In-rest Vehicle GPS Proximity Warning in Surface Operations

By

Stephen J. Miller

Thesis submitted to the Faculty of the Virginia Polytechnic Institute and State University in partial fulfillment of the requirements for the degree of:

Master of Science

in

Mining and Minerals Engineering

Committee Members:

Dr. Antonio Nieto, Chair
Dr. Erik Westman
Dr. Mario Karfakis

May 2005
Blacksburg, VA

Keywords: In-Rest, GIS, GPS, Proximity Warning
In-rest Vehicle GPS Proximity Warning in Surface Operations

Stephen J. Miller

(ABSTRACT)

Proximity Warning Systems are currently the focus of many research groups. Their goal is to produce a system that will reduce the number of run over incidents that occur in large mobile equipment operations. A majority of these incidents occur when the equipment starts from the In-rest state and begins to move. The addition of a transmission locking mechanism to a GPS based proximity warning system will prevent more run over incidents. This transmission locking mechanism will automatically prevent equipment from moving when there is a high risk for a potential run over incident to occur. Additional safety and optimization for surface equipment can be provided by utilizing GIS software for data analysis just by using the data collected from a GPS based proximity warning system. Combining these ideas and methods can provide better safety for large mobile equipment operations.
Acknowledgements

I want to thank my advisor Dr. Antonio Nieto and my committee members, Dr. Mario Karfakis and Dr. Erik Westman, for their support, ideas, and encouragement during my research.

I would like to thank the entire Mining and Minerals Engineering department at Virginia Tech for their financial support during my research.

Finally, I would like to thank my family and friends for their support and encouragement over the years and for making everything possible.
# TABLE OF CONTENTS

Abstract .................................................................................................................. ii  
Acknowledgements ................................................................................................ iii  
Table of Contents ................................................................................................... iv  
Acronyms ............................................................................................................... v  
List of Figures......................................................................................................... vi  
List of Tables .......................................................................................................... vii  
1. Introduction ....................................................................................................... 1  
# ACRONYMS

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAD</td>
<td>Computer Aided Design</td>
</tr>
<tr>
<td>CDMA</td>
<td>Code-Division Multiple Access</td>
</tr>
<tr>
<td>CVS</td>
<td>Comma Separated Values</td>
</tr>
<tr>
<td>DGPS</td>
<td>Differential Global Positioning System</td>
</tr>
<tr>
<td>ECEF</td>
<td>Earth Centered Earth Fixed</td>
</tr>
<tr>
<td>GGA</td>
<td>Global Positioning System Fix Data</td>
</tr>
<tr>
<td>GIS</td>
<td>Geographic Information System</td>
</tr>
<tr>
<td>GPS</td>
<td>Global Positioning System</td>
</tr>
<tr>
<td>GSA</td>
<td>GPS DOP and Active Satellites</td>
</tr>
<tr>
<td>GSV</td>
<td>GPS Satellites in View</td>
</tr>
<tr>
<td>IP</td>
<td>Internet Protocol</td>
</tr>
<tr>
<td>LIDAR</td>
<td>Laser Infrared Detection and Ranging</td>
</tr>
<tr>
<td>MANET</td>
<td>Mobile Ad hoc Network</td>
</tr>
<tr>
<td>MSHA</td>
<td>Mine Safety and Health Administration</td>
</tr>
<tr>
<td>NAVSTAR</td>
<td>Navigation Satellite Timing and Ranging</td>
</tr>
<tr>
<td>NIOSH</td>
<td>National Institute for Occupational Safety and Health</td>
</tr>
<tr>
<td>NMEA</td>
<td>National Marine Electronics Association</td>
</tr>
<tr>
<td>PWS</td>
<td>Proximity Warning System</td>
</tr>
<tr>
<td>RTK</td>
<td>GPS - Real Time Kinematic Global Positioning System</td>
</tr>
<tr>
<td>USGS</td>
<td>United States Geological Society</td>
</tr>
<tr>
<td>VBA</td>
<td>Visual Basic for Applications</td>
</tr>
<tr>
<td>VTG</td>
<td>Course Over Ground and Ground Speed</td>
</tr>
<tr>
<td>WGS1984</td>
<td>World Geodetic System 1984</td>
</tr>
</tbody>
</table>
# LIST OF FIGURES

## Chapter 2
2.1 Truck Blind Spots ................................................................. 4  
2.2 Run-Over Incident at Surface Coal Operation, September, 2003…… 6  
2.3 Proximity Warning System Overview ....................................... 7  
2.4 Proximity Zone ...................................................................... 8  
2.5 GPS and IP Radios ................................................................... 9  
2.6 Onboard Computer System ...................................................... 10  
2.7 MANET .................................................................................... 13  
2.8 Virtual Mine Software ............................................................ 15  
2.9 Mounted Rugged Computer Hardware ....................................... 16  
2.10 Automatic (left) and Manual (right) transmissions for Large Mobile Equipment ................................................................. 18  

## Chapter 3
3.1 Client and Server Interaction ................................................... 23  
3.2 GPS Tracking Server Control Flow Chart .................................... 25  
3.3 GPS Tracking Server Software ................................................ 27  
3.4 GPS Client Software Flow Chart ............................................. 30  
3.5 GPS Client Software Display ................................................. 32  

## Chapter 4
4.1 Stored Vehicle Data in CSV Format .......................................... 36  
4.2 Stored Vehicle Data in CSV Format With Added Header .......... 37  
4.3 ArcMap Data Frame Setup ...................................................... 38  
4.4 Selection of CSV Data File To Load into ArcMap ...................... 39  
4.5 Source Selection For the Data Layer in ArcMap ......................... 39  
4.6 Viewing the Loaded Data Table in ArcMap ............................... 40  
4.7 Table Attributes of Loaded Data in ArcMap .............................. 41  
4.8 Displaying the XY Data for the Loaded Data in ArcMap ............ 42  
4.9 Setting Up the XY Display of the Loaded Data in ArcMap ........... 43  
4.10 Displayed Location Data For Vehicle 1 in ArcMap .................... 44  
4.11 Exporting Vehicle Location Data to usable GIS Shapefile in ArcMap ................................................................. 45  
4.12 Data Export Settings in ArcMap ............................................. 46  
4.13 Shapefile Attribute Table Containing Vehicle Location Data in ArcMap .......................................................................... 47  
4.14 Loaded Vehicle Location Data as Shapefiles in ArcMap ............ 48  
4.15 Joining Attribute Tables in ArcMap ........................................ 49  
4.16 Join Data Settings between Shapefiles in ArcMap ..................... 49  
4.17 Joined Table Attributes in ArcMap ......................................... 50  
4.18 Add Field Settings in ArcMap ............................................... 50  
4.19 Calculating Field Values in ArcMap ....................................... 51  
4.20 Field Calculator with VBA Code in ArcMap ............................. 52  
4.21 Distance Field with calculated distances using VBA script in ArcMap .......................................................................... 53
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.22</td>
<td>Query Definition in ArcMap</td>
<td>54</td>
</tr>
<tr>
<td>4.23</td>
<td>Query Builder in ArcMap</td>
<td>55</td>
</tr>
<tr>
<td>4.24</td>
<td>Display of Query Data in ArcMap</td>
<td>56</td>
</tr>
<tr>
<td>4.25</td>
<td>Add Cycle Field Settings in ArcMap</td>
<td>57</td>
</tr>
<tr>
<td>4.26</td>
<td>Selected Cycle in Attributes in ArcMap</td>
<td>58</td>
</tr>
<tr>
<td>4.27</td>
<td>Field Calculator for Cycle in ArcMap</td>
<td>59</td>
</tr>
<tr>
<td>4.28</td>
<td>Symbology Settings for location Data Display in ArcMap</td>
<td>60</td>
</tr>
<tr>
<td>4.29</td>
<td>Display of individual Cycles in ArcMap</td>
<td>61</td>
</tr>
</tbody>
</table>

