Designing a Remote Navigation System

Jared P. Lazzaro
Colby College

Follow this and additional works at: https://digitalcommons.colby.edu/seniorscholars

Colby College theses are protected by copyright. They may be viewed or downloaded from this site for the purposes of research and scholarship. Reproduction or distribution for commercial purposes is prohibited without written permission of the author.

Recommended Citation
https://digitalcommons.colby.edu/seniorscholars/273

This Senior Scholars Paper (Open Access) is brought to you for free and open access by the Student Research at Digital Commons @ Colby. It has been accepted for inclusion in Senior Scholar Papers by an authorized administrator of Digital Commons @ Colby.
DESIGNING A REMOTE NAVIGATION SYSTEM

by

JARED P. LAZZARO

Submitted in Partial Fulfillment of the Requirements of the Senior Scholars Program

COLBY COLLEGE
2000
APPROVED:

Allen B. Downey

Randolph M. Jones

Leonard S. Reich

Fernando Q. Gouvêa
ABOUT THE AUTHOR

Jared Lazzaro grew up near the coast of Massachusetts, in the small town of Hamilton. After graduating from Hamilton-Wenham Regional High School in 1996, he came to Colby. Though relatively far from the sea, Jared focused his studies on maritime issues. His Science, Technology, and Society minor enabled him to study topics such as “transportation at sea” and “the history of navigation” as well as to obtain his USCG Master 100 Ton Near-coastal license over Jan-Plan. In the spring of 1999, Lazzaro took his semester “abroad” to Mystic, Connecticut, where he attended the Williams College–Mystic Seaport Maritime Studies Program. With this Senior Scholars project and other maritime experiences, he hopes to find a job related to maritime technology on or near the sea.
ABSTRACT

DESIGNING A REMOTE NAVIGATION SYSTEM

Jared P. Lazzaro
jplazzar@colby.edu
Colby College
7136 Mayflower Hill
Waterville, ME 04901-8871

Advisor: Allen B. Downey
abdowney@colby.edu

This project involves the design and implementation of a global electronic tracking system intended for use by trans-oceanic vessels, using the technology of the U.S. Government’s Global Positioning System (GPS) and a wireless connection to a networked computer.

Traditional navigation skills are being replaced with highly accurate electronics. GPS receivers, computers, and mobile communication are becoming common among both recreational and commercial boaters. With computers and advanced communication available throughout the maritime world, information can be shared instantaneously around the globe. This ability to monitor one’s whereabouts from afar can provide an increased level of safety and efficiency.

Current navigation software seldom includes the capability of providing up-to-the-minute navigation information for remote display. Remote access to this data will allow boat owners to track the progress of their boats, land-based organizations to monitor weather patterns and suggest course changes, and school groups to track the progress of a vessel and learn about navigation and science. The software developed in this project allows navigation information from a vessel to be remotely transmitted to a land-based server, for interpretation and deployment to remote users over the Internet. This differs from current software in that it allows the tracking of one vessel by multiple users and provides a means for two-way text messaging between users and the vessel.

Beyond the coastal coverage provided by cellular telephones, mobile communication is advancing rapidly. Current tools such as satellite telephones and single-sideband radio enable worldwide communications, including the ability to connect to the Internet. If current trends continue, portable global communication will be available at a reasonable price and Internet connections on boats will become more common.
Goals of the project are:

• To develop a system for reliable, remote access to GPS data.
• To implement the system, RemoteNav, in software.

The project uses the Java programming language for both the navigation system and the remote interface. The result is a GPS interface allowing both local and remote display of navigation information on a vessel. Other features include two-way text messaging and multiple client connections. Through the use of a proxy server, the tracking system ensures the integrity of transmitted data and provides a reliable framework for client connections to the server.

Future work on this project could include: a full implementation of the proposed design; support for tracking data from multiple vessels; design of an effective graphical user interface for viewing/interpreting navigation data; additional navigation information processing such as sunrise/sunset times; support for sending data to other devices such as autopilots.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Approval Page</td>
<td>ii</td>
</tr>
<tr>
<td>About the Author</td>
<td>iii</td>
</tr>
<tr>
<td>Abstract</td>
<td>iv</td>
</tr>
<tr>
<td>Table of Contents</td>
<td>vi</td>
</tr>
<tr>
<td>1. Background</td>
<td>1</td>
</tr>
<tr>
<td>2. Project Overview</td>
<td>6</td>
</tr>
<tr>
<td>3. Project Design</td>
<td>9</td>
</tr>
<tr>
<td>4. Development Environment</td>
<td>13</td>
</tr>
<tr>
<td>5. Software Details</td>
<td>15</td>
</tr>
<tr>
<td>6. Prototype Development</td>
<td>25</td>
</tr>
<tr>
<td>7. Conclusions and Future Work</td>
<td>27</td>
</tr>
<tr>
<td>8. The Experience</td>
<td>29</td>
</tr>
<tr>
<td>Appendix A: The NMEA Standard</td>
<td>30</td>
</tr>
<tr>
<td>Appendix B: Acknowledgements</td>
<td>34</td>
</tr>
<tr>
<td>Appendix C: Bibliography</td>
<td>35</td>
</tr>
<tr>
<td>Appendix D: Software on CD-ROM</td>
<td>36</td>
</tr>
</tbody>
</table>
1. Background

In the past hundred years, electronic navigation has grown rapidly. Since radio waves were first used to transmit wireless morse code messages around 1900, the use and application of wave theory have enabled the development of direction-finding and positioning systems. The recent trend in computer technology has greatly furthered the art of navigation. The last three decades have seen tremendous advances in the types of navigation devices available, as well as the reliability and accuracy of the information they provide. Systems such as LORAN-C, CONSOLAN, and most recently, the U. S. Government’s Global Positioning System (GPS) require sophisticated computer processing to provide useful position information.

a) Global Navigation

Land-based radio navigation systems such as LORAN (Long Range Navigation) transmit radio signals from stations along the coast. The LORAN receiver processes the signals from multiple stations to obtain a position fix. Developed in the 1940s and improved through the early 1980s, LORAN navigation proved useful when travelling within a few hundred miles of the coast. In 1978, the U.S. Department of Defense began plans for a satellite navigation with global coverage. Originally dubbed Navstar, this system is now known as the Global Positioning System (GPS).

GPS is the most recent electronic navigation system to gain mass public appeal. Today’s system employs a constellation of 24 satellites and a ground-based control station. Each satellite transmits a unique message containing its identification number, position in space, and time of transmission. A GPS receiver can compute the distance to a satellite by comparing the time a signal was transmitted to the time it was received. Using three or more satellites, a receiver can accurately compute latitude and longitude. One of the most compelling factors of GPS is that its coverage is global. The GPS constellation ensures ample coverage at any point on the globe at any time. GPS is also less prone to error than land-based radio navigation systems. Over long distances, land-based systems lose accuracy from the Earth’s curvature. Though small errors can be introduced by
propagation delay due to atmospheric effects and clock errors in the receiver, most receivers are designed to compensate for these effects and will do so quite efficiently.¹

The accuracy of GPS depends on the type of receiver. GPS was designed as a military tool, but in the early 1980s, President Ronald Reagan announced that GPS would be made available to everyone—with the exception that the best accuracy would still be reserved for the military.² With this in mind, Selective Availability (SA) was designed and implemented in all new satellites. Selective Availability provides a way for civilian and military users to access different levels of accuracy. Civilian GPS receivers are usually accurate within 65 meters. Though SA provides position data within 100 meters 95% of the time, the accuracy is usually much better. Military and specialty receivers can provide increased accuracy to within a centimeter. For most purposes, the accuracy of GPS with SA is sufficient for most navigators. As an alternative, a Differential GPS (DGPS) receiver can provide position corrections transmitted from a nearby land station. This is helpful in coastal navigation where it is often desirable to precisely know your position.

