_ntoa(sinInterface.sin_addr) << endl;
          listen(sd, SOMAXCONN);
          cout << "Listen complete" << endl;
          return sd;
        }
        else
         WAYK_NET_ERROR("Bind did not work");
      }
    }

    #if defined(WAYK_OS_WIN32)
    return INVALID_SOCKET;
    #elif defined(WAYK_OS_LINUX)
    return -1;
    #endif
}

// Accept Connection on a Listening Socket

SOCKET Accept_Sock_Connection(SOCKET ListeningSocket, sockaddr_in& sinRemote)
{
    int nAddrSize = sizeof(sinRemote);
    return accept(ListeningSocket, (sockaddr*)&sinRemote, &nAddrSize);
}

// Shutdown Connection Gracefully

bool Shutdown_Sock_Connection(SOCKET sd)
{

    // Disallow futher data sends.
    #if defined(WAYK_OS_WIN32)
    if (shutdown(sd, SD_SEND) == SOCKET_ERROR)
    #elif defined(WAYK_OS_LINUX)
    if (shutdown(sd,1) < 0)
    #endif
    {
      WAYK_NET_ERROR("shutdown()");
    }

    // Grab all left over data.

    char acReadBuffer[kBufferSize];

    while (1)
    {
      int nNewBytes = recv(sd, acReadBuffer, kBufferSize, 0);
      #if defined(WAYK_OS_WIN32)
      if (nNewBytes == SOCKET_ERROR)
      #elif defined(WAYK_OS_LINUX)
      if (nNewBytes < 0)
      #endif
      {
        return false;
      }
      else if (nNewBytes != 0)
      {
        cerr << endl << "FYI, received " << nNewBytes <<
            " unexpected bytes during shutdown." << endl;
      }
      else
      {
        // Okay, we're done!
        break;
      }

    }


    // Close the socket.
    #if defined(WAYK_OS_WIN32)
    if (closesocket(sd) == SOCKET_ERROR)
    #elif defined(WAYK_OS_LINUX)
    if (close(sd) < 0)
    #endif
    {
      return false;
    }

    return true;
  }


  // The send message function

  bool Send_Sock_Message(const char* message, SOCKET sd)
  {
    const int message_Lenght = strlen(message);
    if (send(sd,message,message_Lenght,0)
    #if defined(WAYK_OS_WIN32)
      != SOCKET_ERROR)
    #elif defined(WAYK_OS_LINUX)
      > 0)
    #endif
      return true;
    else
      return false;

  }


  // The Recieve message function
  // Returns the number of bytes received or a 0 if nothing.

  int Receive_Sock_Message(char *message, int len, SOCKET sd)
  {
    return (recv(sd,message,len,0));

  }

  // A bit of a test function which implements a echo client.

  int Send_Sock_Echo(SOCKET sd)
  {
     // Send the string to the server
    const char* kpcEchoMessage = "This is the test message!";

    const int kEchoMessageLen = strlen(kpcEchoMessage);
     if (send(sd, kpcEchoMessage, kEchoMessageLen, 0)
     #if defined(WAYK_OS_WIN32)
    == SOCKET_ERROR)
     #elif defined(WAYK_OS_LINUX)
         < 0)
     #endif
     {
         return -1;
     }
    cout << "Sent Message Waiting for reply" << endl;

     // Read reply from server
     char acReadBuffer[kBufferSize];
     int nTotalBytes = 0;
     while (nTotalBytes < kEchoMessageLen) {
         int nNewBytes = recv(sd, acReadBuffer + nTotalBytes,
                 kBufferSize - nTotalBytes, 0);
         #if defined(WAYK_OS_WIN32)
    if (nNewBytes == SOCKET_ERROR)
    #elif defined(WAYK_OS_LINUX)
    if (nNewBytes < 0)
    #endif
         {
             return -1;
         }
         else if (nNewBytes == 0) {
             cerr << "Connection closed by peer." << endl;
             return 0;
         }

         nTotalBytes += nNewBytes;
     }


    acReadBuffer[nTotalBytes] = '\0';
    cout << acReadBuffer << endl;

     // Check data for sanity
     if (strncmp(acReadBuffer, kpcEchoMessage, nTotalBytes) != 0) {
         cerr << "Mismatch in data received from server. " <<
                 "Something's broken!" << endl;
     }

     return nTotalBytes;
  }
```

net.h

```
/*
 * net.h
 *
 * Writen by: Kyle Andrew McGrath Geske
 *
 * Created as a requirement of my Undergraduate thesis in
 * Computer Engineering for the University of Manitoba.
 *
 * Description: Header files for net.cpp
 *
 * Copyright Kyle Geske 1999/2000
 *
 */

#ifndef WAYK_NET_H
#define WAYK_NET_H

typedef unsigned short PortType;

#if defined(WAYK_OS_WIN32)

  #include <winsock2.h>
  #define wayk_Net_Init() Winsock_Init()
  #define wayk_Net_Cleanup() Winsock_Cleanup()

  void Winsock_Init();
  void Winsock_Cleanup();

#elif defined(WAYK_OS_LINUX)

  #include <netinet/in.h>
  #define wayk_Net_Init()
  #define wayk_Net_Cleanup()
  typedef int SOCKET;

#endif // WAYK_OS_WIN32

const char* get_Current_Hostname();
bool Valid_Socket(SOCKET sd);
SOCKET Establish_Sock_Connection(const char* hostAddress, PortType Port);
SOCKET Setup_Sock_Listener(const char* hostAddress, int nPort);
SOCKET Accept_Sock_Connection(SOCKET ListeningSocket, sockaddr_in& sinRemote);
bool Shutdown_Sock_Connection(SOCKET sd);
int Receive_Sock_Message(char *message, int len, SOCKET sd);
bool Send_Sock_Message(const char* message, SOCKET sd);
int Send_Sock_Echo(SOCKET sd);

#endif //WAYK_NET_H
```

net_error.cpp

```
/*
 * net_error.cpp
 *
 * Writen by: Kyle Andrew McGrath Geske
 *
 * Created as a requirement of my Undergraduate thesis in
 * Computer Engineering for the University of Manitoba.
 *
 * Description: This file contains cross-platform
 * functions for creating human readable error network
 * messages for the WIN32 and Linux Platforms.
 *
 * Copyright Kyle Geske 1999/2000
 *
 */

#if defined(WAYK_OS_WIN32)

// Windows Include Files

#include <iostream>
#include <strstream>
using namespace std;
#include <winsock2.h>

// Constant Declarations

const int kBufferSize = 1024;

//  Local Static Variables and Structures

static struct ErrorEntry
{
  int nID;
  const char* pcMessage;

} gaErrorList[] = {

  { 0,                         "No error" },

  { WSAEINTR,                  "Interrupted system call" },

  { WSAEBADF,                  "Bad file number" },

  { WSAEACCES,                 "Permission denied" },

  { WSAEFAULT,                 "Bad address" },

  { WSAEINVAL,                 "Invalid argument" },

  { WSAEMFILE,                 "Too many open sockets" },

  { WSAEWOULDBLOCK,            "Operation would block" },

  { WSAEINPROGRESS,            "Operation now in progress" },

  { WSAEALREADY,               "Operation already in progress" },

  { WSAENOTSOCK,               "Socket operation on non-socket" },
  { WSAEDESTADDRREQ,       "Destination address required" },
  { WSAEMSGSIZE,           "Message too long" },
  { WSAEPROTOTYPE,         "Protocol wrong type for socket" },
  { WSAENOPROTOOPT,        "Bad protocol option" },
  { WSAEPROTONOSUPPORT,    "Protocol not supported" },
  { WSAESOCKTNOSUPPORT,    "Socket type not supported" },
  { WSAEOPNOTSUPP,         "Operation not supported on socket" },
  { WSAEPFNOSUPPORT,       "Protocol family not supported" },
  { WSAEAFNOSUPPORT,       "Address family not supported" },
  { WSAEADDRINUSE,         "Address already in use" },
  { WSAEADDRNOTAVAIL,      "Can't assign requested address" },
  { WSAENETDOWN,           "Network is down" },
  { WSAENETUNREACH,        "Network is unreachable" },
  { WSAENETRESET,          "Net connection reset" },
  { WSAECONNABORTED,       "Software caused connection abort" },
  { WSAECONNRESET,         "Connection reset by peer" },
  { WSAENOBUFS,            "No buffer space available" },
  { WSAEISCONN,            "Socket is already connected" },
  { WSAENOTCONN,           "Socket is not connected" },
  { WSAESHUTDOWN,          "Can't send after socket shutdown" },
  { WSAETOOMANYREFS,       "Too many references, can't splice" },
  { WSAETIMEDOUT,          "Connection timed out" },
  { WSAECONNREFUSED,       "Connection refused" },
  { WSAELOOP,              "Too many levels of symbolic links" },
  { WSAENAMETOOLONG,       "File name too long" },
  { WSAEHOSTDOWN,          "Host is down" },
  { WSAEHOSTUNREACH,       "No route to host" },
  { WSAENOTEMPTY,          "Directory not empty" },
  { WSAEPROCLIM,           "Too many processes" },
  { WSAEUSERS,             "Too many users" },
  { WSAEDQUOT,             "Disc quota exceeded" },
  { WSAESTALE,             "Stale NFS file handle" },
  { WSAEREMOTE,            "Too many levels of remote in path" },
  { WSASYSNOTREADY,        "Network subsystem is unavailable" },
  { WSAVERNOTSUPPORTED,    "Winsock version not supported" },
  { WSANOTINITIALISED,     "Winsock not yet initialized" },
  { WSAHOST_NOT_FOUND,     "Host not found" },
  { WSATRY_AGAIN,          "Non-authoritative host not found" },
  { WSANO_RECOVERY,        "Non-recoverable errors" },
  { WSANO_DATA,            "Valid name, no data record of requested type" },
  { WSAEDISCON,            "Graceful disconnect in progress" },
  { WSASYSCALLFAILURE,     "System call failure" },
  { WSA_NOT_ENOUGH_MEMORY, "Insufficient memory available" },
  { WSA_OPERATION_ABORTED, "Overlapped operation aborted" },
  { WSA_IO_INCOMPLETE,   "Overlapped I/O object not signalled" },
  { WSA_IO_PENDING,        "Overlapped I/O will complete later" },
  //{ WSAINVALIDPROCTABLE,   "Invalid proc. table from service provider" },
  //{ WSAINVALIDPROVIDER,    "Invalid service provider version number" },
  //{ WSAPROVIDERFAILEDINIT, "Unable to init service provider" },
  { WSA_INVALID_PARAMETER, "One or more parameters are invalid" },
  { WSA_INVALID_HANDLE,    "Event object handle not valid" }
};
const int kNumMessages = sizeof(gaErrorList) / sizeof(ErrorEntry);