**Chapter 5**

- **5.1** An unlocked manual transmission and a locked manual transmission
- **5.2** An unlocked automatic transmission (left) and a locked automatic transmission (right)
LIST OF TABLES

Chapter 2
2.1 Examples of NMEA Sentences.................................................. 12

Chapter 3
3.1 Data Received from Client Software........................................ 27
3.2 Data written to CSV File and sent to Client(s)......................... 28
3.3 NMEA Sentences from GPS Receiver..................................... 33
3.4 Processed Location Data.......................................................... 33
CHAPTER 1 – INTRODUCTION

Each year, dozens of accidents and several fatalities occur in large equipment operations when smaller equipment is run over by larger equipment. These accidents occur when a smaller vehicle is in the blind spot or path of a larger vehicle and is crushed beneath its tires. Lack of communication between equipment operators and poor visibility are common causes of these accidents.

Accident problems will persist in the future as the size of large mobile equipment is increasing over time to meet production needs. Haul trucks have grown in size up to 400 tons and can easily overshadow smaller equipment. The increase in size has also come with larger blind spots and reduced visibility, both of which magnify the risk of run over incidents. Haul trucks used in mining and construction operations, have blind spots so large that standard size cars such as a supply van could not be seen by the operator in certain conditions. Some blind spot areas can be decreased through the use of mirrors and camera systems in larger equipment. Despite the success of the use of mirrors and camera systems in reducing run over incidents, these incidents still occur.

Imagine this situation in which a run over incident could occur. The operator of a 200 ton haul truck has just returned to his truck after taking a lunch break. The mine supervisor had driven the haul truck to keep the production cycle at its maximum while the operator ate his lunch. Upon completion of his lunch, the operator drives the mine supervisor’s pickup truck and parks it next to the haul truck. The mine supervisor and the haul truck driver switch vehicles and the haul truck operator climbs back into the 200 ton haul truck. The haul truck operator attempts to put the haul trucks transmission into gear, but the gear select lever will not move from neutral. Approximately five seconds later, the mine supervisor’s pickup appears from the blind spot directly in front of the haul truck. The mine supervisor had driven his pickup in front of the haul trucks path, which was unknown to the haul truck operator at the time. Ten seconds later, the haul truck operator is able to put the transmission into gear and proceed with his work. Had the
haul truck operator moved the haul truck forward when he initially attempted, the mine supervisor and his pickup would have been the victims of a run over incident.

The potential accident described above was prevented through the use of a Proximity Warning System and a transmission locking mechanism. In this case, both the pickup truck and haul truck were equipped with GPS units and onboard computer systems. The haul truck was also equipped with a locking mechanism on its transmission. When the onboard computer system of the haul truck detected that the pickup truck was in the proximity zone of the haul truck, it sent a signal to the locking mechanism to prevent the transmission from being placed into gear until the pickup was out of the proximity zone. This prevented the haul truck from moving forward and crushing the pickup.

The potential run over event was logged and plotted on the surface operation map using GIS software along with all potential run over events that have occurred. GIS software allows for areas in which a large number of potential run over events occur to be analyzed to determine if any changes in the operation site will reduce the potential for a run over event.

The haul truck in the above scenario is in what I consider the In-rest state. The In-rest state is where the vehicle is running, the vehicle is not in gear, and the vehicle is not moving. The truck has a potential to move with little or no warning. A parked vehicle will be in the In-rest state when the engine is started.

This report describes the possibility of integrating a transmission lock with a proximity warning system to prevent run over incidents that occur from the In-rest state and utilizing GIS to analyze data for safety and performance in large mobile equipment operations. The system can be developed using basic computer systems, a long range wireless network, GPS receivers, Proximity Warning System software, and GIS software. The system is also available for all equipment in the operation and all equipment that may visit the operation.
2.1 MSHA DATABASE AND FATALGRAM REVIEW AND ANALYSIS

Since 1987, 58 miners have died in accidents involving large haul trucks where reduced visibility contributed to the accident cause (McAteer, 2000). The increase in the size of haulage trucks and other equipment in construction and mining operations over the years have led to reduced visibility for drivers and the increase in the risk of run over incidents. These incidents can result in severe injuries and in some cases fatalities (MSHA 2004). The operator of a haul truck cannot see a vehicle or person close behind or beside the vehicle. A pickup crushed by one of these trucks often resembles a flattened aluminum can. The blind spots where operators cannot see vehicles or people around them have increased in size as the equipment has become larger over time.

Between 1990 and 1996 there were forty-six accidents that occurred when the haulage truck drivers’ vision was obstructed due to the configuration or location of the cab (MSHA 2004). Of these accidents, fifteen involved large capacity haulage vehicles running over smaller vehicles and crushing them, resulting in fatalities in all cases. Eighty-seven accidents occurred when the haulage trucks ran into stationary objects, equipment, or another haulage truck, in which eight fatalities occurred due to the collisions. Some of these collisions may have occurred due to driver error; however, accident investigations indicate that several accidents occurred due to poor communication between drivers of there was an obstruction in the visibility between the vehicles involved.

2.1.1 VEHICLE BLIND SPOTS

During the past six and one-half years, blind spot hazards have attributed to the cause of fatal accidents (Fesak, 1996). A majority of the large haul trucks used today have a skewed blind spot, which is that the blind spot on the left of the vehicle is smaller than
that of the blind spot on the right side of the vehicle. This skew is caused by the position of the operators cab (Figure 2.1).