One example of the early use of GPS was during Operation Desert Storm. When faced with a shortage of military GPS receivers, regular civilian handheld units were sent to the troops. The less accurate civilian units employed in Desert Storm proved helpful nonetheless. Rumors exist that the government turned off selective availability at times, providing all GPS users with extremely precise navigation information.

As of May 1, 2000, the U.S. discontinued the use of SA. The accuracy of civilian GPS receivers can still be degraded on a regional level if necessary, but the overall accuracy of GPS is greatly increased. Instead of being able to limit accuracy to the area of a football field, users can feel confident that they’re within a tennis court’s distance from their plotted position.

In the past ten years, GPS has gained acceptance as a reliable, affordable navigation technique. At the same time as GPS was growing in

¹ Garmin Electronics. “What is GPS?”. www.garmin.com. Propagation delay is the ‘slowing down’ of the GPS signal as it passes through Earth’s ionosphere and troposphere. In space, radio signals travel at the speed of light, but they are significantly slower once they enter our atmosphere.

popularity, there was tremendous growth in other related areas of navigation technology. With computers becoming smaller and faster, many applications for interfacing with the navigation technology were developed. As electronics on board ships packed increased computing power in display units and receivers, the amount of available information increased.

b) Interfacing Navigation Devices

Electronics such as autopilots, GPS receivers, and depth sounders become more convenient and valuable when they share their data. In order to share information between electronic devices, they must use the same type of data interface. Most devices employ a talker and listener design, where one device broadcasts its information to all other connected devices. When electronics manufacturers first began implementing data interfaces in their products, many used their own designs. As a result, many types of equipment could only interface with products from the same manufacturer.

In 1980, a group of professionals from the navigation industry met to develop a standard "language" for marine interfaces. Their goal, to provide a universal interface between LORAN and autopilots, was achieved in the National Marine Electronics Association (NMEA) 0180 standard. The acceptance and use of NMEA-0180 interfaces proved that a standard could be adopted and would be followed by manufacturers. In the years since the 0180 standard, improvements, revisions, and other changes were made to allow for detailed information to be sent from a variety of different talkers. These changes are now a part of the NMEA-0183 interface standard, the most common marine data protocol in use today.

The widespread use of the NMEA-0183 standard has encouraged the design of computer software that processes navigation data from any NMEA-0183-compatible device. Computers are becoming increasingly portable as well as more powerful. It is now common for sailors and power boaters alike to bring their laptop computers on voyages so they can process, display, and log all their navigation information. Today, many boats are equipped with integrated electronics for managing all on-board systems. The NMEA interface uses the same electrical signals as the ports on the back of standard

---

computers, so it lends itself nicely to extensive software development. This, coupled with current global communication systems, allows boaters to share large amounts of detailed information from the most remote locations around the globe.

c) Global Communication

Communication at sea provides many challenges. Standard two-way radio communication is commonly limited by the power of the radio signal and the curvature of the earth. Near the shore, many vessels can take advantage of land-based wireless communication systems, such as cellular networks, shore stations, and radio broadcast towers. (See Table 1 for more information.)

From the middle of the ocean, these systems are useless. As a solution, vessels can use earth-orbiting communication satellites for their communication needs. Several constellations of satellites exist, but the most common are Inmarsat and Orbcomm. Until 1999, a handheld satellite telephone system was in use, but the privately owned company, Iridium LLC, struggled with financial problems and recently (March 17, 2000) terminated

Table 1: Current Available Wireless Communication Systems

<table>
<thead>
<tr>
<th>Type of System</th>
<th>Technology Description</th>
<th>Coverage</th>
<th>Capability</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cellular &amp; PCS</td>
<td>Modern Cellular Telephone Networks</td>
<td>Cost-effective for near-shore infrequent reporting (a few times per day)</td>
<td>Data or Voice</td>
<td>Pay for time used or amount of data transferred</td>
</tr>
<tr>
<td>Packet Data and Trunked Radio</td>
<td>Uses Two-Way Radio Technology to Interpret and Relay Data or voice.</td>
<td>Cost-effective for near-real-time reporting (a few times per hour)</td>
<td>Data or Voice</td>
<td>Depends on service provider and/or amount of data transferred</td>
</tr>
<tr>
<td>Dedicated Radio</td>
<td>Uses 2-way Radios for Point-to-Point Communication</td>
<td>Great for real-time reporting (a few times per minute)</td>
<td>Data or Voice</td>
<td>Large capital outlay for infrastructure, usage is unlimited.</td>
</tr>
<tr>
<td>Space-Based Satellite Network</td>
<td>Satellites relay information between users and a land-based station</td>
<td>Great where wide coverage is desired, expensive</td>
<td>Data or Voice (voice is only available through higher-end systems designed for larger vessels)</td>
<td>Positions appx: $0.05 per report, Messages appx: $0.01 per character.</td>
</tr>
</tbody>
</table>
all commercial service. Users of the Iridium system now must employ the services of another satellite communication provider. Despite the difficulties presented by global communication systems, and the relatively low demand for their use, the surviving communication systems such as Inmarsat and Orbcomm each offer useful services. Despite the utility of satellite communication systems, they suffer from connection-related problems when the satellites are not positioned in an advantageous way. At times, a signal may not be strong enough to ensure proper communication between devices. At other times, a satellite may be out of range of the land-based control station, in which case, unidirectional messages could be sent to the satellites and forwarded to their appropriate recipients when the satellite makes contact with the control station. Most often, these types of errors will occur when the satellite in view is close to the horizon. Then, the effect of Earth's atmosphere is more pronounced. Apart from the sometimes unreliable connection at sea, communication can occur from anywhere on Earth. The combination of communication, navigation, and computing at sea creates a unique setting for the development of specialized software.
2. Project Overview

The main components of a tracking system are:

- a positioning system
- a communications link
- a user (client) program

Communication at sea is often intermittent. Voice or data must be transmitted to a powerful receiver on land or in the sky. From the middle of the ocean to the high-latitudes of the polar regions, radio communication has proved to be a challenge. Given recent growth in the computer industry, and the use of the Internet as an important communication link, many marine communication systems have provided methods for data transfer (See Table 1). It is only a matter of time before communication at sea will be able to handle the bandwidth of common Internet data transfer. Near shore, cellular service providers have designed wireless Internet access that allows efficient data transfer through their digital communication networks. In other areas, cell phones can be used like a standard phone line, in order to obtain dial-up access to the Internet. This project takes into consideration the upcoming “webification” of communication systems across the globe.