// Returns a readable form of the winsock errormessage.
const char* WSAGetLastErrorMessage(const char* pcMessagePrefix)
{
  static char acErrorBuffer[256];
  ostrstream outs(acErrorBuffer, sizeof(acErrorBuffer));
  outs << pcMessagePrefix << ": ";

  int nLastError = WSAGetLastError();
  int i;
  for (i = 0; i < kNumMessages; ++i)
  {
    if (gaErrorList[i].nID == nLastError)
    {
      outs << gaErrorList[i].pcMessage;
      break;
    }
  }

  if (i == kNumMessages)
  {
    outs << "unknown error";
  }

  outs << " (" << nLastError << ")";


  outs << ends;
  acErrorBuffer[sizeof(acErrorBuffer) - 1] = '\0';
  return acErrorBuffer;

}

#elif defined(WAYK_OS_LINUX)

// Linux Include Files

#include <string.h>
#include <errno.h>
#include <stdio.h>

// Returns a readble form of the Linux network errno.

const char* strerrorcat(const char* msg)
{
  static char errorbuffer[256];
  char *ptr = errorbuffer;
  ptr += sprintf(ptr,"%s: ",msg);
  sprintf(ptr,"%s",strerror(errno));

  return errorbuffer;
}

#endif //WAYK_OS_WIN32
```

net_error.h

```
/*
 * net_error.h
 *
 * Writen by: Kyle Andrew McGrath Geske
 *
 * Created as a requirement of my Undergraduate thesis in
 * Computer Engineering for the University of Manitoba.
 *
 * Description: Header files for net_error.cpp. Defines
 * the net error logging macro for WIN32 and Linux.
 *
 * Copyright Kyle Geske 1999/2000
 *
 */

#ifndef WAYK_NET_ERROR_H
#define WAYK_NET_ERROR_H

#include "file_io.h"

#if defined(WAYK_OS_WIN32)

  const char* WSAGetLastErrorMessage(const char* pcMessagePrefix);

  #define WAYK_NET_ERROR(err) ERROR_REPORT(WSAGetLastErrorMessage(err))

#elif defined(WAYK_OS_LINUX)

  const char* strerrorcat(const char* msg);

  #define WAYK_NET_ERROR(err) ERROR_REPORT(strerrorcat(err))

#endif //WAYK_OS_?

#endif //WAYK_NET_ERROR_H
```

serial_win.cpp

```
/*
 * serial_win.cpp
 *
 * Writen by: Kyle Andrew McGrath Geske
 *
 * Created as a requirement of my Undergraduate thesis in
 * Computer Engineering for the University of Manitoba.
 *
 * Description: This file contains the WIN32 specific
 * serial communication functions
 *
 * Copyright Kyle Geske 1999/2000
 *
 */

// Include Files

#include <iostream.h>
#include <stdio.h>
#include <windows.h>

#include "serial.h"

// Local Static Variables

static HANDLE comm;
static COMMTIMEOUTS old_Timeouts;

// Functions

void Serial_Init(const char* comport, const char* settings)
{
  comm =
CreateFile(comport,GENERIC_READ|GENERIC_WRITE,0,NULL,OPEN_EXISTING,FILE_ATTRIBU
TE_NORMAL,0);
  if (comm == INVALID_HANDLE_VALUE)
  {
    cout << "Error: Cannot open " << comport << "." << endl;
  }
  else // Configure the port
  {
    DCB dcb;
    COMMTIMEOUTS comm_Timeouts;

    GetCommTimeouts(comm, &comm_Timeouts);

    comm_Timeouts.ReadIntervalTimeout = MAXDWORD;
    comm_Timeouts.ReadTotalTimeoutMultiplier = 0;
      comm_Timeouts.ReadTotalTimeoutConstant = 100;
    comm_Timeouts.WriteTotalTimeoutMultiplier = 0;
    comm_Timeouts.WriteTotalTimeoutConstant = 0;

    SetCommTimeouts(comm, &comm_Timeouts);

    GetCommState(comm, &dcb);
    BuildCommDCB(settings, &dcb);
    SetCommState(comm, &dcb);

  }
```

```
}

bool Send_Serial_Message(const char* message, const int len)
{
  unsigned long returnlen;
  int totalbytes = 0;

  WriteFile(comm,message,len,&returnlen,NULL);
  totalbytes += returnlen;


  if (totalbytes == len)
    return true;
  else
    return false;


}


int Receive_Serial_Message(char *message, int len)
{
  int totalbytes = 0;
  DWORD bytesread;
  char ch[10];
  bool tried = false;

  int limit = (len / 10) * 10;

  while(totalbytes < limit)
  {
    tried = true;
    ReadFile(comm, &ch, 10, &bytesread, NULL);

    if (bytesread)
       {
      for (int i = 0; i < (int)bytesread; i++)
      {
        message[totalbytes+i] = ch[i];
      }
      totalbytes += bytesread;
    }
    else
         break;
  }

  if ((totalbytes < len) && ((totalbytes != 0) || !tried))
  {
    ReadFile(comm, &ch, len - totalbytes, &bytesread, NULL);

    if (bytesread)
       {
      for (int i = 0; i < (int)bytesread; i++)
      {
        message[totalbytes+i] = ch[i];
      }
      totalbytes += bytesread;
    }
  }

  message[totalbytes] = '\0';
  return totalbytes;
```

```
}

void Serial_Cleanup()
{
  SetCommTimeouts(comm, &old_Timeouts);
  CloseHandle(comm);
}
```

serial_linux.cpp

```
/*
 * serial_linux.cpp
 *
 * Writen by: Kyle Andrew McGrath Geske
 *
 * Created as a requirement of my Undergraduate thesis in
 * Computer Engineering for the University of Manitoba.
 *
 * Description: This file contains the Linux Specific
 * serial communication functions.
 *
 * Copyright Kyle Geske 1999/2000
 *
 */

// Include Files

#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <termios.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <iostream.h>

#include "serial.h"

// Local Static Variable and Constants

static int fd;
static struct termios old_Timeouts;

// Functions

void Serial_Init(const char* comport, const char* settings)
{
  fd = open(comport, O_RDWR | O_NOCTTY);
  if (fd < 0)
  {
    cout << "Error: Could not open " << comport << endl;
  }
  else
  {

    struct termios newtio;

    tcgetattr(fd,&old_Timeouts); /* save current port settings */

    bzero(&newtio, sizeof(newtio));
    newtio.c_cflag = B4800 | CRTSCTS | CS8 | CLOCAL | CREAD;
    newtio.c_iflag = IGNPAR;
    newtio.c_oflag = 0;

     /* set input mode (non-canonical, no echo,...) */
    newtio.c_lflag = 0;

    newtio.c_cc[VTIME]    = 5;   /* inter-character timer unused */
    newtio.c_cc[VMIN]     = 10;   /* blocking read until 5 chars received */
```

```
     tcflush(fd, TCIFLUSH);
    tcsetattr(fd,TCSANOW,&newtio);
  }
}

bool Send_Serial_Message(const char* message, const int len)
{
  unsigned long returnlen;
  int totalbytes = 0;

  cout << "Seding shit" << endl;
  for (int i = 0; i < len; i++)
  {
    returnlen = write(fd,&message[i],1);
    totalbytes += returnlen;
  }

  if (totalbytes == len)
    return true;
  else
    return false;