In most cases, the drivers cannot see the ground, other vehicles, or pedestrians for distances that can be greater than 100 feet from the driver’s seat. While the distance is only 16 feet from the operators cab for the operator’s side of the cab, the opposite side has a blind spot distance of 105 feet. This is large enough for large objects to be hidden from the view of the operator. On larger equipment, the blind spots can increase in size. Large 190-ton and 240-ton haul trucks have blind spots large enough to hide pickup trucks and supply vans from the view of the haul truck operator.

2.1.2 CURRENT PROXIMITY DETECTION SYSTEMS

Over the past several years, research projects by Dalagden, Nieto and Ruff have been conducted to aid in the detection of vehicles and personnel. The first approach to help increase operator visibility was to install mirrors. Mirrors are a significant safety aid; however, they do not achieve adequate coverage of the blind areas, and operator perception of size and distance between objects can be affected by the size and shape of the mirror (Fesak, 1996). Recent advances in cab designs and locations as well as the installation of discriminating warning devices, video cameras, and other state of the art “blind area surveillance systems” have been done in to greatly reduce “blind-spot” hazards. The fatalities that have occurred over the years where the drivers view is
obstructed may have been avoided with effective proximity warning devices, cameras, mirrors or improved cab designs (Fesak G. 1996). MSHA studies have shown that video cameras on the rear and side of the vehicle can improve safety around large vehicles (McAteer, 2000). A monitor in the operator’s cab allows the operator to see blind spots around the vehicle. However, the visibility of the cameras can be affected by fog, mud, or glare on the video screens at night. A camera system can cost up to $7,000 depending on the number of cameras and the special features installed. Typically, a single 320 ton haul truck costs well over $2 million. The cost for camera technology is minimal compared to the potential safety benefit that would occur from the usage of camera systems. The addition of a camera system to large equipment does not prevent all run over incidents. Current research is underway to develop numerous proximity warning systems based on LIDAR, Radar, stereovision camera systems, and GPS (MSHA 2004). The use of these detection systems has been limited due to the high cost associated with the technology.

2.1.3 RUN-OVER CASE STUDY

On September 17, 2003, an incident involving a run over was reported at a coal surface mine (MSHA, 2005). This particular run over incident involved a 190-ton haulage truck and a Ford F-350 van (Figure 2.2). The van approached the haulage truck’s right side and stopped to drop off supplies. The haulage truck moved forward, knocking the van over and crushing it under the right front tire. Both the driver of the van and a passenger of the van were fatally injured.
There are several steps that could have been taken to prevent this particular incident. The driver of the haulage truck could have communicated with the van operators to let them know he was moving forward. This step would have only been effective if the driver of the haulage truck was aware that the van was near. Another way this accident could have been prevented is that the driver of the van could have parked farther away, not in the path of the haulage truck, and signaled the driver of the haulage truck that he was near. Had any proximity warning system been in place and operating, the incident could have been prevented as well.
2.2 GPS BASED PROXIMITY WARNING SYSTEMS

Over the past 5 years, there has been an increase in the research into developing a Vehicle Proximity Warning System that is based on using GPS (Ruff and Steele, 2004). Dagdelen and Nieto (2003) have shown that it is possible to track equipment and in real time and display the locations of other vehicles in the system. Similarly, Ruff and Steele (2004) have shown that this can be done using a Windows CE based computer. These systems involve three different technologies: the GPS receiver, the wireless network system, and a computer with a graphical display interface (Azuma et al. 2001, Nieto 2001, ESRI 2003, Goldbeck et al. 2000, Dagdelen & Nieto 2000, Dagdelen & Nieto 2003, Sung-Ju-Lee et al. 2001). A real-time proximity warning system can be developed by combining a wireless network, a GPS system, and a system of computers (Figure 2.3).

Figure 2.3: Proximity Warning System Overview (Dalagen & Nieto, 2003)
A GPS base proximity warning system works by reading GPS positions in NMEA format from a GPS unit using an onboard computer, extracting position data (latitude, longitude, and altitude) from the NMEA sentences, then converting the data into the local coordinates. The system increases the GPS accuracy by using signal correction received from a differential base station using a radio signal (also called DGPS) or by using the more sophisticated RTK GPS technology, which can be more expensive. Once the system solves for the vehicle location, the system calculates the position of the truck with respect to other trucks by broadcasting its GPS location to those other vehicles using a wireless network. The computer keeps track of vehicle proximity based on the vector distance resulting from each of the GPS vehicle locations and the predefined proximity distance, activating a warning signal when a vehicle is within the predefined distance.

The proximity zone can be dynamically adjusted depending of predetermined safety factors and vehicle characteristics (truck weight, speed, visibility, terrain, and weather). If visibility were low due to fog or bad weather, the zone can be increased to compensate for poor visibility (Seymour, 2004). The proximity zone is a predetermined area or “bubble” surrounding the equipment (Figure 2.4).

![Figure 2.4: Proximity Zone (Miller and Nieto)](image-url)
The size of the proximity zone can be adjusted based on the size of the equipment, as well as the surrounding conditions. The advantage of using a proximity warning system based on GPS is that the system functionality is not affected by the environmental conditions.

A critical factor in a GPS based proximity system is the wireless component and the ability to transmit wirelessly GPS positioning data from each enabled vehicle to a remote client/server. Under the wireless network component, each vehicle is assigned with an IP address creating a mobile wireless network that can function under a peer-to-peer scheme or a standard network interface (Figure 2.5)

Figure 2.5: GPS and IP Radios (Ruff and Steele, 2004)
The GPS and IP radios are mounted on the outside of the vehicle and in a manner that will allow them to receive a good signal that does not depend on their orientation to the source transmission.

Proximity Warning Systems based on GPS allow for real time tracking of equipment and the display of the location of other nearby vehicles on a computer screen for the operator to view (Figure 2.6). When a vehicle is nearby, an audible alert and a visual warning are triggered to alert the operator.

![Figure 2.6: Onboard Computer System (Ruff and Steele, 2004)](image)

Both the GPS signal and the wireless network signal are an integral part of the GPS based proximity warning system. A loss of either signal can result in the potential failure of the system. Other possible failures include hardware failures, such as the GPS units, the onboard computer systems, the central server, and the system software.
The latest development in the science of proximity warning is the application of GPS hardware and software to display vehicle locations and calculate distances between vehicles (O’Connor et al. 1996, Suh et al. 2003). The distances are easily calculated once the NMEA sentences are collected from the GPS receiver and transmitted to other vehicles in the proximity warning system.