When developing a global tracking system for vessels, we must keep in mind the intermittent and possibly unreliable communication link between land and a vessel at sea. The system must be reliable, efficient, and effective. In other words, it must be able to provide accurate, appropriate information upon request from its users. Tracking a vessel could mean providing its current position upon a user’s request. It could also involve the intermittent logging of the vessel’s position to provide additional data.
We developed the following goals for our tracking system:

- To allow multiple users access to one vessel’s position information.
- To allow for intermittent updates of the vessel’s position, as well as the potential for streaming real-time position data to users.
- To incorporate a text messaging system for sending short messages between users and the vessel.

In designing our system, we acknowledged the wireless communication link as the weakest part. The design we propose minimizes the effect of this weak link on the users of the tracking software. Users can download the history of where the vessel has been, as well as navigation information such as its heading and speed. This information could be used for educational purposes, to track a vessel’s progress around the world and teach children about navigation. It could also be used by friends and family to track a vessel’s progress on a voyage. The safety and educational value of the system is also furthered by the ability for two-way text messaging. Messages about weather conditions or course changes could be sent to the users; as long as the vessel is connected to the system, interactive messages can be sent.

a) RemoteNav Features

The software design we present here is called RemoteNav. It is intended to be a low-cost, versatile, platform-independent, vessel-tracking system that provides reliable service despite an often-sporadic ship-to-shore connection. RemoteNav can provide real-time navigation data to a remote computer when the vessel is connected. It is often not cost-effective, nor even possible for a vessel to stay connected throughout its voyage. The ability to connect and upload current navigation data at each update interval is a more reasonable and affordable solution. Given the unreliable nature of the wireless link, some of these intermittent connections may not be successful. In these cases, the system logs the navigation information on the vessel when a connection can’t be made, for later uploading to avoid gaps in the tracking data. We want the user to have access to a complete history of the vessel’s progress, with a precision set by the system’s data update
interval. The on-board logging of the navigation data prevents gaps in the data available to users.

The RemoteNav software contains three main components:
1. Nav. Data Processor and Vessel User Interface (GPServer)
2. Land-Based Proxy Server (ProxyServer)
3. Client User Interface (Client)

Figure 2: Interaction between Clients and the Server
3. Project Design

The primary design decision was the selection of a medium for communication between the vessel being tracked and the client “trackers.” The desire to make the tracking data available to many users, as well as the potential for future development of a web-based client interface, led us to choose a network socket interface. The only requirement is that the computers are connected to a suitable network. For most purposes, this can be the Internet. Based on an evaluation of currently available communication systems, we developed the software with the understanding that a ship-to-shore data connection to a networked computer exists on board the vessel. In some regards, this project is designed with the future in mind. If current trends continue, the cost of wireless global Internet access will continue to drop; service and coverage will become faster and more reliable. Because the current cost per unit of data is high, a large emphasis was placed on minimizing the amount of data sent. Also, speeds of data connections through wireless communication systems are generally not as fast as in normal land-based communication systems such as dial-up access to Internet providers. While current cellular data communication can reach speeds of 33600 bps, most satellite data transfer occurs at speeds around 2400 bps.

a) More Wireless Details

The signal quality of wireless messages often limits the system’s ability to use high baud rates. Some remote parts of the world (generally in higher latitudes near the poles) have weak, if any, coverage, due to the lack of satellites positioned overhead. The most extensive coverage is provided by the Inmarsat satellite network. Rush Hambleton (Colby Class of 1997), an experienced ocean-sailor, shares his experience with Inmarsat during his most recent circumnavigation:

The Inmarsat system has been a little slow to modernize to all-data connections, but they’re getting there. They have high altitude satellites which provide spotty coverage at times (there are only four of them), so it is possible to sail out of their area. When you’re in areas where the coverage is good, and all but the most exotic sailors are, it performs very well.
Just watch out if you're ever in the Indian Ocean, there's a big hole there.4

For many navigators, and potential users of the RemoteNav software, global coverage is not a requirement. Many coastal and offshore vessels are equipped with single-sideband radio (SSB). Data connections over SSB are available through service providers, just as Internet access companies provide service over telephone lines. Some SSB stations support text-messaging, while others provide access to e-mail and the Internet. In many coastal areas, especially around well-populated cities, local cellular phone service can often handle the task of providing dial-up access.

There are many options for connecting a shipboard computer to a networked computer on land. We leave the physical details of making a network connection to the users of the GPServer software on the vessel.

b) RemoteNav Components

With the understanding that an Internet connection can be achieved on board a boat, we began planning the components of the GPServer program. The first step was to figure out the interactions between the vessel and the clients. Given the sporadic nature of Internet connections at sea, influenced greatly by geographic location, weather systems, and cost, we decided that the clients should not rely on a direct connection to the vessel. A direct connection would extend the problems of the wireless communication systems to all users, severely reducing the reliability of the entire system. It would also involve more data transfer over the wireless link, which would incur unnecessary cost for the system’s use.

We want the system to be reliable. By sending tracking information to a remote computer for storage and processing when the vessel is connected, we can achieve this goal. Clients can then connect to this proxy server, a land-based networked computer running the ProxyServer software, and obtain the tracking data current to the time of the vessel’s last connection. If the vessel happens to be connected at the same time as clients, the vessel

---

4 Rush Hambleton, Assistant Director of Admissions, Williams-Mystic Maritime Studies Program, Mystic, CT 06355. After graduating from Colby in 1997, Rush sailed around the world and later captained for ActionQuest Teen Sail Training Programs.
has the option of streaming current navigation data to, and exchanging messages with, the users.

In order to provide position information to the ProxyServer, the computer on board the vessel must be running the GPServer program and must be connected to a NMEA-compatible navigation device. In most cases, this will be a GPS receiver, but the use of the NMEA protocol will allow any NMEA-compatible device, such as a Loran or other positioning device, to provide navigation information. The GPServer program processes the navigation data and stores the vessel’s position, time, course, speed, and heading periodically. When a user on board the vessel connects to the Internet, he or she can connect to the ProxyServer. A connection to the server begins with the most current line of navigation data (68 characters), and is followed by an upload of the log file’s data since the previous connection. Given an update interval of 30 minutes, if a vessel were to send its data once a day, the data in the log file would total approximately 3.3KB (48 data entries, 68 characters each). In addition to navigation information stored in the log file, any log messages would also be posted to the ProxyServer. Log messages could include anything, but would most likely be used to denote instances of course changes, sightings of other vessels, or any other event that would normally be written in the ship’s log. In this way, the ship’s log can be made available to the remote clients. Every time a log message is entered on the vessel, it is time-stamped with a line of navigation data from the GPS, providing accurate time, position, speed, and heading when the message was sent. While the vessel is connected to the server, it sees a list of clients that are currently tracking its progress. Messages can be posted to and from everyone connected to the server, similar to an Internet chat environment.