}


int Receive_Serial_Message(char *message, int len)
{
  int totalbytes = 0;
  int bytesread;
  char ch[10];

  cout << "Recev shit" << endl;
  int limit = (len / 10) * 10;

  while(totalbytes < limit)
  {
    bytesread = read(fd, ch, 10);
    if (bytesread)
       {
      for (int i = 0; i < bytesread; i++)
      {
        message[totalbytes+i] = ch[i];
      }
      totalbytes += bytesread;
    }
    else
          break;
  }

  if (totalbytes < len-1)
  {
    bytesread = read(fd, ch, len - totalbytes);
    if (bytesread)
       {
      for (int i = 0; i < bytesread; i++)
      {
        message[totalbytes+i] = ch[i];
      }
      totalbytes += bytesread;
    }
  }
  message[totalbytes] = 0;
  return totalbytes;
}

void Serial_Cleanup()
{
  tcsetattr(fd,TCSANOW,&old_Timeouts);
}
```

serial.h

```
/*
 * serial.h
 *
 * Writen by: Kyle Andrew McGrath Geske
 *
 * Created as a requirement of my Undergraduate thesis in
 * Computer Engineering for the University of Manitoba.
 *
 * Description: Header files for serial_linux.cpp and
 * serial_win.cpp
 *
 * Copyright Kyle Geske 1999/2000
 *
 */

#ifndef WAYK_SERIAL_WIN_H
#define WAYK_SERIAL_WIN_H

void Serial_Init(const char* comport, const char* settings);
bool Send_Serial_Message(const char* message, const int len);
int Receive_Serial_Message(char *message, int len);
void Serial_Cleanup();

#endif //WAYK_SERIAL_WIN_H
```

file_io.cpp

```
/*
 * file_io.cpp
 *
 * Writen by: Kyle Andrew McGrath Geske
 *
 * Created as a requirement of my Undergraduate thesis in
 * Computer Engineering for the University of Manitoba.
 *
 * Description: This file contains functions used to deal
 * with file i/o.
 *
 * Copyright Kyle Geske 1999/2000
 *
 */

// Include File

#include <stdio.h>
#include <fstream.h>
#include <string.h>

#include "file_io.h"
#include "time.h"

// Local Static Variables

static char error_file[256];
static char log_file[256];
static char config_file[256];
static fstream error_stream;
static fstream log_stream;
static fstream config_stream;
static bool verbose;

// Functions

bool open_file(const char mode, const int file_type)
{


  switch(mode)
  {
    case 'o' : if (file_type == ERROR_FILE)
                 error_stream.open(error_file,ios::out);
               else if (file_type == LOG_FILE)
                 log_stream.open(log_file,ios::out);
               else
               {
                 cout << "Error: Cannot set this type of file to ouput" << endl;
                 return false;
               }
               break;

    case 'a' :
               if (file_type == ERROR_FILE)
               {
                 error_stream.open(error_file,ios::app);
               }
               else if (file_type == LOG_FILE)
```

```
            {
              log_stream.open(log_file,ios::app);
            }
            else
            {
              cout << "Error: Cannot set this type of file to append." << endl;
              return false;
            }
            break;
      case 'i' : if (file_type == CONFIG_FILE)
                   config_stream.open(config_file,ios::in);
            else
            {
              cout << "Error: Cannot set this type of file to input." << endl;
              return false;
            }
            break;

      default    :
            cout << "Error: Unknown file mode. File not opened." << endl;
            return false;
    }

    return true;
  }

  void close_file(const int file_type)
  {
    if (file_type == ERROR_FILE)
    {
      error_stream.close();
    }
    else if (file_type == LOG_FILE)
    {
      log_stream.close();
    }
    else if (file_type == CONFIG_FILE)
      config_stream.close();
    else
    {
      cout << "Error: Unknown file type." << endl;
    }
  }

  bool write_file(const char* message, const int file_type, bool date)
  {


    const char *timedate;
    fstream *file;

    if (file_type == ERROR_FILE)
      file = &error_stream;
    else if (file_type == LOG_FILE)
      file = &log_stream;
    else
    {
      cout << "Error: Cannot write to this type of file." << endl;
      return false;
    }

    timedate = getTime();
    if (verbose)
    {
      if (date)
      {
        cout << timedate << ": ";
      }
      cout << message;
      cout << endl;
    }

    if (date)
    {
      *file << timedate << ": ";
    }
    *file << message;
    *file << endl;

    return true;

  }

  const char* read_file(const char delim, const int file_type)
  {
    if (file_type == CONFIG_FILE)
    {
      int i = 0;
      char ch[1];
      static char message[256];
      message[0] = '\0';

      config_stream.get(ch[0]);
      while (ch[0] != delim)
      {
        if (ch[0] != 10)
          strncat(message,ch,1);
        config_stream.get(ch[0]);
        i++;
      }

      message[i] = '\0';
      return message;

    }
    else
      cout << "Error: Cannot read from this type of file." << endl;

    return "";
  }

  void set_filename(const char* filename, const int file_type)
  {
    int size = strlen(filename);

    if (size < 256)
    {
      if (file_type == ERROR_FILE)
      {
        strncpy(error_file,filename,size);
        error_file[size] = '\0';
      }
      else if (file_type == LOG_FILE)
      {
        strncpy(log_file,filename,size);
        log_file[size] = '\0';
      }
      else if (file_type == CONFIG_FILE)
      {
        strncpy(config_file,filename,size);
        error_file[size] = '\0';
      }
      else
      {
        cout << "Error: Unknown file type." << endl;
      }

    }
    else
      cout << "Error: File name could not be set. Too large";
  }

  const char* get_filename(const int file_type)
  {
    if (file_type == ERROR_FILE)
      return(error_file);
    else if (file_type == LOG_FILE)
      return(log_file);
    else if (file_type == CONFIG_FILE)
      return(config_file);
    else
    {
      cout << "Error: Unknown file type." << endl;
      return "";
    }
  }

  void set_file_verbose(const bool set)
  {
    verbose = set;
  }
```

file_io.h

```
/*
 * file_io.h
 *
 * Writen by: Kyle Andrew McGrath Geske
 *
 * Created as a requirement of my Undergraduate thesis in
 * Computer Engineering for the University of Manitoba.
 *
 * Description: Header File for file_io.cpp. Defines
 * macros for error logging.
 *
 * Copyright Kyle Geske 1999/2000
 *
 */

#ifndef WAYK_FILE_IO_H
#define WAYK_FILE_IO_H

#include <fstream.h>

#define ERROR_FILE 1
#define LOG_FILE 2
#define CONFIG_FILE 3

#define ERROR_REPORT(msg) write_file(msg,ERROR_FILE,true)
#define LOG_REPORT(msg) write_file(msg,LOG_FILE,true)

bool open_file(const char mode, const int file_type);
void close_file(const int file_type);
void set_filename(const char* filename, const int file_type);
const char* get_filename(const int file_type);
bool write_file(const char* message, const int file_type, bool date);
const char* read_file(const char delim, const int file_type);
void set_file_verbose(const bool set);

#endif //WAYK_FILE_IO_H
```

time.cpp

```
/*
 * time.cpp
 *
 * Writen by: Kyle Andrew McGrath Geske
 *
 * Created as a requirement of my Undergraduate thesis in
 * Computer Engineering for the University of Manitoba.
 *
 * Description: This file contains a function which
 * returns a nicely formatted date string.
 *
 * Copyright Kyle Geske 1999/2000
 *
 */

// Include Files

#include <time.h>
#include <stdio.h>

// Functions

const char* getTime()
{
  char *wday_name[7] = {"Sun","Mon","Tue","Wed","Thr","Fri","Sat"};
  char *mon_name[12] =
{"Jan","Feb","Mar","Apr","May","Jun","Jul","Aug","Sep","Oct","Nov","Dec"};

  static char timedate[30];
  char *ptr = timedate;

  time_t tod;
  tod = time(&tod);
  tm *timeptr = localtime(&tod);
  ptr += sprintf(ptr,"%s",wday_name[timeptr->tm_wday]);
  ptr += sprintf(ptr," %s",mon_name[timeptr->tm_mon]);
  ptr += sprintf(ptr," %d",timeptr->tm_mday);
  ptr += sprintf(ptr," %d:",timeptr->tm_hour);
  ptr += sprintf(ptr,"%d:",timeptr->tm_min);
  ptr += sprintf(ptr,"%d ",timeptr->tm_sec);
  ptr += sprintf(ptr,"%d",1900 + timeptr->tm_year);

  return timedate;
}
```

time.h

```
/*
 * time.h
 *
 * Writen by: Kyle Andrew McGrath Geske
 *
 * Created as a requirement of my Undergraduate thesis in
 * Computer Engineering for the University of Manitoba.
 *
 * Description: Header files for time.cpp
 *
 * Copyright Kyle Geske 1999/2000
 *
 */

#ifndef WAYK_TIME_H
#define WAYK_TIME_H

const char* getTime();

#endif
```

kbhit.c

```
/*
 * kbhit.c
 *
 * Writen by: Unknown programming in The Linux Journal
 *
 * Description: Allows the kbhit() function to be used
 * under linux.
 *
 */