2.2.1 GPS

The Global Positioning System (GPS) is a constellation of navigation satellites called Navigation Satellite Timing And Ranging (NAVSTAR), that is maintained by the U.S. Department of Defense (USGS, 1999). Many handheld GPS devices are used by outdoor enthusiasts as an accurate tool for determining their location on the terrain. The GPS receiver determines the current location on the Earth's surface by collecting signals from three or more satellites, and through numerous signal calculations, determines the location though process called triangulation.

The GPS receiver units that can be used with a proximity warning system can be any standard GPS receiver that connects to a computer using either a serial port or a USB port, and outputs the standard NMEA sentences. There are wide varieties of GPS receivers available on the market with many different features available. This variety allows for the selection of a GPS unit that will meet all the requirements needed for the application it will be used with and for the selection for the desired level of accuracy of the GPS location calculated.

There are numerous NMEA sentences, but the most common ones used are GGA, GSA, GSV, RMC, and VTG (Gpsinformation.org, 2005). The most important NMEA sentences include the GGA sentence which provides the current Fix data, the RMC sentence which provides the minimum GPS location information, and the GSA sentence which provides the Satellite status data. The GSV sentence provides the satellites in view data and the VTG sentence provides the velocity information. Most standard off the shelf GPS receivers will output a combination of NMEA sentences that can be used to provide
information to programs about the location, direction, and speed of the equipment that the GPS is mounted in. NMEA sentences are based upon a combination of letters and numbers that are separated by commas and have a predefined structure (Table 2.1).

Table 2.1: Examples of NMEA Sentences

<table>
<thead>
<tr>
<th>Sentence</th>
</tr>
</thead>
<tbody>
<tr>
<td>$GPGGA,123519,4807.038,N,01131.000,E,1,08,0.9,545.4,M,46.9,M,,*47</td>
</tr>
<tr>
<td>$GPGSA,A,3,04,05,,09,12,,,24,,,,2.5,1.3,2.1*39</td>
</tr>
<tr>
<td>$GPGSV,2,1,08,01,40,083,46,02,17,308,41,12,07,344,39,14,22,228,45*75</td>
</tr>
<tr>
<td>$GPRMC,123519,A,4807.038,N,01131.000,E,022.4,084.4,230394,003.1,W*6A</td>
</tr>
<tr>
<td>$GPVTG,054.7,T,034.4,M,005.5,N,010.2,K</td>
</tr>
</tbody>
</table>

The details of the GPS NMEA sentences can be viewed in Appendix A.
2.2.2 WIRELESS NETWORKS

Currently there are several options available for a Wireless Network, such as IP radios, cell phone modems, or CDMA wireless cards. With the recent advances in wireless network technology, it is possible to use standard 802.11 wireless: this standard is an inexpensive but less rugged option that uses standard radio pc-cards based on the 802.11 specification also known as (Wi-Fi) working with one watt amplifiers and omni-directional antennas (Molta, 1999). The range expected using this approach is in the order of several hundred meters, range is increased with the use of routers and repeaters.

In addition to the 802.11 standard, MANET wireless, which was originally developed for military applications, can be used to form a network with individual nodes (Sung-Ju-Lee et al., 2001). MANET is based on more rugged and also more expensive radios; however a MANET signal range can reach several miles without using repeaters or routers and is capable of hopping through its network nodes to increase the total range of an individual node. Recent advances in the algorithms used in node hopping have increased the reliability of peer-to-peer networks.

Figure 2.7: MANET (Buckner and Batsell, 2001)
An advantage of using a MANET is that the nodes can be mobile and positioning of nodes is not a factor. Nodes can be added or dropped from the network with ease.

Most modern wireless networks can transfer data up to 54 Mbps. Most vendors argue that 2 Mbps is enough bandwidth for most terminal-to-host and well written client/server applications, provided that the network is engineered to avoid severe contention (Molta, 1999). Provided that voice and video are not needed to be transferred across the network, data can be successfully transferred across a network at connections speeds of 1 Mbps with little or no data loss.

2.2.3 ONBOARD COMPUTER SYSTEMS AND SOFTWARE

Many different computer systems and software have been developed for use with GPS proximity systems. Dagdelen and Nieto, 2003, are using software they developed called Virtual Mine, which allows for 3D representation of the vehicles involved as well as utilizing the actual terrain (Figure 2.8).
The Virtual Mine software allows for real-time tracking and display of vehicles in 3D. Existing CAD drawings of the topography and other features may also be added as needed to allow for updates to the topography.

The hardware to display the software to the operator is mounted in the cab to provide the operator with the information on the location of other equipment nearby (Figure 2.9).
The rugged computer systems mounted in the test vehicles used by Dagdelen and Nieto can withstand more abuse than a normal computer. This helps prevent computer hardware breakdowns and increases the reliability of the system.

Ruff and Steele, (2004), used a similar program and onboard computer as seen in Figure 2.6. They used a 2D software display on a smaller handheld computer. This smaller version reduces the space required in an operators cab for the computer to be mounted, while still allowing for the visual display.

Both types of computers and software can be used depending on the equipment they are mounted in and the requirements for usage. A rugged tablet PC or laptop computer will provide more processing power to the operator and will be able to perform more
functions than a handheld device. Certain smaller equipment will not need as much computing power and a handheld device would be sufficient to perform the basic functions required by the proximity warning system software.
2.3 TRANSMISSION DESIGN

With the numerous types of mobile equipment available today, it is difficult to analyze the individual transmissions used. However, it is possible to analyze the basic types of transmissions used. Transmissions for large mobile equipment are available in automatic and manual transmissions (ZF Friedrichshafen AG 2004). Figure 2.10 shows a general example of an automatic and a manual transmission for Large Mobile Equipment.

![Automatic and Manual Transmissions](image)

Figure 2.10: Automatic (left) and Manual (right) transmissions for Large Mobile Equipment (ZF Friedrichshafen AG 2004)

All transmissions serve the same basic purpose for mobile equipment. They change the gear ratio between the engine and the drive train to allow for different levels of torque and speed (Howstuffworks 2004). With the larger equipment that is used in construction and mining, most transmissions are geared toward providing more torque and power to the wheels than to providing the equipment with a greater speed. Manual transmissions contain a gear shift lever that is used to place the transmission into gear. Manual transmissions usually have several forward gears and a reverse gear. Automatic
transmissions use a gear select lever that will place the transmission into forward, reverse, park, or neutral. Both transmission types use a gear select lever to place the transmission into gear even though the operation of the transmission is different.
2.4 GIS

In today’s society, Information Technology has become era where the dissemination of information from one location to another has become almost real-time for many applications (Baijal et al, 2004). GIS (Geographical Information Systems) based Information Technology has numerous applications in many different working fields, from civil engineering to military use. Today’s GIS applications can be adapted to almost any data type and any industry (ESRI, 2005). Most datasets can be easily imported into a GIS program and utilized to produce maps of specific areas, as well as provide analysis of many problems. The primary benefit of GIS is that it provides an analysis tool, a storage solution, and a display of spatial and non-spatial data all in one system. It takes the power of relational database software with the power of a CAD based package to display spatial data.