The ProxyServer manages the data and messages that pass through. This information is stored in the server’s log file so other users can obtain an accurate history of the vessel’s progress. The GPServer software on the vessel can’t rely on the data in the server’s log file (it may not be connected), so it manages its own log in such a way that it can later be sent directly to the ProxyServer. The only difference is that the server’s log file holds data over a long period of time, whereas the vessel’s log file can be cleared as long as it has already uploaded its data to the server. When planning the log file control mechanisms in the software, we needed to implement a way to check
the last line of data in the *ProxyServer*’s log file, and compare it to the last line of the *GPServer*’s log file when connected. If they differ, then all navigation data with a time and date more recent than the *ProxyServer*’s last logged line will be uploaded to the *ProxyServer*.

While designing the components, their layout, and interactions in the system, we tested our ideas with prototypes. Developing software for a design that wasn’t fully specified helped to give us insight toward some of the less-obvious design issues. Some of the software design details are explored below. More information is also available in the prototype source code on the CD-ROM in Appendix D.
4. Development Environment

At the beginning of the project, we explored the different programming languages and platforms that could be used to write the software. Preliminary research, conducted in September of 1999, led to exploration of C++ in a Linux environment. After exploring the environment for several weeks, we decided that the combination of Linux and C++ programming would limit the usage of the final product. We wanted the system to be used by as many people as possible, with the least amount of equipment costs. Although the server program could be implemented in this environment, we decided the user program should be designed for computers running a variety of operating systems. This led to the choice of Java as the language to use. Previous experience with Java, and familiarity of the Windows operating system, played a large part in our choice to make Sun's Java Developer's Kit 1.2.2 running on the Windows 98 operating system our development environment. Since we were writing the software on a Windows machine, our prototype software was designed for the same operating system. In addition, a Garmin GPSIII handheld GPS receiver was used to provide NMEA-0183 formatted navigation data.

The GServer software relies on a connection to a NMEA talker device through one of the computer's ports. In order to transfer data from a talker to the computer (the listener) through a serial port, special Java classes must be used that manage port connections. Due to the hardware-dependent nature of serial ports on computers, the lower-level classes of the extension package are platform-dependent. As of this Spring, the javax.comm classes are only available for Solaris and Windows operating systems. There are classes available for dealing with ports on other platforms, but this version of the vessel's software will focus only on the Microsoft Windows environment. Though the COM port classes are limiting for the GServer software, the ProxyServer and Client software can be run by a Java interpreter on any platform. The requirements of the RemoteNav system, broken down into the three components are listed below (see Table 2).

---

1 Sun Microsystems. "Java Communications API 2.0", java.sun.com/products/javacommm
<table>
<thead>
<tr>
<th></th>
<th>GPServer</th>
<th>ProxyServer</th>
<th>Client</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Hardware</strong></td>
<td>Windows NT/95-98, Navigation Device with NMEA-0183 data output PC connection cable from nav. device to COM port</td>
<td>Any Platform capable of running Java</td>
<td>Any Platform capable of running Java 1.2</td>
</tr>
<tr>
<td><strong>Software</strong></td>
<td>Java 1.2 (with Swing classes) javax.comm extension package</td>
<td>Java interpreter</td>
<td>Java 1.2 (with Swing classes)</td>
</tr>
<tr>
<td><strong>Communications and Networking</strong></td>
<td>Communication device capable of providing a network connection</td>
<td>Must be continuously networked.*</td>
<td>Must be networked to access navigation data.</td>
</tr>
</tbody>
</table>

*Note: Any computer can be used to run the ProxyServer program, as long as it has a continuous Internet connection.*
5. Software Details

a) Handling Commands and Data

With any client-server software package, there must be a protocol with which the devices interact. They must be able to understand the commands they receive. The protocol we decided to use extends the NMEA-0183 protocol, adding several additional sentences to those specified in the NMEA-0183 documentation (see Appendix A). By extending the NMEA protocol, we were able to use the error checking provided by NMEA, as well as the same structure for parsing the fields of each sentence.

The NMEA compatible GPS receiver we used outputs approximately twelve sentences every two seconds. Of these twelve, we are concerned with one of them: the one beginning with $GPRMC (GP for "GPS", and RMC for "recommended minimum specific data"). The RMC line, as we'll refer to it, contains the time, date, receiver status, position, speed, course, heading, and magnetic variation. All this information is computed automatically by the GPS receiver and output every two seconds.

In addition to the RMC lines, we created some new sentences to allow bidirectional communication between the software components in a universal format. These sentences begin with $PJPS indicating they are proprietary sentences for "JP's System". We developed three types of sentences to handle: server commands, client commands, and text messages (see Table 3).

Consider the following interaction between a client and the proxy server: A client logs onto the server, with the name Jared. The server acknowledges his login. Jared then requests the log data starting from the last date in his local log file. The server begins streaming data from the date specified to its most recent entry. After the log file has been uploaded from the server to the client, GPServer logs onto the server as Navigator1. Immediately, Navigator1's current position is sent to the proxy server, the new log file entries from the vessel's log are appended to the server's log file and also streamed to all clients, so everybody's logs are current. Jared then sends a message: "Hi Captain, how's the sailing?" which, along with the most current RMC line, is sent to ProxyServer and forwarded to all connected Clients. The RMC line serves as a time-stamp for the message. On the vessel, Navigator1 receives this message and returns "Making good
progress.” This is logged locally with a time-stamp RMC line and sent to the server to be forwarded to all connected clients along with the current RMC line. Jared logs off, and the server tells all connected users “Jared is disconnected.”

Table 3: Proprietary Sentences

<table>
<thead>
<tr>
<th>Sentence Type</th>
<th>Starts With</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server Commands</td>
<td>$PJPS, ...</td>
</tr>
<tr>
<td>Client Commands</td>
<td>$PJPS, ...</td>
</tr>
<tr>
<td>Text Messages</td>
<td>$PJPSM, ...</td>
</tr>
</tbody>
</table>

The client–server dialog between a Client (Jared) and someone using GPServer aboard the vessel (Navigator1) described in words above, is shown here as a sequence of sentences:

Client connects: $PJPS, Jared, hello, 0*4F
ProxyServer replies: $PJPS, Jared, welcome, 0*56
Client requests data: $PJPS, Jared, log, 000415201200*8E
(ProxyServer then streams the log data from 4/15/00 at 8:12 pm to the most recent entry)

GPServer connects: $PJPS, Navigator1, hello, 1*A3
GPServer sends: $GPRMC, <current nav data fields>
ProxyServer sends: $PJPS, Navigator1, hello, 0*A8
ProxyServer sends: $GPRMC, <current nav data fields>
(GPServer starts streaming RMC lines to the proxy server. These new lines are sent to the Client)

Client sends: $PJPSM, 1, 1, Jared, Hi Cap! how’s the trip?*71
ProxyServer replies: $PJPSM, 1, 1, Jared, Hi Cap! how’s the trip?*71
ProxyServer replies: $GPRMC, <nav data fields>
GPServer sends: $PJPSM, 1, 1, Navigator1, Making progress.*8B
GPServer sends: $GPRMC, <nav data fields>
ProxyServer replies: $PJPSM, 1, 1, Navigator1, Making progress.*8B
ProxyServer replies: $GPRMC, <nav data fields>
Client sends: $PJPSC, Jared, bye, 0*4C
Server replies: $PJPS, Jared, Jared is disconnected, 0*83