#include  <termios.h>
#include  <signal.h>
#include  <stdlib.h>
#include  <stdio.h>
#include  <unistd.h>
#include  <sys/time.h>
#include  <sys/types.h>


static struct termios  current,  /* new terminal settings             */
                       initial;  /* initial state for restoring later */


/* ------------------------------------------------------------------------ */


/* Restore term-settings to those saved when term_init was called */

void  term_restore  (void)  {
  tcsetattr(0, TCSANOW, &initial);
}  /* term_restore */


/* ------------------------------------------------------------------------ */


/* Clean up terminal; called on exit */

void  term_exit  ()  {
  term_restore();
}  /* term_exit */


/* ------------------------------------------------------------------------ */


/* Will be called when ctl-z is pressed, this correctly handles the terminal */

void  term_ctrlz  ()  {
  signal(SIGTSTP, term_ctrlz);
  term_restore();
  kill(getpid(), SIGSTOP);
}  /* term_ctrlz */


/* ------------------------------------------------------------------------ */


/* Will be called when application is continued after having been stopped */
```

```
void  term_cont  ()  {
  signal(SIGCONT, term_cont);
  tcsetattr(0, TCSANOW, &current);
}  /* term_cont */


/* ------------------------------------------------------------------------ */

void  term_init  (void)  {
  /* if stdin isn't a terminal this fails. But    */
  /* then so does tcsetattr so it doesn't matter. */

  tcgetattr(0, &initial);
  current = initial;                       /* Save a copy to work with later  */
  signal(SIGINT,  term_exit);              /* We _must_ clean up when we exit */
  signal(SIGQUIT, term_exit);
  signal(SIGTSTP, term_ctrlz);             /* Ctrl-Z must also be handled     */
  signal(SIGCONT, term_cont);
  atexit(term_exit);
}  /* term_init */


/* ------------------------------------------------------------------------ */


/* Set character-by-character input mode */

void  term_character  (void)  {
  /* One or more characters are sufficient to cause a read to return */
  current.c_cc[VMIN]   = 1;
  current.c_cc[VTIME]  = 0;  /* No timeout; read forever until ready */
  current.c_lflag     &= ~ICANON;           /* Line-by-line mode off */
  tcsetattr(0, TCSANOW, &current);
}  /* term_character */


/* ------------------------------------------------------------------------ */


/* Return to line-by-line input mode */

void  term_line  (void)  {
  current.c_cc[VEOF]  = 4;
  current.c_lflag    |= ICANON;
  tcsetattr(0, TCSANOW, &current);
}  /* term_line */


/* ------------------------------------------------------------------------ */

/* Key pressed ? */

int  kbhit  (void)  {
  struct timeval  tv;
  fd_set          read_fd;


  /* Do not wait at all, not even a microsecond */
  tv.tv_sec  = 0;
  tv.tv_usec = 0;

  /* Must be done first to initialize read_fd */
  FD_ZERO(&read_fd);

  /* Makes select() ask if input is ready;  0 is file descriptor for stdin */
  FD_SET(0, &read_fd);

  /* The first parameter is the number of the largest fd to check + 1 */
  if  (select(1, &read_fd,
                 NULL,          /* No writes        */
                 NULL,          /* No exceptions    */
                 &tv)  == -1)
    return(0);                  /* an error occured */

  /* read_fd now holds a bit map of files that are readable. */
  /* We test the entry for the standard input (file 0).      */
  return(FD_ISSET(0, &read_fd)  ?  1  :  0);
}  /* kbhit */
```

kbhit.h

```c
/*
 * kbhit.h
 *
 * Writen by: Unknown programming in The Linux Journal
 *
 *
 * Description: Header file for kbhit.c
 *
 *
 */

#include  <termios.h>
#include  <signal.h>
#include  <stdlib.h>
#include  <stdio.h>
#include  <unistd.h>
#include  <sys/time.h>
#include  <sys/types.h>

void  term_restore  (void);
void  term_exit  ();
void  term_ctrlz  ();
void  term_cont  ();
void  term_init  (void);
void  term_character  (void);
void  term_line  (void);
int  kbhit  (void);
```

data.cfg

```
// Client Configuration File

ERRORFILE!error.log!

LOGFILE!ops.log!

COMPORT!Com1!

COMSET!baud=4800 parity=O data=8 stop=1!

PVTIME!0000!

LNTIME!0150!

IDNUM!1001!

SERVERIP!207.161.231.161!

SERVERPORT!31337!

END!END!
```

Appendix C - Server Source Code (Java)

GPSserver.java

```java
/* GPSserver.java
 *
 * Written by: Kyle Andrew McGrath Geske
 *
 * Created as a requirement of my undergraduate thesis in
 * Computer Engineering for the University of Manitoba.
 *
 * Description: The main program class which spawns the
 * server threads and the front-end GUI.
 *
 * Copyright Kyle Geske 1999/2000
 *
 */

import java.io.*;
import java.net.*;

public class GPSserver
{
  public static void main(String args[])
  {

     location_file locationFile = new location_file("locations.txt");
     history_file historyFile = new history_file("history.txt",locationFile);
     historyFile.initHistory();
     locationFile.initLocations();

     status_vector exceptions = new status_vector();

     status_vector webStatus = new status_vector();
     server_thread web_thread = new
server_thread(80,locationFile,historyFile,webStatus,exceptions,1);
     web_thread.start();
     status_vector dataStatus = new status_vector();
     server_thread data_thread = new
server_thread(31337,locationFile,historyFile,dataStatus,exceptions,2);
     data_thread.start();
     new gps_UI(locationFile,historyFile,webStatus,dataStatus,exceptions);
  }
}
```

gps_coords.java

```java
/* gps_coords.java
 *
 * Written by: Kyle Andrew McGrath Geske
 *
 * Created as a requirement of my undergraduate thesis in
 * Computer Engineering for the University of Manitoba.
 *
 * Description: A class created to handle longitude/latitude
 * co-ordinates. Includes a helper method to calculate the
 * distance between two longitude/latitude co-ordinate sets.
 *
 * Copyright Kyle Geske 1999/2000
 *
 */

import java.io.*;

public class gps_coords
{
  private double longitude;
  private double latitude;

  private char eastWest; // For Longitude
  private char northSouth; // For Latitude
  private int longDegree;
  private int longMinute;
  private int longSecond;

  private  int latDegree;
  private int latMinute;
  private int latSecond;

  private String timeOrLocation;

  /*
  gps_coords(double lon, double lat)

  This constructor is to be used with decimal degress

  eg: -49.1234,97.4321

  */

  public gps_coords(double lat, double lon, String timeOrLocation)
  {

    this.timeOrLocation = timeOrLocation;

    longitude = lon;
    latitude = lat;

    if (latitude > 0)
      northSouth = 'N';
    else
    {
      northSouth = 'S';
      latitude *= -1;
    }
    if (longitude > 0)
      eastWest = 'E';
    else
    {
      eastWest = 'W';
      longitude *= -1;
    }


    int coords[] = new int[3];

    convert(longitude,coords);
    longDegree = coords[0];
    longMinute = coords[1];
    longSecond = coords[2];


    convert(latitude,coords);
    latDegree = coords[0];
    latMinute = coords[1];
    latSecond = coords[2];


  }

  /*
  gps_coords(int latDeg, int latMin, int latSec, char ns, int longDeg, int
longMin, int longSec, char ew)

  This constructor is to be used when the gps coordinates have been split into
  degrees, minutes, seconds and direction.

  */

  public gps_coords(int latDeg, int latMin, int latSec, char ns, int longDeg,
int longMin, int longSec, char ew, String timeOrLocation)
  {

    this.timeOrLocation = timeOrLocation;

    eastWest = ew;
    northSouth = ns;

    longDegree = longDeg;
    longMinute = longMin;
    longSecond = longSec;

    latDegree = latDeg;
    latMinute = latMin;
    latSecond = latSec;

    longitude = longDegree + ((double)longMinute + (double)longSecond/60)/60;
    latitude = latDegree + ((double)latMinute + (double)latSecond/60)/60;

    longitude = (ns == 'S')? longitude : -longitude;
    latitude = (ew == 'W')? latitude : -latitude;


  }

  /*
  gps_coords(String latlong)
```

```
This constructor is to be used with a formatted GPS string.

eg: 36°66'6"S 76°6'66"W

*/

public gps_coords(String latlong)
{

  int loc = latlong.indexOf('°');
  latDegree = Integer.parseInt(latlong.substring(0,loc));
  int loc2 = latlong.indexOf("'");
  latMinute = Integer.parseInt(latlong.substring(loc+1,loc2));
  loc = latlong.indexOf((char)34);
  latSecond = Integer.parseInt(latlong.substring(loc2+1,loc));
  northSouth = latlong.charAt(loc+1);
  loc = latlong.indexOf(' ');
  loc2 = latlong.indexOf('°',loc);
  longDegree = Integer.parseInt(latlong.substring(loc+1,loc2));
  loc = latlong.indexOf("'",loc2);
  longMinute = Integer.parseInt(latlong.substring(loc2+1,loc));
  loc2 = latlong.indexOf((char)34,loc);
  longSecond = Integer.parseInt(latlong.substring(loc+1,loc2));
  eastWest = latlong.charAt(loc2+1);

  timeOrLocation = latlong.substring(loc2+3,latlong.length());

  longitude = longDegree + ((double)longMinute + (double)longSecond/60)/60;
  latitude = latDegree + ((double)latMinute + (double)latSecond/60)/60;

}

/*
public gps_coords(String lat, String lon, String timeOrLocation)

This constructor is to be used with latitude and longitude strings
provided by the ACEII receiver.

eg.
lat = SBBCCCCC
lon = SDDDEEEEE

S = +/-

Decimal place added between B&C and D&E.