2.4.1 GIS APPLICATIONS FOR DECISION MAKING

The ESRI GIS software package provides numerous tools in performing calculation for decision making. Baijal et al, 2004, have shown that the process for determining optimal positions for equipment can be done. In their paper, it is shown that using terrain data, stream data, land use data, and probable enemy location data can provide the optimal points for placing military bridges, deploying troops, or dropping off supplies. These points are determined by using the GIS data and placing a weighted factor on each part of the decision. Using this system, the area in question can be then mapped with an overall optimization based upon the factors used. Satyanarayana and Yogendran, 2004 also show how data can be calculated and utilized in making many decision and optimizing operations. With GIS, the data can be collected, stored, analyzed, manipulated, and presented for making many important decisions.
2.5 EQUIPMENT OPTIMIZATION

In surface operations, Flinn and Shields, 1999, show that using GPS, dozer production can be optimized. GPS units are located and calibrated to the dozer blade, which allows for accurate grading. Flinn et al., 1998, show that using GPS in all equipment has aided in the optimization of the mining process. With GPS units located on equipment, routing of equipment can be optimized. Service vehicles can locate their next job on a display and quickly travel to the location of their next service. Mutapcic and Sroub, 2003, show that using dynamic wireless networks with GPS data can optimize real-time vehicle routing. This combination of a dynamic network, GPS and GIS enables valuable data to be collected and analyzed. This can allow for the cycle times of equipment can be optimized for increased performance and production. Cycle times are obtainable from GPS location information when it is plotted against an operation map.
CHAPTER 3 – SOFTWARE DEVELOPMENT

Currently, there are few software packages available that use GPS for proximity warning systems. Most available software is in development and is currently developed specific configurations, and may not contain the necessary components needed by this project. Therefore, it was necessary to develop both the server software and the client software to the needs of this project. This method will allow the use of the In-Rest locking mechanism without having to modify any existing code.

The software created for this project was coded in Visual Basic.NET using Visual Studio.NET 2003. The two programs developed during the research are the Server software and the Client software. Within this project, the Server Software will be referred to as the GPS Tracking Server, and the Client software will be referred to as the GPS Tracking Client. The GPS Tracking Server Software will be run on only one computer in a central location, while the GPS Tracking Client software will run on multiple computers onboard any mobile equipment to be used in the GPS Proximity Warning System. All the GPS Tracking Clients will connect to the GPS Tracking Server to send and receive location data between mobile equipment used in the GPS Proximity Warning System. Both the client software and the server software were tested during this research using simulated data as it was not possible to test the functionality of the entire system.

Figure 3.1 shows the interaction between the GPS Tracking Server and the GPS Tracking Client software.
There are multiple clients and a centralized server. The number of clients varies with the number of equipment that are to be tracked in the system. Each client sends the local position data to the GPS Tracking Server, which sends all the client positions to each client. All GPS Tracking Clients can then display the location of all other GPS Tracking Clients in the system.

Figure 3.1: Client and Server Interaction
3.1 GPS TRACKING SERVER SOFTWARE

The GPS Tracking Server software receives the information from all the GPS Tracking Clients, processes, stores, and sends back information to the GPS Tracking Clients based on the information received from each of the GPS Tracking Clients. Figure 3.2 shows the general flow chart for the computer algorithms used in the GPS Tracking Server software.
The flowchart shown above shows the basic operation of the GPS Tracking Server software. Location data is transmitted from each piece of mobile equipment within the system using the GPS Tracking Client software. The Client 1 and Client 2 Data that are
sent to the server contain the location information and the ID information of the individual equipment with the GPS Tracking Client software installed. The GPS Tracking Server Software then collects the incoming data, combines all the location data together, reduced the data to an easier to transmit size, and then transfers it back to the GPS Tracking Client software. The input and output of the data from the GPS Tracking Server, including the combined data from the GPS Tracking Clients is shown in Figure 3.3.

The server program is written in Visual Basic .NET 2003. The code for the server is listed in Appendix B. The code for the server program is written in the Form1.vb. The code contains six subroutines. The first subroutine, called “frmGPSTS_Load”, is run when the program is started. This subroutine sets up the initial settings for the files where the data will be stored. The second subroutine, “btnExit_Click” exits the program. The third subroutine starts the data collection and processing by calling the other three subroutines, “Client1”, “Client2”, and “ProcessSendData”. The two client subroutines handle the IP network connections to and from the clients. The “ProcessSendData” processes the received client data and stores it to CSV files. The data is then reduced, combined, and stored to another CSV file. That reduced data is then sent back to the clients over the network connection. The server is currently set to accept only two connections. Additional connections can be added by modifying the source code.

The server software receives client connections. The client connections can be monitored on the GPS Tracking Server Display (Figure 3.3). Once connections are established, the server will wait to receive data from a GPS Tracking Client.
Here the server software has received connections from two clients and a data string from both. The data received from the GPS Tracking Client software will be in a string format as seen in Table 3.1.

Table 3.1: Data Received from Client Software

<table>
<thead>
<tr>
<th>ID, DATE, TIME, LATITUDE, LONGITUDE, ALTITUDE</th>
</tr>
</thead>
<tbody>
<tr>
<td>01,234343,040405,37.23423,-80.54324,654</td>
</tr>
</tbody>
</table>

The Server software receives basic location information from all the clients. It then combines the data together and stores it into a CSV file that can later be opened for further analysis (Table 3.2). It also takes that combined data and sends it out all the clients that are connected.
Table 3.2: Data written to CSV File and sent to Client(s)

<table>
<thead>
<tr>
<th>XX, DATE, TIME, V1, LATITUDE, LONGITUDE, ALTITUDE, Vn, LATITUDE, LONGITUDE, ALTITUDE...</th>
</tr>
</thead>
<tbody>
<tr>
<td>2, 40405, 174534, 1, 37.23008703, -80.42227906, 600.7, 2, 37.23012059, -80.42252489, 600.5</td>
</tr>
</tbody>
</table>

The XX field is the number of vehicles that are currently being logged. This simulated data has two vehicles that were logged. This field will tell the client software how many times it must perform the distance calculation and how many vehicles it will display. The server software will only relay this information and does not require it for any calculations.