The NMEA sentence format is straightforward. The comma-separated fields can be easily parsed to obtain the necessary information. Here are descriptions of the sentences used in this software. Figures 3-6 describe the four sentences used in the RemoteNav system.
Figure 3: GPS Data (RMC line)

SGPRMC,<1>,<2>,<3>,<4>,<5>,<6>,<7>,<8>,<9>,<10>,<11>,<12><LF><CR>
$GPRMC,205001,A,4433.782,N,06939.751,W,000.0,214.8,080400,017.7,79<LF><CR>

205001 Time of fix 20:50:01 UTC

A Navigation receiver warning A = okay, V = warning

4433.782,N Latitude 44° 33.782' North

06939.751,W Longitude 69° 39.751' West

000.0 Speed over ground, 0.0 Knots

214.8 Course Made Good, 214.8 True

080400 Date of fix 08 April 2000 (according to UTC time)

017.7,E Magnetic variation 17.7° East

*79 Mandatory checksum

'UTC time is calculated by the GPS receiver, which synchronizes itself with the satellites' atomic clocks. It is the time at the Prime Meridian, also known as Greenwich Mean Time. Time-zone offsets can be calculated from this data. (Eastern Standard Time is five hours behind GMT.)

The NMEA protocol is not Y0.1K compliant. The rollover for the year occurs every 100 years. This is something that the software needs to address.

Figure 4: Text Messages

$PJPSM,<1>,<2>,<3>,<4>*<5><LF><CR>
$PJPSM,2,1,Jared,Hello! I have a message to pass on *8D
$PJPSM,2,2,Jared,to the captain: "Sail faster!"*A3

<1> Total number of lines in the message: an integer between 1 and 9 (currently messages are limited to 9 lines of data)

<2> Current line number of the message (an integer between 1 and 9)

<3> Username of the person sending the message (each user chooses a name upon connection to the server; each name can be 1 to 8 letters or numbers)

<4> Message body (maximum size is 52 characters per line, it can't contain any backslash (\) characters, or they might be treated as control characters (see <LF>))

<5> Mandatory checksum

<LF> Linefeed control character denoted by \r in Java

<CR> Carriage return, or newline control character, denoted by \n in Java

Figure 5: Client Commands

$PJPSC,<1>,<2>,<3>*<4><LF><CR>

$PJPSC,Jared,hello,0*4F

<1> Username (person sending the command)

<2> Command string

<3> Value, if any

<4> Checksum

<LF><CR> End-of-line characters
Figure 6: Server Commands

```
$PJPSJ,{<1>,<2>,<3>*<4><LF><CR>

$PJPSJ,Jared,welcome,0*56

<1> Username to which the command is responding
<2> Command string
<3> Value, if any.
<4> Checksum
<LF><CR> End-of-line characters
```

\textit{b) Server Details}

The first thing the server does when it receives a line of data is test to be sure the checksum is valid. It computes the checksum for the line and then compares it to the value that was transferred. If they match, the line is valid, otherwise, it has been corrupted and the server decides whether to ignore the line (if it is just a streamed RMC line) or requests it to be retransmitted if it were a command or message. After ten unsuccessful requests, the connection is deemed too poor for reliable communication and the proxy server closes the socket. When the server receives a valid line, it processes it. Processing a line may include streaming it to the clients, storing it in a log file, or sending an appropriate message to one or all clients.

ProxyServer is a Java application that must handle several tasks. It needs to monitor client socket connections, listen to connected clients for commands and messages, send navigation data, commands and messages to clients, read to and write from the log file, and listen for a vessel connection. When GPServer or Clients connect, ProxyServer must send appropriate information based on the data it receives and status of connections. For each task mentioned here, a separate thread of execution is necessary. Java’s flow of execution is usually sequential; when a method is called, it runs until completion, then returns to the next line. Java allows for the creation of separate threads of execution, which take turns using processor time, to perform several tasks concurrently, or at least to keep a program from freezing while it blocks (waits) for data. In the event of a connection error, most likely caused by a poor wireless connection between GPServer and ProxyServer, the proxy server will try to send a disconnect message to the unresponsive user and will then close its socket connection.
ProxyServer starts several threads, which each perform a unique task:

1) **LogThread** - manages writing to and reading from the log file.

2) **ToSocketThread** - monitors outgoing data and sends output to each client.

3) **SocketThread** - a new SocketThread is started for each client that connects. SocketThread’s only task is to listen to the socket for incoming commands and messages.

4) **ProxyServer** - the main thread of execution, waits for a client to connect and starts each newly connected client in a new SocketThread.

5) **CommandProcessorThread** - looks at the incomingLines buffer and processes all incoming lines, dispatching them to the necessary output buffers.

Each thread contains a loop. As soon as its task is performed, the cycle begins again. It may block while waiting for input or output; this cycle generally continues until a condition (generally a boolean value tested inside the loop) is changed. The booleans and buffers are embodied in a class called Environment. The Environment class contains all the shared information and parameters. It is passed as an argument to the constructor methods of all the threads, so the threads can interact with each other by changing values in the Environment object. The contents of the Environment class are described below.

**Configuration Information:**

- **Vessel Port Number**
- **Max Number of Clients**
- **Server Port Number**
- **Log File Name**
- **Log File Update Interval**

The log file update interval is an important control. By selecting a larger interval, the GPServer sends less data over the wireless communication link. This is a tradeoff between the precision desired by users of the tracking system, and the availability or cost of transmitting the data from the vessel to the proxy server.
Thread Management:
- **Array of ClientThreads** holds ClientThreads so they can be started when clients connect.
- **Number of Threads Counter** keeps track of how many clients are connected.
- **Free Thread Array** contains a T or F, telling if the corresponding ClientThread array entry is available to be assigned.
- **Thread Name Array** stores the names of the connected clients.

Note that all the array indices are correlated. For example, a traversal through the ThreadName array may indicate a client "Jared" is connected at index 2. Therefore, when "Jared" logs off, freeThread[2] can be set to true and clientThreads[2] can be set to null. The design of the arrays is not object oriented. It evolved through the slow progress of our software testing and prototype design. The organization of the arrays dealing with the client threads will be one of the first things considered when designing a new version of the RemoteNav system.

Log File Transfers:
- **Get Log From** the date from which to output log lines to a specific client.
- **Receiving Thread Index** the index (of ClientThreads array) of the specific client who requested the logfile data.

StringBuffers used to communicate between threads:
- **individualStream** the data from the logfile requested by a client is placed here by the LogThread.
- **linesToStream** anything placed in this buffer is sent to all connected clients.
- **commandsToProcess** all commands, GPRMC lines, and messages are placed in this buffer, which is read by the commandProcessorThread.
- **linesToLog** anything that is to be written to the log is put here.

The String "currentRMCLine" holds the most recent GPRMC line to be used for timestamping log messages when the vessel is connected.

Booleans for controlling the behavior of the program:
- **runServer** will shut down the entire ProxyServer if false.
- **processCommands** tells the commandProcessorThread to process commands.
- **logData** turns on the logger.
readLogFile retrieves data posted to the log after "getLogFromDate," and puts it in the "individualStream" buffer to be sent to the client who requested it.

clearLogFile clears the log file.