*/
public gps_coords(String lat, String lon, String timeOrLocation)
{
  this.timeOrLocation = timeOrLocation;

  if (lat.charAt(0) == '+')
    northSouth = 'N';
  else
    northSouth = 'S';
  if (lon.charAt(0) == '+')
    eastWest = 'E';
  else
    eastWest = 'W';

  latitude = (double)Integer.parseInt(lat.substring(1,8))/100000;
  longitude = (double)Integer.parseInt(lon.substring(1,9))/100000;
  int coords[] = new int[3];

  convert(longitude,coords);
  longDegree = coords[0];
  longMinute = coords[1];
  longSecond = coords[2];


  convert(latitude,coords);
  latDegree = coords[0];
  latMinute = coords[1];
  latSecond = coords[2];

}


/*
convert(double value, int coords[])

This method takes in a longitude or latitude in decimal degrees
and converts it places degree, minute and second information
in the coords array. The second information is decimally truncated
since the GPS reciever has a accuracy of 100m 95% of the time and
one second is equal to approximatly 30 meters at the equator.

*/

private void convert(double value,int coords[])
{
  coords[0] = (int)value;
  double tempMin = value - (float)coords[0];
  tempMin *= 60;
  coords[1] = (int)tempMin;
  double tempSec = tempMin - (float)coords[1];
  tempSec *= 60;
  coords[2] = (int)tempSec;

}

public double getLongitude()
{
  return this.longitude;
}

public double getLatitude()
{
  return this.latitude;
}

/*
double distance(gps_coords coord1, gps_coords coord2)

A helper method used to calculate the distance in meteres
between to longitude/latitude co-ordinate sets.

*/
public static final double distance(gps_coords coord1, gps_coords coord2)
{
  double lat1 = Math.toRadians(coord1.getLatitude());
  double lat2 = Math.toRadians(coord2.getLatitude());
  double lon1 = Math.toRadians(coord1.getLongitude());
  double lon2 = Math.toRadians(coord2.getLongitude());
```

```
    double d = 2*Math.asin(Math.sqrt(Math.pow((Math.sin((lat1-lat2)/2)),2) +
Math.cos(lat1)*Math.cos(lat2)*Math.pow((Math.sin((lon1-lon2)/2)),2)));

    d = Math.abs(Math.toDegrees(d) * 111132.95);

    return d;
  }

  public String toString()
  {
    return this.toString(false);
  }

  public String toString(boolean descriptions)
  {
    String tempString = "";

    if (descriptions)
    {
      tempString = "Latitude: " + latDegree + "°" + latMinute + "'" +
latSecond + (char)34 + northSouth + " ";
      tempString += "Longitude: " + longDegree + "°" + longMinute + "'" +
longSecond + (char)34 + eastWest + " " + timeOrLocation;
    }
    else
    {
      tempString = latDegree + "°" + latMinute + "'" + latSecond + (char)34 +
northSouth + " ";
      tempString += longDegree + "°" + longMinute + "'" + longSecond +
(char)34 + eastWest + " "  + timeOrLocation;
    }

    return tempString;
  }

}
```

gps_file.java

```
/* gps_file.java
 *
 * Written by: Kyle Andrew McGrath Geske
 *
 * Created as a requirement of my undergraduate thesis in
 * Computer Engineering for the University of Manitoba.
 *
 * Description: A base class to be used for file
 * manipulation. This class is further extended by
 * the history_file, html_file and location_file
 * classes.
 *
 * Copyright Kyle Geske 1999/2000
 *
 */

import java.io.*;

public class gps_file
{
  String fileName;
  Lock fileLock = new Lock();

  public gps_file(String fileName)
  {
    this.fileName = fileName;

    FileInputStream fs = null;
    int fileLength = 0;

    try
    {
      fs = new FileInputStream(fileName);
    }
    catch (FileNotFoundException fnfe)
    {
      try
      {
        FileOutputStream temp =  new FileOutputStream(fileName);
        temp.close();
        fs = new FileInputStream(fileName);
        fileLength = fs.available();
      }
      catch (Exception e)
      {
        System.out.println("Error: Could not find or open data file: " + e +
"\n");
        System.exit(1);
      }
    }

    try
    {
      fs.close();
    }
    catch (Exception e)
    {
      System.out.println("Error: Could not close file: " + e + "\n");
    }
  } // end gps_file constructor

  public String read()
  {
    String readData = "";

    // lock readers
   fileLock.lock();
    try
    {
      FileInputStream fs = new FileInputStream(fileName);
      int fslength = fs.available();

      for (int i = 0; i < fslength; i++)
      {
        char ch = (char)fs.read();

        if (ch == '\n')
          readData += "!";
        else
          readData += ch;
      }
      readData += "!";
      fs.close();
    }
    catch (Exception e)
    {
      System.out.println("Error: Could not read from data file: " + e + "\n");

    }

   fileLock.releaseLock();

   return readData;
  }

  public void write(String gpsEntry,boolean append)
  {
    fileLock.lock();
    try
    {
      FileOutputStream fs = new FileOutputStream(fileName,append);
      for (int i = 0; i < gpsEntry.length(); i++)
      {
        fs.write((int)gpsEntry.charAt(i));
      }
      fs.flush();
      fs.close();

    }
    catch (Exception e)
    {
      System.out.println("Error: Could not write to data file: " + e + "\n");
    }
    fileLock.releaseLock();
  }

}
```

history_file.java

```
/* history_file.java
 *
 * Written by: Kyle Andrew McGrath Geske
 *
 * Created as a requirement of my undergraduate thesis in
 * Computer Engineering for the University of Manitoba.
 *
 * Extends: gps_file
 *
 * Description: This class is used to manipulate the
 * file which contains all uploaded gps info. To save
 * on file i/o the information is stored in a circular
 * memory buffer until the program is closed. The size
 * of the buffer is controlled by the HISTORYSIZE
 * variable.
 *
 * Copyright Kyle Geske 1999/2000
 *
 */

import java.io.*;

public class history_file extends gps_file
{
  public static final int HISTORYSIZE = 100;
  private gps_coords history[];
  private int numItems;
  private int tailPointer;
  private int headPointer;
  private boolean rolled;
  private location_file locationFile;

  public history_file(String fileName, location_file locationFile)
  {
    super(fileName);
    history = new gps_coords[HISTORYSIZE];
    headPointer = 0;
    tailPointer = 0;
    numItems = 0;
    rolled = false;
    this.locationFile = locationFile;
  }

  public void initHistory()
  {
    String hisString = read();
    if (hisString.length() <= 1); // Do nothing.
    else
    {
      int loc = 0,loc2 = 0;
      while(true)
      {
        loc = loc2;
        loc2 = hisString.indexOf("!",loc);
        if (loc2 == -1)
          break;
        addCoord(new gps_coords(hisString.substring(loc,loc2)));
        loc2++;
      }
    }
  } // init_history

  public void addCoord(gps_coords newCoord)
  {
    history[tailPointer] = newCoord;
    tailPointer++;
    if (tailPointer == HISTORYSIZE)
    {
      tailPointer = 0;
    }
    if (numItems < HISTORYSIZE)
      numItems++;
    if (tailPointer == headPointer)
    {
      headPointer++;
      rolled = true;
      if (headPointer == HISTORYSIZE)
        headPointer = 0;
    }
  }

  public void dumpHistory()
  {
    String crLf = "\n";
    String text = "";
    int index = 0;
    write("",false);

    try
    {
      for (int i=0; i < numItems; i++)
      {
        index = headPointer + i;
        if (rolled)
          index--;
        if (index > HISTORYSIZE -1)
          index -= HISTORYSIZE;
        if (i == numItems - 1)
          crLf = "";
        if (index >= 0)
        {
          text += history[index];
        }
        if (index < 0)
        {
          text += history[HISTORYSIZE-1];
        }

        text += crLf;