The amount of processing that the data undergoes on the server side of the system is minimal. The server only collects, stores, and distributes data; it does no calculations. The goal of this requirement is to reduce the information that must be sent across the network between the Client and the Server software. When less data is sent across the network, it will result in a lower chance of data being lost across the network.
3.2 GPS TRACKING CLIENT SOFTWARE

The GPS Client Software provides visual display for the equipment operator and also performs the calculations for the distances between equipment in the system. Performing the distance calculations using the GPS Tracking Client software reduces the processing power needed by the server computer when numerous vehicles are used by utilizing the computing power contained in the onboard computer system of each piece of equipment in the GPS proximity warning system.

The Client software consists of several Visual Basic modules. These modules are Form1.vb, CR232.vb, ECEF.vb, DataCollect.vb, ProcessData.vb, and ActivateSystem.vb. The general data flow of the software is shown in Figure 3.4 and the code for each of the modules is in Appendix C.
The client processes the incoming data from the server and the GPS receiver, sends the new GPS location data back to the server, then calculates the distances between vehicles, displays the locations of nearby vehicles to the operator, checks to see if the locking conditions have been met, and then locks the transmission if the conditions are met.
The Form1.vb code consists of the majority of the calculations involved, the network connection, and the program display. It also handles the data from the server to be used in the calculations. The CR232.vb module controls the collection of the GPS NMEA data from the GPS receiver. The ECEF.vb module is used to convert coordinates from latitude, longitude, and altitude to Earth Centered Earth Fixed xyz, coordinates to be used in the distance calculation. The DataCollect.vb module processes the GPS NMEA sentences that are received from the GPS receiver and reduces them to only the location data. The ProcessData.vb module calculates the distances between the local vehicle and other vehicles in the system. The ActivateSystem.vb controls the locking mechanism by checking to see if the conditions are met to lock the transmission.

The client application does all the local data processing and position display locally. It takes incoming locations and processes the GPS location into ECEF xyz coordinates, which are then used for the distance calculations between vehicles. Each local vehicle calculates the distance between itself and every other vehicle using their onboard computer system. To display the local truck location, speed, and direction, the information received from the GPS receiver is processed and then displayed to the operator locally (Figure 3.5)
Figure 3.5: GPS Client Software Display

This display shows to the operator the direction the equipment is heading, and the location of other nearby vehicles within the proximity zone. In this case, the red zone is within a 50 meter radius, the yellow zone is between a 50 meter and a 100 meter radius, and the green zone is from a 100 meter radius to a 150 meter radius. Equipment located beyond 150 meters will show up on the outer edge of the green zone.

The client software receives vehicle location data from the server software at and uses the processed GPS NMEA data collected by the client software to continuously update the display for the operator. The local GPS NMEA sentence information is read into arrays to be processed into a string that can be sent to the server software so that the server can send the data to other clients. Table 3.3 shows the GPS NMEA sentences before they are processed to be used by the client software and sent to the server.
The sentences contain information that is not used by the GPS Tracking Clients or GPS Tracking Server, and is removed to reduce the amount of data that must be transmitted across the network. This reduction of excess data results in only a few bytes of data needing to be transmitted across the wireless network. This size restriction is important as the wireless signal can weaken, depending on the distance between the nodes. A connection of 1 Mbps is enough bandwidth to transfer information and will help prevent any data loss due to data transmission problems.

The NMEA data is processed into a string by the GPS Tracking Client Software, eliminating over 80% of the data that is received from the GPS receiver. The information contained in the five NMEA sentences is processed into only one string that contains all the information needed to be used by the GPS Tracking Client (Table 3.4).

Table 3.4: Processed Location Data

<table>
<thead>
<tr>
<th>ID, DATE, TIME, LATITUDE, LONGITUDE, ALTITUDE, DIRECTION, SPEED</th>
</tr>
</thead>
<tbody>
<tr>
<td>01, 110105, 174250, 37.256544, -80.402323, 606.6, 34.4, 5.5</td>
</tr>
</tbody>
</table>

The data contained in the NMEA sentences is reduced to only the data needed by the software. Some data will be passed to the GPS Tracking Server Software, while other data will be used locally for calculations. The data sent to the server software will consist of the ID, the Date, the Time, and the locations in latitude, longitude, and altitude format.
The Direction and Speed data will be used for calculations by the GPS Tracking Client software and does not need to be sent to the GPS Tracking Server software.

To engage the locking mechanism for the transmission, IF/THEN statements are used to check for the proper conditions. The speed is checked to see if it is less than 1 mph. The speed check is not set to zero due to the GPS receiver drift, which results in a non-zero, but low velocity. The GPS receiver will register a velocity of less than 1 mph, even when it is stationary. If this condition is met, it then will check to see if the distance between the local equipment and another vehicle is less than a specified amount. For testing purposes, 25 meters are used. The last condition is to check for vehicle types. Larger equipment are assigned a different class than smaller equipment. This prevents two larger pieces of equipment from locking each other and producing a stalemate. If the three conditions are met, then the ActivateSystem.vb module will initiate the locking mechanism. The ActivateSystem.vb module will monitor the distances until the other vehicle has moved a safe distance away. The module is also monitoring the other vehicle distances and will keep the lock engaged if another vehicle approaches the danger zone of the equipment with the locked transmission.

The data cycle of sending information to the server GPS Tracking Server, receiving data from the GPS Tracking Server software, and processing the data is continued until the user stops either the client software or the server software. This software allows for continuous monitoring of vehicles that are part of the proximity warning system software.

For the GPS Tracking Client Software to function properly, it must be connected to the GPS Tracking Server, and a GPS receiver must be connected to the computer. If the GPS is not receiving a valid signal, the program will operate, but will not be able to calculate the correct location of the GPS receiver until the receiver receives valid data from the GPS satellites.
CHAPTER 4 – GIS ANALYSIS FOR SAFETY AND OPTIMIZATION

GIS software is used to analyze the data from the vehicles that have GPS installed on them to optimize the operation site to improve production and safety. The data collected may be analyzed to determine any potential high risk run over areas where vehicles were in the danger zones of other equipment by looking at all the areas in which potential run over events could have occurred. To determine these high risk areas, the data must be loaded into ESRI ArcMap and the locations where the distances between equipment were within the proximity zone must be found. These tasks are all be done within ESRI’s ArcMap software and will result in the locations shown on a map of the operation area.