In addition to this information, the environment also contains the method "validChecksum," which takes a line and tests to make sure it has not become corrupted in the transfer. All threads check to make sure the checksum is valid before writing data to a buffer or an I/O stream.

c) Processing Commands

The CommandProcessorThread is really the heart of the ProxyServer software. It takes each line sent by a Client or GSServer, and decides what actions to take based on the current state of the environment. In most cases, the sentence from "commandsToProcess" is placed in another buffer. LogThread, ToSocketThread, and CommandProcessorThread all check for data in their corresponding buffers. If their buffer is empty, the thread loops and checks the buffer again. This brings up a concern that processor cycles will be wasted checking to see if data is available. To fix this, the threads can use wait and notify to prevent unnecessary checking of the buffers. A thread can enter the waiting state when its buffer is empty; when another thread puts data in a buffer, it can call notify on that buffer, and all threads waiting on that buffer will resume.

Another concern is synchronization. We don't want a thread reading a line from a buffer if the writer thread has not completed writing it to the buffer. Fortunately, Java's StringBuffer objects are synchronized internally, so only one thread can access it at a time. Therefore, we can be sure that when one thread writes data to the StringBuffer, no other threads will be able to access it until the writing is complete. It works the same way if a thread tries to write data while another thread is reading from it. This eliminates the need to explicitly implement synchronization in the software.

The ProxyServer software requires little from the computer on which it is running. When no clients or vessel are connected, the server does nothing but listen to two ports for socket connections. When only clients are connected, it may access the log file and stream some or all of its data. It may also log and stream messages sent between connected clients.
ProxyServer is most active when the GPServer is connected, but even in this situation, the server spends most of its time waiting for I/O from the input streams or file access.

*d) Software Similarities*

The Client, ProxyServer, and GPServer employ similar design principles to accomplish their specific tasks. The graphical-interface provided to users of both the client and vessel software can be almost identical. The major differences are on a lower level, such that the GPServer software can manage the hardware connection and process the data from the navigation device.

The design of the Client software is similar to that of the GPServer. The client can choose to store navigation data and messages in a log of its own, for access when not connected to the proxy server. The client can perform the following tasks:

1) connect/disconnect to server
2) view navigation data
3) view/send messages
4) download new log data
5) view the log file.
6) set preferences

Java's Swing classes can be used in the development of the Client software. Swing is an improvement from Java's "AWT" environment. Its use will enable the development of an effective, modern user interface.

The design of the GPServer software needs to differ only slightly. There is a choice of streaming the GPS data and uploading the log to the server rather than downloading the log data. Navigation data is piped directly into GPServer's CommandProcessorThread from the GPSThread, instead of relying on the server to send updated GPS lines (as done by Client).

The Client software is also multithreaded, and operates in a similar manner. Client sets up the following threads, then reads the server's incoming lines:

- guiThread sets up the graphical user interface (GUI)
- logThread performs reading and writing to the log file
- toServerThread sends outgoing messages and commands to the server
commandProcessorThread manages the incoming lines and user commands.

Environment is the shared object that the other threads access (similar to Environment in ProxyServer).

Much of the structure for reading and processing data, messages, and commands is the same across the three systems. The command processor thread must be able to handle different types of commands in different situations. Though some customization is necessary, many of the classes developed for each component of the system can be re-used. A next step would be to design the classes so they are more object-oriented and reusable.

**e) Proposed GUI Design**

The design of the GUI is an important part of the overall system. The navigation window will require the most work when implementing this design. We feel a graphical representation of the vessel is the best way to display the ship’s path and location. The latitude and longitude for each line of navigation data is plotted in the navigation window. When the mouse is moved over the navigation window, the latitude and longitude of the mouse is displayed. To measure distance or heading from any point on the plotter, the user clicks and drags the mouse; the latitude and longitude, as well as bearing and distance to the location of the pointer are shown at the top of the window.

Depending on the speed of the boat, or the range of its travel, the user may wish to zoom in or out, or scroll the plotter. A toolbar at the top of the plotter window will allow this type of graphical manipulation. In addition to the navigation information shown in the plotter window, at the bottom of the application’s main window is a status bar, which indicates whether the vessel is connected, and displays the most current navigation data available, including lat, lon, speed, course, time, and date.

All the time and date information from the navigation data is in Universal Coordinated Time (UTC). UTC, also known as Greenwich Mean Time, is the time at 0° longitude. The vessel could be anywhere in the world, as could the clients tracking the vessel. Deciding to keep all times and dates to a standard, we kept the data in UTC. The client software can set the time.
offset to avoid having to make the calculations between UTC and local time when referencing the navigation information.

This design for a graphical user interface is currently only a prototype. The actual software developed does not contain a graphical user interface beyond that used to demonstrate the data-parsing capabilities of the NMEA protocol.

For a lower-level explanation of the software design, see the source code for the prototypes on the CD-ROM in Appendix D. The design process of the prototype allowed us to experiment with many of the principles outlined above. Though not a complete implementation of the RemoteNav software, the prototype provides a similar service, and is a helpful example of the services the final software will provide.
6. Prototype Development

At first, we wrote simple multithreaded programs, then client-server software, then began looking at the details of input and output streams, file I/O, and serial port access. Once we had gained an understanding of these principles, we started developing the protocol, methods, and overall design to help achieve the goal of producing a simple, straightforward tracking program that will minimize the effect of an often unreliable communication link between the server and the vessel at sea.

The primary goal of the software is to provide navigation information. In order to do this, we first developed a program that took data from a connected GPS and printed it to the screen. Then, we modified it to send just the GPRMC line to an output stream over a network to a client. This worked well, until we realized that the client needed to send commands to the server. A second thread was needed to manage incoming commands from the server. With the multithreading came the need for a shared environment, so the Environment class was developed. The simple solution was growing quickly.

The desire to keep the amount of data transferred from the vessel to the client at a minimum led to the concept of a server on land, to which the vessel could upload its data. We also had to develop a means of storing the navigation data, which resulted in the log file and LogThread, which manages the software's interaction with the log file. The desire to allow more than one person to run the client program at a time led us to create a multi-threaded server with independent socket threads for each client.

The majority of the development time was spent on the server's components. Due to the time constraints, we were forced to combine the GPServer and ProxyServer implementation as one package. The design is such that it could be split into the two separate components with minimal effort. Our prototype software, RemoteServer, normally implements the situation where the vessel is constantly logged on to the server, and is streaming its navigation data. Boolean values in the environment of ProxyServer can be changed to simulate a break in the vessel-server connection. When the connection is broken between the GPServer and ProxyServer, the clients receive a message saying the vessel disconnected. The proxy server cleans up after the GPServer, ensuring that the threads and buffers are configured properly.
We designed a simple client interface for most of our testing. *TestClient* makes a connection to the server, starts a thread that listens to user input to send to the server, and begins posting the server's output to the client's output window. *TestClient* proved invaluable throughout the development of the prototype.

Throughout the learning process of implementing our design choices in Java, it became apparent that we could easily include text messaging into the software. This simple addition could allow users to communicate with each other, or with the vessel being tracked. We chose to implement the chat program and a basic navigation program, which each highlight features of the actual interface. An example interface for the *Client* software was designed (see Section 5.e: Proposed GUI Design), but its features were not implemented.