      }
      write(text,false);
    }
    catch(Exception e)
    {
      System.out.println("Could not dump history["+index+"] : " + e);
    }
  }

  public String toString()
  {
    return this.toString(false);
  }

  public String toString(boolean html)
  {
    String crLf = "\n";
    String text = "";
    int index = 0;

    try
    {
      for (int i=0; i < numItems; i++)
      {
        index = headPointer + i;
        if (rolled)
          index--;
        if (index > HISTORYSIZE -1)
          index -= HISTORYSIZE;
        if (i == numItems - 1)
          crLf = "";
        if (index >= 0)
        {
          text += locationFile.knownLocation(history[index],html) + crLf;
        }
        if (index < 0)
        {
          text += locationFile.knownLocation(history[HISTORYSIZE -1],html) +
crLf;
        }
      }
    }
    catch (Exception e)
    {
      System.out.println("Could not read history["+index+"] : " + e);
    }
    finally
    {
      return text;
    }

  }

}
```

location_file.java

```
/* location_file.java
 *
 * Written by: Kyle Andrew McGrath Geske
 *
 * Created as a requirement of my Undergraduate thesis in
 * Computer Engineering for the University of Manitoba.
 *
 * Extends: gps_file
 *
 * Description: This class is used to manipulate
 * the file which contains all the uploaded location
 * data info. To save on file i/o the information is
 * sroted in a memory vector during program operation.
 *
 * Copyright Kyle Geske 1999/2000
 *
 */

import java.io.*;
import java.util.*;

public class location_file extends gps_file
{
  private Vector locations;

  public location_file(String fileName)
  {
    super(fileName);
    locations = new Vector();
  }

  public void initLocations()
  {
    String locString = read();
    int loc = 0,loc2 =0;
    while(true)
    {
      loc = loc2;
      loc2 = locString.indexOf("!",loc);
      if (loc2 == -1)
        break;
      locations.addElement(new gps_coords(locString.substring(loc,loc2)));
      loc2++;
      loc = loc2;
      loc2 = locString.indexOf("!",loc);
      if (loc2 == -1)
        break;
      locations.addElement(locString.substring(loc,loc2));
      loc2++;
              loc = loc2;
    }
  }

  public void addLocation(gps_coords gpsCoords, String distance)
  {
    locations.addElement(gpsCoords);
    locations.addElement(distance);
  }
```

```java
  public String knownLocation(gps_coords gpsCoords, boolean html)
  {

    String returnString = "";
    int loc;

    for(int i=0; i< locations.size(); i+=2)
    {
      gps_coords tempCoords = (gps_coords)locations.elementAt(i);
      double d1 = gps_coords.distance(tempCoords,gpsCoords);
      double d2 = Double.parseDouble((String)locations.elementAt(i+1));
      if (d1 <= d2)
      {
        String string2 = tempCoords.toString(false);
        loc = string2.indexOf(' ');
        loc = string2.indexOf(' ',loc+1);
        String location = string2.substring(loc+1,string2.length());
        returnString += location + " : ";
      }
    }
    if (html)
      returnString = "<B>" + returnString + "</B>";
    return returnString + gpsCoords.toString();
  }

  public void dumpLocations()
  {
    String crLf = "\n";
    write("",false);
    for(int i=0;i < locations.size();i++)
    {
      write((gps_coords)locations.elementAt(i) + crLf,true);
      if (i == locations.size() -2)
        crLf = "";
      write((String)locations.elementAt(++i) + crLf,true);
    }
  }

  public String toString()
  {
    return locations.toString();
  }

}
```

html_file.java

```java
/* html_file.java
 *
 * Written by: Kyle Andrew McGrath Geske
 *
 * Created as a requirement of my Undergraduate thesis in
 * Computer Engineering for the University of Manitoba.
 *
 * Extends: gps_file
 *
 * Description: This class is used to manipulate
 * the html files displayed to the user.
 *
 * Copyright Kyle Geske 1999/2000
 *
 */

import java.io.*;

public class html_file extends gps_file
{
  html_file(String filename)
  {
    super(filename);
  }

  public String read()
  {
    return read("No Such User","No Such User");
  }

  public String read(String titleString, String dataString)
  {
    String readData = "";

    // lock readers
   fileLock.lock();
    try
    {
      FileInputStream fs = new FileInputStream(fileName);
      int fslength = fs.available();
      int i = 0;

      while (i < fslength)
      {
        String line = readln(fs);

        if (line.startsWith("<ADDTITLE>"))
          readData += "<TITLE>"+titleString+"</TITLE>";
        else if (line.startsWith("<ADDINFO>"))
          readData += htmlize(dataString);
        else
          readData += line;
        i += line.length() + 1;
      }
      fs.close();
    }
    catch (Exception e)
    {
        System.out.println("Error: Could not read from data file: " + e + "\n");

      }

     fileLock.releaseLock();

      return readData;
   }

   private String readln(FileInputStream fs)
   {
      String readData = "";
      char ch = ' ';
      while(true)
      {

         try
         {
            ch = (char)fs.read();
         }
         catch (Exception e)
         {
            System.out.println("Could not read from html file: " + e);
         }
         if (ch != '\n')
            readData += ch;
         else
            break;
      }

      return readData;
   }

   private String htmlize(String textData)
   {
      String htmlData = "";
      for (int i = 0; i < textData.length(); i++)
      {
         if (textData.charAt(i) == '\n')
            htmlData += "<BR>" + "\n";
         else
            htmlData += textData.charAt(i);
      }

      return htmlData;
   }

}
```

Lock.java

```
/* Lock.java
 *
 * Written by: Christopher McGee
 *
 * http://journal.iftech.com/articles/threadsync/default.asp
 *
 * Description: This class can be used to control
 * simultaneous access to java resources.
 *
 */

class Lock extends Object
{
     private boolean m_bLocked = false;

     public synchronized void lock()
     {
          // if some other thread locked this object then we need to wait
          // until they release the lock
          if( m_bLocked )
          {
               do
               {
                    try
                    {
                         // this releases the synchronized that we are in
                         // then waits for a notify to be called in this object
                         // then does a synchronized again before continuing
                         wait();
                    }
                    catch( InterruptedException e )
                    {
                         e.printStackTrace();
                    }
                    catch( Exception e )
                    {
                         e.printStackTrace();
                    }
               } while( m_bLocked );     // we can't leave until we got the
lock, which
                                             // we may not have got if an
exception occured
          }

          m_bLocked = true;
     }

     public synchronized boolean lock( long milliSeconds )
     {
          if( m_bLocked )
          {
               try
               {
                    wait( milliSeconds );
               }
               catch( InterruptedException e )
               {
                    e.printStackTrace();
               }
```

```
                if( m_bLocked )
                {
                    return false;
                }
            }

            m_bLocked = true;
            return true;
        }

        public synchronized boolean lock( long milliSeconds, int nanoSeconds )
        {
            if( m_bLocked )
            {
                try
                {
                    wait( milliSeconds, nanoSeconds );
                }
                catch( InterruptedException e )
                {
                    e.printStackTrace();
                }

                if( m_bLocked )
                {
                    return false;
                }
            }

            m_bLocked = true;
            return true;
        }

        public synchronized void releaseLock()
        {
            if( m_bLocked )
            {
                m_bLocked = false;
                notify();
            }
        }

        public synchronized boolean isLocked()
        {
            return m_bLocked;
        }
    }
```

server_thread.java

```
/* server_thread.java
 *
 * Writen by: Kyle Andrew McGrath Geske
 *
 * Created as a requirement of my Undergraduate thesis in
 * Computer Engineering for the University of Manitoba.
 *
 * Description: This thread handles the incoming data
 * connections from the GPS client or from a web
 * browser. Once the connection is established, the
 * thread passes the socket to an instance of the
 * data_sever or web_server class.
 *
 * Copyright Kyle Geske 1999/2000
 *
 */

import java.io.*;
import java.net.*;

public class server_thread extends Thread
{
  int port;
  int type;
  Socket sock;
  ServerSocket servsock;
  location_file locFile;
  history_file hisFile;
  status_vector statusVector,exceptVector;

  public server_thread(int port,location_file locFile,history_file
hisFile,status_vector statusVector,status_vector exceptVector, int type)
  {
    this.type = type;
    this.port = port;
    this.locFile = locFile;
    this.hisFile = hisFile;
    this.statusVector = statusVector;
    this.exceptVector = exceptVector;
  }

  public void run()
  {
    try
    {
      servsock = new ServerSocket(port);
    }
    catch (Exception e)
    {
      exceptVector.addItem(time.getTime() + ": Could not create ServerSocket:
" + e);
    }


    while(true)
    {
      try
      {
        sock = servsock.accept();
```

```
          if (type == 1) // ie a web server
          {
            web_server my_server = new
web_server(sock,locFile,hisFile,statusVector,exceptVector);
            my_server.start();
          }
          else if (type == 2) // ie a data server
          {
            data_server my_server = new
data_server(sock,locFile,hisFile,statusVector,exceptVector);
            my_server.start();
          }
          else
          {
            System.out.println("Error: This server type does not exist");
          }

        }
        catch (Exception e)
        {
          exceptVector.addItem(time.getTime() + ": Could not spawn new server: "
+ e + "\n");