4.1 IMPORTING CSV DATA INTO GIS

To analyze the data, first the CSV file of the vehicle data be opened in ArcMap. The CSV File, shown in Figure 4.1, contains the values for the ID of the vehicle, the Date, the GPS time, the latitude, the longitude, and the altitude in meters of the equipment monitored.
To add the file into ArcMap, a header row must be added to the file with unique fields, which is done by adding the following line to the CSV file:

```
ID, DATE, TIME, LAT, LONG, ALT
```

ArcMap will not add the data without a header line to label the fields that it will make. The header is added by opening the CSV file in Notepad or Microsoft Excel and inserting the row shown above (Figure 4.2).
Now the CSV data file is ready to be added into ArcMap. When ArcMap is first opened, a new map is selected and the data frame properties must be set to the WGS1984 projection. This setting change is done by going to the View Menu and then to the Data Frame Properties. Select the coordinates tab and select the WGS1984 projection under the Predefined / Geographic Coordinate Systems / World Folder (Figure 4.3).
Now that the map coordinates are setup correctly, the edited CSV file is added to ArcMap. The add data button is used to add the data. The vehicle data CSV file is selected (Figure 4.4).
Once the data is loaded into ArcMap, the CSV data table is displayed. To display the table data, click the source tab at the bottom of the layer window (Figure 4.5).

Then right click on the truck1.txt file in the layer lists and go to open (Figure 4.6).
Figure 4.6: Viewing the Loaded Data Table in ArcMap

This selection will display the CSV data in a table format with the fields all labeled correctly with the header row that was inserted into the CSV file (Figure 4.7).
Now that the CSV data is properly loaded into ArcMap, the location of each point logged in the data is displayed on the map. To display the data, right click on the vehicle data file and select the Display XY Data option (Figure 4.8).
This option will open the Wizard to display the XY data (Figure 4.9)
The X Field will be the longitude field and the Y Field will be the latitude field. The Geographic Coordinate System must be set to WGS 1984. This setting is changed by selecting the Edit button and then selecting the coordinate projection in the same manner as the data frame coordinate system was selected when ArcMap was first opened. Once
this process is complete, the vehicle locations are shown on the map for all the times they were collected (Figure 4.10)

Figure 4.10: Displayed Location Data For Vehicle 1 in ArcMap

To get the points into a GIS shapefile to be used for further analysis, the data must be exported into a shapefile and then added back to the map. Right click on the truck1.txt events and go to Data and then Export (Figure 4.11)
Figure 4.11: Exporting Vehicle Location Data to usable GIS Shapefile in ArcMap

This export will create a GIS Shapefile that contains both the spatial information and the table information of the original logged CSV data. To correctly export the data, the coordinates must remain the same as the original dataset. A save location is selected and the file is saved as truck1.shp (Figure 4.12).
Add the shapefile back to the map using the same button to add the original data, except this time add the newly created GIS shapefile. The original data is then removed. The new shapefile will contain the original data as well as the new point information for GIS (Figure 4.13).
The second logged data set available, truck 2 in this example, must be converted to a shapefile using the same method as used to convert the truck 1 data. Once both data sets are converted and added, both should be viewable in ArcMap (Figure 4.14).
4.2 GIS DATA ANALYSIS

To analyze the data in this example, the two shape file tables must be joined. This join will relate the information in the truck 1 table to the information in the truck 2 table. The link will be established using the GPS time in both tables since this will show each vehicle's position for each GPS time. This link will allow for the distance between the two vehicles to be calculated for a given time.

To join the data, right click on truck 1 under the layers display and select Joins and Relates and then Join (Figure 4.15).
The field to join from each table is the “TIME” field (Figure 4.16).

Figure 4.15: Joining Attribute Tables in ArcMap

Figure 4.16: Join Data Settings between Shapefiles in ArcMap
This join will link the two tables together based upon the GPS Time field and allow the table to be viewed as one table (Figure 4.17).

Figure 4.17: Joined Table Attributes in ArcMap

The tables are still individual tables, but the data is linked together and displayed together. Now the table is ready to add a distance field. To do this addition, click the options button and select the add field. Add the distance field as a double (Figure 4.18)

Figure 4.18: Add Field Settings in ArcMap
Once the distance field added, the distances are then calculated by right clicking on the distance field header and then selecting Calculate Values (Figure 4.19).

![Figure 4.19: Calculating Field Values in ArcMap](image)

This click will bring up the Field Calculator in which VBA scripts are run to perform complex calculations to fill in an entire field at once based on other data in the table. In this example, a VBA script is used that calculates the distances between each point using the latitude and longitude of each point (Figure 4.20).
A predefined script `latlongdist.cal` is used to fill in the values for the distances between each of the vehicles at a given time. The script uses the great circle method for calculating the distances using latitude and longitude and finds the distance, in meters, between the points. The full script are seen in Appendix D. After the calculation is complete, the distance field will contain values for the distances between vehicles, which are used for safety analysis (Figure 4.21).
Now that the distance field is populated with the distances between each point at a given GPS time, the locations where one vehicle is within the proximity zone of another vehicle are found.

4.3 GIS DATA QUERIES

To find all the instances where vehicle 2 is within the proximity zone of vehicle 1, a data query is built to find all the calculated distances that are 25 meters or less. This query will show the locations on the map where there was a high risk for a potential run over incident. To perform a query, right click on the Truck1 layer and go to properties. Select the Definition Query Tab (Figure 4.22).
Use the Query builder to set the query correctly. This query will look at the “truck1.distance” field and select all data points that are less than or equal to 25 (Figure 4.23).
After the Query is built, the data in the display will automatically be reduced to just the data that meets the query requirements (Figure 4.24).
This data is exported on its own to separate it from the other data if desired. Queries are a quick method of data reduction within GIS. This example contains only two minutes of simulated operation time for two pieces of equipment. Had this been actual logged data, each vehicle would have thousands of data points. Reducing the data is necessary to properly view only the information that is required. In this case, only the data points where the distances between two points are less than 25 meters are of interest. These points in Figure 4.24 represent the areas where there is a high risk for a potential run over incident. The areas where there is a high risk can be evaluated to determine if there is a possibility to reduce the potential for a run over. Using the distance calculations, the areas that have frequent occurrences are analyzed to see if any changes need to be made.
to reduce the number of potential run over incidents. Certain areas could have smaller structures repositioned to increase visibility to all equipment. Other areas could have additional lighting installed to increase visibility. Certain areas may require equipment to use a different path to reduce the potential risk for run over incidents.

4.4 GIS DATA MANIPULATION

The cycle time of large mobile equipment are easily optimized to maximize production in the operation. Any piece of equipment that is not used to its full potential is loosing money though decreased production. With the data collected by a GPS based Proximity Warning System, cycle time calculations for all equipment logged are possible.

To display the cycle times, first insert a field into the data table called Cycle (Figure 4.25)

![Add Field](image)

Figure 4.25: Add Cycle Field Settings in ArcMap

Then select all the points and times that are in one cycle and then press the “Selected Button”.