The project would never have evolved to its current size, nor would it have developed the structure presented in the “Project Design” and “Software Details” sections, without the consistent development and evolution of the prototype. The prototype forced us to evaluate our design decisions as we were implementing them. Often, we came up with improvements and new ideas as a result of troubleshooting some aspect of our design. For example, the use of StringBuffers to hold lines of code was chosen after experiencing problems when we were using ordinary String objects.

The progression of the implementation from simple software to a complicated design was a rewarding and frustrating experience. The prototype is not a polished version of the design, but rather, a necessary learning tool, which proved invaluable to the development of the software and design solution we presented.
7. Conclusions and Future Work

We've discussed threads, sockets, and the user interface as components of the project. We've discussed clients, servers, and data interfaces. Now let's broaden our view, to see how these components contribute to the overall goals of the project.

In today's computer-savvy society, people use electronics to provide automation, optimization, and organization in their everyday life. In a business environment, the use of computers has helped to increase productivity. In the maritime world, electronics have greatly increased the safety and efficiency of vessels at sea.

The concept of globalization has spread as people's views and worldwide experience have changed. The ability to be in constant communication from anywhere on Earth further helps to make the world seem like a smaller place. Though a telephone call from the middle of a trans-oceanic crossing would most likely always be an effective way to assure one's safety, it is not always the best method. The ability to post detailed navigation information and access a track of the vessel's progress is an alternative to a standard telephone call or an e-mail. Our software solution should fill the gap between traditional communication methods and costly commercial tracking software.

We have addressed the issues and expect the final product to be a reliable and affordable solution for tracking vessels at sea. Though we've been discussing this software in the context of maritime tracking, it could just as easily be used on land, or in the air, to provide a means of providing navigation information from a vessel to a larger audience on land.

We've introduced the problem of creating a global vessel tracking program, presented a solution, and provided prototypes with software components for implementing the solution. What's next?

The software outlined above can be expanded to accommodate other types of navigation data, as well as extended to further interpret the data. It is a suggestion—a starting point for future programs to build on. The details of the software may change, but the fundamentals of the tracking program will remain the same.

Just as the prototype shows the features and principles of the proposed software design, the proposed software design highlights the basic features of
a global tracking program. The possibilities for expansion of the project are plentiful, and its applications could be further refined to provide more information. One feature we had hoped to implement in this project was a moving map display on the navigation screen. This feature would have allowed a map to be loaded based on the vessel’s position, upon which the track of the vessel could be superimposed. Another was to provide for the storage of waypoints and other marks, so the clients could download and store positions of features such as racing marks, shipwrecks, channel markers, and good fishing spots.

As designers of this software, we will be the first to acknowledge that this is not the only solution. There are other ways of accomplishing what we have done. We do feel, however, that our solution fulfills the needs of a large percentage of the maritime world. We envision this product in use by yachting clubs to provide tracking of their fleet, by ocean racers desiring to stay in touch with their families during long passages, or maybe by the local pizza parlor that wants to keep track of its delivery personnel.

We hope this project doesn’t fall into the forgotten realm of “things to accomplish.” There is great potential for a system following the fundamentals described here. The further development of this system would make a great “weekend hobby” for the next few years, or would provide a great framework for a large independent study project.
8. The Experience

Is this project almost over? In some ways, I hope so, but for other reasons presented here, I hope it continues to develop. It is impossible to count the number of hours spent reading, planning, discussing, designing, and otherwise developing this project. What started out as a cool idea soon became a great production, as the Senior Scholars Program will inevitably do for any simple-sounding plan. I have had an exciting experience with this project over the past 13 months, seen many successes, failures, and changes in both the design and implementation of the project. I realized my goals were bigger than I expected—I had initially hoped to present a final product by the Spring of 2000.

The most important thing I learned from this project was not in the final product, but rather in the entire process. The importance of setting goals, staying focused, and trying not to be discouraged by setbacks were all challenges I encountered along the way. I realized the importance of setting goals, yet throughout the project, I struggled to achieve only partial success with each goal. This frustration also reinforced the value of working with others on large projects. Most of this work was completed on my own, and at times when writing some of the software I found myself stumped for hours on an aspect of the program, only to have it answered quickly by a fellow computer science major.

Aside from the setbacks, there were many rewarding experiences, often when I needed them most. These helped keep my interest in completing the project. Unfortunately, because I set lofty goals early on, I have trouble accepting my accomplishment to this point. I am confident, though, that this project will be completed in my spare time, so that at some point I can have the sense of closure and satisfaction of a successful implementation of my design.

I’m sailing from Bermuda, to Manchester, Massachusetts, in the end of June 2000. I hope to have a version of the software working by then, so my friends and family can track my progress as we sail.
APPENDIX A: NMEA INFORMATION

The information contained in this appendix comes from the following two sources:


The full NMEA-0183 standard is available from:
Cindy Ensley
National Marine Electronics Association
P.O. Box 3435
New Bern, North Carolina 28564-3435
Phone (919)637-7759
Fax (919)637-8136

NMEA data is in standard ASCII text form; the basic unit, a sentence, is in the following form:

$TKSEN,field1,field2,...*XS<lf><cr>

Sentences are less than 80 characters in length.
$ starts every sentence.
The first two chars are the talker id, which tells what device sent this data.
The next three characters is an identifier for the format of the sentence.
A comma comes before every field of data.
The last field is followed by an optional checksum, in the form of *XS where
XS is the hexadecimal checksum computed by taking the exclusive or of
the characters between (but not including) the $ and *.
A linefeed and a carriage return are also included at the end of each sentence.

SOME TALKER IDENTIFIER MNEMONICS
(Address Characters 1 and 2)

<table>
<thead>
<tr>
<th>TALKER</th>
<th>DEVICE IDENTIFIER</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUTOPilot:</td>
<td>General</td>
</tr>
<tr>
<td></td>
<td>Magnetic</td>
</tr>
<tr>
<td>COMMUNICATIONS:</td>
<td>Digital Selective Calling (DSC)</td>
</tr>
<tr>
<td></td>
<td>Satellite</td>
</tr>
<tr>
<td></td>
<td>Radio-Telephone (MF/HF)</td>
</tr>
<tr>
<td></td>
<td>Radio-Telephone (VHF)</td>
</tr>
<tr>
<td></td>
<td>Scanning Receiver</td>
</tr>
<tr>
<td>DECCA Navigation</td>
<td></td>
</tr>
<tr>
<td>Direction Finder</td>
<td></td>
</tr>
<tr>
<td>Electronic Chart Display &amp; Information System (ECDIS)</td>
<td></td>
</tr>
<tr>
<td>Global Positioning System (GPS)</td>
<td></td>
</tr>
<tr>
<td>HEADING SENSORS:</td>
<td>Compass, Magnetic</td>
</tr>
<tr>
<td></td>
<td>Gyro, North Seeking</td>
</tr>
<tr>
<td></td>
<td>Gyro, Non-North Seeking</td>
</tr>
<tr>
<td>Integrated Instrumentation</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