        }
      }

    }
}
```

web_server.java

```
/* web_thread.java
 *
 * Writen by: Kyle Andrew McGrath Geske
 *
 * Created as a requirement of my Undergraduate thesis in
 * Computer Engineering for the University of Manitoba.
 *
 * Description: This thread handles the incoming
 * HTTP web requests by serving the gps data
 * as a web page.
 *
 * Copyright Kyle Geske 1999/2000
 *
 */

import java.io.*;
import java.net.*;

public class web_server extends Thread
{

  private Socket sock;
  private PrintWriter out;
  private InputStreamReader inr;
  private BufferedReader in;
  private location_file locFile;
  private history_file hisFile;
  private status_vector webStatus,exceptStatus;
  private html_file htmlFile;

  public web_server(Socket sock,location_file locFile,history_file
hisFile,status_vector webStatus,status_vector exceptStatus)
  {
    this.sock = sock;
    this.locFile = locFile;
    this.hisFile = hisFile;
    this.webStatus = webStatus;
    this.exceptStatus = exceptStatus;
    htmlFile = new html_file("userhtml.txt");
    try
    {
      inr = new InputStreamReader(sock.getInputStream());
      in = new BufferedReader(inr);
      out = new PrintWriter(sock.getOutputStream(),false);
    }
    catch (Exception e)
    {
      exceptStatus.addItem(time.getTime() + ": Could not create web_server: "
+ e);
    }
  }

  private void httpHeader()
  {
    out.println("HTTP/1.1 200 OK");
    out.println("Server: Way-K/0.1 (Java)");
    out.println("Connection: close");
    out.println("Content-Type: text/html");
    out.println("");
  }



  private String isolateRequest(String theRequest)
  {
      String request = theRequest;

      int first = request.indexOf(' ') + 1; // First location of a space
      int last = request.lastIndexOf(' ');  // Last location of a space
      request = request.substring(first,last); // Isolate the request.

      if (request.length() != 1) // If it's not a root request strip the '/'
      {
        last = request.length();

        if (request.endsWith("/"))
        {
          last = request.lastIndexOf('/');
        }
        request = request.substring(1,last);
      }

    return request;
  }

  public void run()
  {
    try
    {

      String line = in.readLine();

      line = isolateRequest(line);

      httpHeader();

      boolean asking = true; // Is the user still sending the request
      boolean once = true; // Has enter been pressed once.

      while(asking) //Read in the web browsers request string. We do not use
this information.
      {
        char ch = (char)in.read();
        if (ch == '\r')
        {
          // System.out.println("");
          if (once)
          {
            asking = false;
          }
          else
          {
            once = true;
          }

        }
        else if (ch == '\n'); // do nothing.
        else
        {
          once = false;
        }
        }
        out.print(htmlFile.read(line + " GPS history",hisFile.toString(true)));
        out.flush();
        webStatus.addItem(time.getTime() + ": Served a web page: " +
sock.getInetAddress());
      }
      catch(Exception e)
      {
        exceptStatus.addItem(time.getTime() + ": Could not serve web page: " + e
+ "\n"); // Remove l8r.
      }
      finally
      {
        try
        {
          sock.close();
        }
        catch (Exception e)
        {
          // Do nothing. Sometime the socket will already be peer closed.
        }
      }

    }
}
```

data_server.java

```
/* data_server.java
 *
 * Written by: Kyle Andrew McGrath Geske
 *
 * Created as a requirement of my undergraduate thesis in
 * Computer Engineering for the University of Manitoba.
 *
 * Description: This thread parses incoming data and
 * adds information to the history and location files.
 *
 * Copyright Kyle Geske 1999/2000
 *
 */

import java.io.*;
import java.net.*;

public class data_server extends Thread
{
  private final static String password = "letmein"; // HAHA! Quite a difficult
password!
  private Socket sock;
  private PrintWriter out;
  private InputStreamReader inr;
  private BufferedReader in;
  private location_file locFile;
  private history_file hisFile;
  private status_vector dataStatus, exceptStatus;

  public data_server(Socket sock,location_file locFile,history_file
hisFile,status_vector dataStatus,status_vector exceptStatus)
  {
    this.sock = sock;
    this.locFile = locFile;
    this.hisFile = hisFile;
    this.dataStatus = dataStatus;
    this.exceptStatus = exceptStatus;
    try
    {
      inr = new InputStreamReader(sock.getInputStream());
      in = new BufferedReader(inr);
      out = new PrintWriter(sock.getOutputStream(),false);
    }
    catch (Exception e)
    {
      exceptStatus.addItem(time.getTime() + ": Could not create data_server: "
+ e);
    }

  }

  /*
  void run()

  If the password received is vailid a string of the following structure will
be parse.

  ##ACCCDDDDDEEEEFFFFF{G}
  ##L+4912345+09754321Pizza Place@500

  A is message type. 'H' = history log  'L' = location log
  CCCDDDDD = latitude
  EEEEFFFFF = longitude
  {G} is the history time for message type 'H' and location for message type
'L'
  If message is of Type L the location will also include a radius.
Location@Radius

  */

  public void run()
  {
    try
    {
      String ip = sock.getInetAddress().toString();
      String line = in.readLine();
      if (line.equals(password))
      {
        line = in.readLine();
        if (line.charAt(0) == '#')
        {
          String latitude = line.substring(3,11);
          String longitude = line.substring(11,20);
          String timeOrLocation = line.substring(20,line.length());
          String radius = "";
          int loc = timeOrLocation.indexOf("@");
          if (loc != -1)
          {
            radius = timeOrLocation.substring(loc+1,timeOrLocation.length());
            timeOrLocation = timeOrLocation.substring(0,loc);
          }
          gps_coords myCoords = new
gps_coords(latitude,longitude,timeOrLocation);
          if (line.charAt(2) == 'H')
          {
            hisFile.addCoord(myCoords);
            dataStatus.addItem(time.getTime() + ": Successfully Received
History Data : " + ip);
            // dataStatus.addItem(myCoords.toString());
          }
          else if (line.charAt(2) == 'L')
          {
            locFile.addLocation(myCoords,radius);
            dataStatus.addItem(time.getTime() + ": Successfully Received
Location Data : " + ip);
            dataStatus.addItem(myCoords + " : " + radius);
          }
          else
          {
            dataStatus.addItem(time.getTime() + ": Unsuccessfully Received
Data : " + ip);
          }
        }
        else
          dataStatus.addItem(time.getTime() + ": Unsuccessfully Received Data
: " + ip);

      }
      else
      {
        dataStatus.addItem(time.getTime() + ": Incorrect Password : " + ip);
      }
    }
    catch(Exception e)
    {
      exceptStatus.addItem(time.getTime() + ": Could not download GPS data: "
+ e);
    }
    finally
    {
      try
      {
        sock.close();
      }
      catch (Exception e)
      {
        // Do nothing. Sometime the socket will already be peer closed.
      }
    }
  }
}
```

time.java

```
/* time.java
 *
 * Writen by: Kyle Andrew McGrath Geske
 *
 * Created as a requirement of my Undergraduate thesis in
 * Computer Engineering for the University of Manitoba.
 *
 * Description: A class used to fetch the current
 * time and data of the host computer.
 *
 * Copyright Kyle Geske 1999/2000
 *
 */

import java.util.Calendar;
import java.util.GregorianCalendar;

public class time
{
  private static final String dayNames = "SunMonTueWedThuFriSat";
  private static final String monthNames =
"JanFebMarAprMayJunJulAugSepOctNovDec";

  private static String dateParse(int position, boolean dayElseMonth)
  {
    String tempString = (dayElseMonth)? dayNames : monthNames;
    return(tempString.substring(position*3,(position*3)+3));
  }

  public static String getTime()
  {
    Calendar theCalendar = new GregorianCalendar();
    int seconds = theCalendar.get(Calendar.SECOND);
    int minutes = theCalendar.get(Calendar.MINUTE);
    int hours = theCalendar.get(Calendar.HOUR_OF_DAY);
    int dayOfWeek = theCalendar.get(Calendar.DAY_OF_WEEK);
    int month = theCalendar.get(Calendar.MONTH);
    int dayOfMonth = theCalendar.get(Calendar.DAY_OF_MONTH);
    int year = theCalendar.get(Calendar.YEAR);

    String sMinutes = (minutes<10)? "0" + minutes : "" + minutes;
    String sSeconds =  (seconds<10)? "0" + seconds : "" + seconds;

    return(dateParse(dayOfWeek-1,true) + " " + dateParse(month,false) + " " +
dayOfMonth + " " + hours + ":" + sMinutes + ":" + sSeconds + " " + year);