30
Integrated Navigation
LORAN: Loran-A Loran-C
OMEGA Navigation System OMEGA
Proprietary Code P
Radar and/or ARPA *RA
Sounder, depth *SD
Electronic positioning system, other/general TR
Sounder, scanning SS
Turn Rate Indicator *TI
TRANSIT Navigation System TR
VELOCITY SENSORS: Doppler, other/general *VD
Speed Log, Water, Magnetic VM
Speed Log, Water, Mechanical VV
TRANSUDER YX
TIMEKEEPERS, TIME/DATE: Atomic Clock ZA
Chronometer ZC
Quartz ZQ
Radio Update, WWV or WWVH ZV
Weather Instruments WI

TABLE 5 - APPROVED SENTENCE FORMATTERS
AAM - Waypoint Arrival Alarm
ALM - GPS Almanac Data
APB - Autopilot Sentence "B"
*ASD - Autopilot System Data
BEC - Bearing & Distance to Waypoint, Dead Reckoning
BOD - Bearing, Origin to Destination
BWC - Bearing & Distance to Waypoint, Great Circle
BWR - Bearing & Distance to Waypoint, Rhumb Line
BWV - Bearing, Waypoint to Waypoint
DBT - Depth Below Transducer
DCN - Decca Position
*DPT - Depth
*FSI - Frequency Set Information
GGA - Global Positioning System Fix Data
GLC - Geographic Position, Loran-C
GLL - Geographic Position, Latitude/Longitude
GSA - GPS DOP and Active Satellites
GSV - GPS Satellites in View
GXA - TRANSIT Position
*HDG - Heading, Deviation & Variation
*HDT - Heading, True
HSC - Heading Steering Command
LCD - Loran-C Signal Data
MTW - Water Temperature
*MWV - Wind Speed and Angle
OLN - Omega Lane Numbers
*OSD - Own Ship Data
RMA - Recommend Minimum Specific Loran-C Data
RMB - Recommend Minimum Navigation Information
RMC - Recommend Minimum Specific GPS/TRANSIT Data
*RST - Rate of Turn
*RPM - Revolutions
*RSA - Rudder Sensor Angle
*RSD - RADAR System Data
RTE - Routes
*SPI - Scanning Frequency Information
STN - Multiple Data ID
TRF - TRANSIT Fix Data
*TTM - Tracked Target Message
*VWV - Dual Ground/Water Speed
VDR - Set and Drift
VHW - Water Speed and Heading

31
VLW - Distance Traveled through the Water
VPW - Speed, Measured Parallel to Wind
VTG - Track Made Good and Ground Speed
WCV - Waypoint Closure Velocity
WNC - Distance, Waypoint to Waypoint
WPL - Waypoint Location
XDR - Transducer Measurements
XTE - Cross-Track Error, Measured
XTR - Cross-Track Error, Dead Reckoning
ZDA - Time & Date
ZFO - UTC & Time from Origin Waypoint
ZTG - UTC & Time to Destination Waypoint

4.3 Sample Sentences Dissected
4.3.1 Standard Sentences

A talker typically sends a group of sentences at intervals determined by the unit's update rate, but generally not more often than once per second.

Characters following the "*" are a checksum. Checksums are optional for most sentences, according to the standard.

BWC - Bearing and distance to waypoint - great circle

BWC, 225444, 4917.24, N, 12309.57, W, 051.9, T, 031.6, M, 001.3, N, 004*29
225444 UTC time of fix 22:54:44
4917.24, N Latitude of waypoint
12309.57, W Longitude of waypoint
051.9, T Bearing to waypoint, degrees true
031.6, M Bearing to waypoint, degrees magnetic
001.3, N Distance to waypoint, Nautical miles
004 Waypoint ID

DBT - Depth below transducer

DBT, 0017.6, f, 0005.4, M
0017.6, f 17.6 feet
0005.4, M 5.4 Metres

GGA - Global Positioning System Fix Data

GGA, 123519, 4807.038, N, 01131.324, E, 1, 08, 0.9, 545.4, M, 46.9, M, , *42
123519 Fix taken at 12:35:19 UTC
4807.038, N Latitude 48 deg 07.038' N
01131.324, E Longitude 11 deg 31.324' E
1 Fix quality: 0 = invalid
1 = GPS fix
2 = DGPS fix
08 Number of satellites being tracked
0.9 Horizontal dilution of position
545.4, M Altitude, Metres, above mean sea level
46.9, M Height of geoid (mean sea level) above WGS84 ellipsoid
(empty field) time in seconds since last DGPS update
(empty field) DGPS station ID number

GLL - Geographic position, Latitude and Longitude

GLL, 4916.45, N, 12311.12, W, 225444, A
4916.45, N Latitude 49 deg. 16.45 min. North
12311.12, W Longitude 123 deg. 11.12 min. West
225444 Fix taken at 22:54:44 UTC
A Data valid
DOP is an indication of the effect of satellite geometry on the accuracy of the fix.

4.3.2 Proprietary Sentences

The following are Garmin proprietary sentences. "P" denotes proprietary, "GRM" is Garmin's manufacturer code, and "M" or "Z" indicates the specific sentence type.

$PGRME,15.0,M,45.0,M,25.0,M*22
15.0,M Estimated horizontal position error in metres (HPE)
45.0,M Estimated vertical error (VPE) in metres
25.0,M Overall spherical equivalent position error

$PGRMZ,93,f,3*21
93,f Altitude in feet
3 Position fix dimensions 2 = user altitude
3 = GPS altitude

This sentence shows in feet, regardless of units shown on the display.
APPENDIX B: ACKNOWLEDGEMENTS

It is probably impossible to mention everybody who helped me throughout this project. But, I’ll try my best:

Mom and Dad, thanks for your support, advice, and encouragement. Thanks for everything you’ve done throughout the last 22 years.

Much of the testing of the GPS interface would not have been possible without the generosity of Dean of the College, Earl Smith. Dean Smith unselfishly let me use the much-coveted steeple office of Lorimer Chapel for the second semester. The unobstructed views of the sky from the high office provided the much-needed satellite reception at many times throughout this project.

A majority of this thesis was written in the bucket of a John Deere 850, owned by the Warners, in Jackson, New Hampshire. On a beautiful 75° day in April, I don’t think I could have found a more perfect retreat than on the back porch at Shadowbrook, with a chair, laptop, extension cord, and the perfect table.

Allen Downey, Randy Jones, and Lenny Reich, thanks for your advice, support, and help with my projects this past year.

I must also thank Janine Schwartz, for her motivation and encouragement when I needed it most, and for teaching me that the Mobil Puffin Stop is exactly twelve miles from Colby.

Special thanks goes to Bill Barton for the opportunity to make the voyage back from Bermuda, and for his input on the software design and satellite communication systems.
APPENDIX C: BIBLIOGRAPHY


Barton, William. Experienced Transatlantic Navigator and Ocean Racer. 279 Sagamore St., South Hamilton, MA 01982. www.tazzarin.com


Hambleton, Rush. USCG Licensed Captain for ActionQuest. Personal Interview. Williams-Mystic, 7500 Greenmanville Ave., Mystic, CT 06355. www.bluewaterlogistics.com