  }

}
```

status_vector.java

```java
/* status_vector.java
 *
 * Writen by: Kyle Andrew McGrath Geske
 *
 * Created as a requirement of my Undergraduate thesis in
 * Computer Engineering for the University of Manitoba.
 *
 * Description: A specialized vector to allow information
 * sharing between the program threads and the GUI.
 *
 * Copyright Kyle Geske 1999/2000
 *
 */

import java.util.*;
import java.io.*;

public class status_vector
{

  private Vector status;
  private int readPtr;
  private boolean updated;
  private boolean enable;
  private Lock vectorLock = new Lock();

  status_vector()
  {
    status = new Vector();
    readPtr = 0;
    updated = false;
    enable = true;
  }

  public void setEnabled(boolean enabled)
  {
    enable = enabled;
    this.clear();
  }

  public void clear()
  {
    vectorLock.lock();
    status.clear();
    readPtr = 0;
    updated = false;
    vectorLock.releaseLock();
  }

  public void addItem(String update)
  {
    if (enable)
    {
      vectorLock.lock();
      updated = true;
      status.addElement(update);
      vectorLock.releaseLock();
    }
  }
  public boolean changed()
  {
    return updated;
  }

  public String readUpdate()
  {
    String update = "";

    if (enable)
    {
      vectorLock.lock();
      updated = false;

      int i;
      for (i = 0; i < (status.size()-readPtr); i++)
      {
        update += (String)status.elementAt(i+readPtr) + "\n";
      }
      readPtr = readPtr + i;
      vectorLock.releaseLock();
    }

    return update;
  }

  public String toString()
  {
    return status.toString();
  }

}
```

gps_UI.java

```java
/* gps_UI.java
 *
 * Written by: Kyle Andrew McGrath Geske
 *
 * Created as a requirement of my undergraduate thesis in
 * Computer Engineering for the University of Manitoba.
 *
 * Description: The Graphical User Interface for the
 * server. Allows the user to monitor web traffic
 * as well as traffic from the GPS client application.
 *
 * Copyright Kyle Geske 1999/2000
 *
 */

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class gps_UI extends JFrame
{
  private location_file locationFile;
  private history_file historyFile;
  private status_vector webStatus,clientStatus,exceptStatus;
  private Container gpsUI;
  private JButton
disableWebButt,disableClientButt,clearWebButt,clearClientButt;
  private JTextArea
webStatusArea,dataStatusArea,exceptStatusArea,historyStatusArea;
  private Timer webTimer,clientTimer,exceptTimer,historyTimer;

  public gps_UI(location_file locationFile, history_file
historyFile,status_vector webStatus,status_vector clientStatus,status_vector
exceptStatus)
   {
     super("Way-K GPS Server");

     this.locationFile = locationFile;
     this.historyFile = historyFile;
     this.webStatus = webStatus;
     this.clientStatus = clientStatus;
     this.exceptStatus = exceptStatus;

     try
    {
     UIManager.setLookAndFeel(UIManager.getCrossPlatformLookAndFeelClassName());
    }
    catch (Exception e)
    {
     System.err.println("Could not load Java look and Feel.");
    }


    JPanel statusPanel = new JPanel();
    webStatusArea = new JTextArea("Web Server Status:\n");
    webStatusArea.setEditable(false);
    dataStatusArea = new JTextArea("Data Server Status:\n");
    dataStatusArea.setEditable(false);
    exceptStatusArea = new JTextArea("Server Exceptions:\n");
```

```
        exceptStatusArea.setEditable(false);
        historyStatusArea = new JTextArea("Server Exceptions:\n");
        historyStatusArea.setEditable(false);
         JScrollPane webScroll = new
JScrollPane(webStatusArea,JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED,JScrollPane.
HORIZONTAL_SCROLLBAR_AS_NEEDED);
     JScrollPane dataScroll = new
JScrollPane(dataStatusArea,JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED,JScrollPane
.HORIZONTAL_SCROLLBAR_AS_NEEDED);
     JScrollPane exceptScroll = new
JScrollPane(exceptStatusArea,JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED,JScrollPa
ne.HORIZONTAL_SCROLLBAR_AS_NEEDED);
     JScrollPane historyScroll = new
JScrollPane(historyStatusArea,JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED,JScrollP
ane.HORIZONTAL_SCROLLBAR_AS_NEEDED);

webScroll.setBorder(BorderFactory.createTitledBorder(BorderFactory.createEtched
Border(),"Web Server Status"));

     dataScroll.setBorder(BorderFactory.createTitledBorder(BorderFactory.createEtc
hedBorder(),"Client Server Status"));

     exceptScroll.setBorder(BorderFactory.createTitledBorder(BorderFactory.createE
tchedBorder(),"Server Exceptions Status"));

     historyScroll.setBorder(BorderFactory.createTitledBorder(BorderFactory.create
EtchedBorder(),"GPS history"));
     statusPanel.setLayout(new GridLayout(1,2));
     statusPanel.add(webScroll);
     statusPanel.add(dataScroll);

     JPanel buttonPanel = new JPanel();
     buttonPanel.setLayout(new GridLayout(1,4));

     disableWebButt = new JButton("Disable Web Logs");
     disableWebButt.addActionListener(new webLogListener());
     disableClientButt = new JButton("Disable Client Logs");
     disableClientButt.addActionListener(new clientLogListener());
     clearWebButt = new JButton("Clear Web Logs");
     clearWebButt.addActionListener(new webClearListener());
     clearClientButt = new JButton("Clear Client Logs");
     clearClientButt.addActionListener(new clientClearListener());

     buttonPanel.add(disableWebButt);
     buttonPanel.add(clearWebButt);
         buttonPanel.add(disableClientButt);
     buttonPanel.add(clearClientButt);

         JPanel mainStatusPanel = new JPanel();
         mainStatusPanel.setLayout(new GridLayout(2,1));
         mainStatusPanel.add(statusPanel);

         JPanel exceptHistPanel = new JPanel();
         exceptHistPanel.setLayout(new GridLayout(1,2));
         exceptHistPanel.add(exceptScroll);
         exceptHistPanel.add(historyScroll);

         mainStatusPanel.add(exceptHistPanel);

         JPanel centerPanel = new JPanel();
         centerPanel.setLayout(new BorderLayout());
         centerPanel.add(mainStatusPanel,BorderLayout.CENTER);
         centerPanel.add(buttonPanel,BorderLayout.SOUTH);
```

```
    gpsUI = this.getContentPane();
    gpsUI.add(centerPanel);

    webTimer = new Timer(5000, new webTimerListener());
    webTimer.setInitialDelay(0);
    webTimer.start();

    clientTimer = new Timer(5000, new clientTimerListener());
    clientTimer.setInitialDelay(0);
    clientTimer.start();

    exceptTimer = new Timer(5000, new exceptTimerListener());
    exceptTimer.setInitialDelay(0);
    exceptTimer.start();

    historyStatusArea.setText(historyFile.toString());
    historyTimer = new Timer(60000, new historyTimerListener());
    historyTimer.setInitialDelay(0);
    historyTimer.start();

    //this.pack();
    this.setSize(600,600);
    this.setResizable(true);
    this.setVisible(true);

    this.addWindowListener(new closeDown());

  }

  class webTimerListener implements ActionListener
  {
    public void actionPerformed(ActionEvent evt)
    {
      if (webStatus.changed())
      {
        webStatusArea.append(webStatus.readUpdate());
      }
    }
  }

  class clientTimerListener implements ActionListener
  {
    public void actionPerformed(ActionEvent evt)
    {
      if (clientStatus.changed())
      {
        dataStatusArea.append(clientStatus.readUpdate());
      }
    }
  }

  class exceptTimerListener implements ActionListener
  {
    public void actionPerformed(ActionEvent evt)
    {
      if (exceptStatus.changed())
      {
        exceptStatusArea.append(exceptStatus.readUpdate());
      }
    }
  }
```

```
class historyTimerListener implements ActionListener
{
  public void actionPerformed(ActionEvent evt)
  {
    historyStatusArea.setText(historyFile.toString());
  }
}

class webLogListener implements ActionListener
{
  public void actionPerformed(ActionEvent evt)
  {
    String text = disableWebButt.getText();
    if (text.equals("Disable Web Logs"))
    {
      disableWebButt.setText("Enable Web Logs");
      webStatusArea.setEnabled(false);
      webStatus.setEnabled(false);
    }
    else
    {
      disableWebButt.setText("Disable Web Logs");
      webStatusArea.setEnabled(true);
      webStatus.setEnabled(true);
    }
  }

}

class clientLogListener implements ActionListener
{
  public void actionPerformed(ActionEvent evt)
  {
      String text = disableClientButt.getText();
    if (text.equals("Disable Client Logs"))
    {
      disableClientButt.setText("Enable Client Logs");
      dataStatusArea.setEnabled(false);
      clientStatus.setEnabled(false);
    }
    else
    {
      disableClientButt.setText("Disable Client Logs");
      dataStatusArea.setEnabled(true);
      clientStatus.setEnabled(true);
    }
  }

}

class clientClearListener implements ActionListener
{
  public void actionPerformed(ActionEvent evt)
  {
    clientStatus.clear();
    dataStatusArea.setText("Data Server Status:\n");
  }
}

class webClearListener implements ActionListener
{
  public void actionPerformed(ActionEvent evt)
```

```
      {
        webStatus.clear();
        webStatusArea.setText("Web Server Status:\n");
      }
    }

    class closeDown extends WindowAdapter
    {
      public void windowClosing(WindowEvent we)
     {
      try
      {
      historyFile.dumpHistory();
        locationFile.dumpLocations();
      }
      catch(Exception e)
      {
        System.out.println("Closing " + e);
      }
      finally
      {
        System.exit(0);
      }
     }
    }

  }
```