57
With the cycle selected, right click the “truck1.cycle” field and select Calculate Values. This calculation will be used to fill in the cycle number by just putting a 1 in the Field Calculator box for the first cycle time (Figure 4.27).
Then select the next cycle and repeat the process, but place a 2 in the box for the truck1.cycle value. Each cycle must be selected and defined. Once the cycles are defined in the table, right click on the truck1 layer and go to the properties and then go to the Symbology Tab (Figure 4.28).
Select the Unique value category. Set the Value Field to the Cycle field and then Add All Values. After you click OK, the data will update itself into the two cycles (Figure 4.29).
Each cycle will be displayed separately. This display will allow any abnormal cycles to be visible. Both cycles in this example are similar and seem normal. The information for each cycle is contained in the truck1 data table. Cycle 1 started at a GPS time of 174534 and ended at a GPS time of 174610 for a time of 76 seconds. Cycle two started at 174612 and ended at 174676 for a cycle time of 64 seconds. With the data collected by this system, cycle times are easily obtained from every vehicle in the system.

The cycle time information is used to maximize the equipment production. The equipment cycle path could be altered to reduce the risk for run over. Using the cycle time information can make sure that the new path does not alter the equipment production.
CHAPTER 5 – TRANSMISSION LOCKING MECHANISM

5.1 TRANSMISSION LOCKING MECHANISM OPERATION

The purpose of the locking mechanism is to prevent larger vehicles from moving when they are at the In-rest condition. This mechanism will take the decision to move the equipment away from the operator when the conditions are not safe for the vehicle to move. The locking mechanism will only activate under the proper conditions, so as to prevent run over incidents when necessary while interference with the daily operation of the equipment is kept minimal.

The first condition requires the vehicle that will have its transmission locked to have a velocity near zero. This safeguard will prevent the system from activating on manual transmissions when the operator is shifting gears. If the locking mechanism was activated while the equipment is moving, the results could be disastrous. The second condition requires the vehicle that will have its transmission locked to have its transmission in neutral. Applying the locking mechanism while the transmission is in gear will lock the transmission in gear, which could still allow it to move. The transmission will only lock if the transmission is in the neutral or park position. The third condition requires the other vehicle that triggers the locking mechanism to be within the predefined proximity zone of the larger equipment. This means that another vehicle is at risk for a run over incident.

A vehicle that has an active locking mechanism will not activate the locking mechanism of other vehicles when they enter each others’ predetermined proximity zones. This rule will prevent two vehicles from parking next to each other that both have locking mechanisms and subsequently locking the other vehicles transmission and causing a stalemate of the two vehicles.

The three conditions that must be met to lock the transmission will be controlled by two sources, the software and the locking mechanism. The software, which is discussed in
the next chapter, will monitor for the velocity and the distance conditions. The status of the transmission will be checked by the locking mechanism. The signal to lock the transmission will be sent to the locking mechanism when the first two locking conditions are met. The locking mechanism will detect whether or not the transmission is in neutral or park. If the transmission is in the proper state, i.e. in neutral or park, the locking mechanism will lock the gear select lever. If the transmission is not in the proper state, the locking mechanism will not lock the gear select lever.

For maintenance purposes, an override option will be available to disable the system when a situation arises. This override may be as simple as shutting down the onboard computer system or using a switch located on the outside of the equipment that controls the locking mechanism. This control will not be made easy for the operator to turn off. Bypassing the system will only be performed when necessary as to increase the reliability of the system.

The locking conditions were selected based upon certain factors that are involved in most run over incidents that occur from the In-Rest condition. The In-Rest condition is generally when the equipment has been parked and is no longer in gear. Preventing a run over from occurring could be done while the equipment is still in the In-Rest condition. With the majority of run over incidents occurring from the In-Rest condition, stopping a run over before the equipment begins to move is the best way to prevent it.

The approach taken to design the locking mechanism is to prohibit large equipment from moving if it is not in gear and not moving when the proximity warning system detects a vehicle or person in the proximity zone of the larger equipment. While the locking mechanism is locked, the system initiates a series of audio and visual warning signals to inform all the operators involved. If the locking mechanism can not be engaged, due to one of the conditions not being met, the operators will still be warned that there is a potential for a run over.
5.2 TRANSMISSION LOCKING MECHANISM DESIGN

The locking mechanism is a fairly simple design depending on the equipment type and the transmission. There are numerous methods that could be used to lock a transmission depending on the mobile equipment used. The locking mechanism could be an electronic or mechanical block on the transmission mechanism that is only activated if the equipment is at rest and in neutral.

In order to correctly design a locking mechanism for large mobile equipment, the equipment transmission must be analyzed to determine the best locking approach and design. There are two possible locking solutions based on the characteristics of the transmission in consideration. For example, if contemplating a manual transmission, a mechanical lock is the only reasonable solution, if considering an automatic transmission an electronic lock alternative may be considered if the transmission is controlled electronically, otherwise a mechanical lock should be considered.

The locking mechanism design for manual transmissions is a basic fork like brace that is placed around the shift lever to prevent it from moving into gear. The fork like brace is put into place by a pneumatic mechanism or electronic mechanism. The locking mechanism is installed in the transmission gear lever enclosed by the transmission housing. For the locking mechanism to be successfully engaged, the gear select lever must not be in a gear so that the fork can prevent it from moving into a gear. Figure 5.1 shows both an unlocked and a locked transmission for a manual gear transmission.
This locking mechanism is activated with an electronic signal generated by the onboard computer system that is monitoring vehicles for proximity and monitoring the local vehicle velocity.

In case of automatic transmissions, the mechanism used will include a device similar to the manual transmission lock. The mechanism will prevent the gear select lever from the transmission from going into gear from neutral or park. Automatic transmissions may use a locking device similar to the design for manual transmissions. The basic design of the locking fork is seen in Figure 5.2, and shows how this mechanical device will work on an automatic transmission.
The mounting and the location of installment will be based on the transmission and equipment design. It may be possible to use the same mechanical device on both automatic and manual transmissions. The majority of larger equipment will likely have an automatic transmission; therefore it may not be necessary to develop a locking mechanism for a manual transmission.

Many of the new automatic transmissions designed and built today for large mobile equipment are electronically controlled. Depending on the transmissions design, electronic or mechanical, it is feasible to use an electronic locking device to prevent the equipment transmission from being put into gear. An electronically controlled switch could be added to the control system of the transmission, which is activated under the same conditions described earlier. It will prevent the signal from the gear selection device from telling the transmission to switch gears.

With the variation in equipment design, it will be difficult to design a locking mechanism that will fit with all equipment easily. The need for a locking mechanism on certain large equipment will dictate which equipment designs to apply a locking mechanism to.
Several sizes and designs of locking mechanisms may need to be designed to outfit all equipment designs that will benefit from the use of a transmission locking mechanism.
CHAPTER 6 – CONCLUSIONS AND FUTURE WORK

6.1 CONCLUSIONS

The majority of run over accidents occurs from the In-rest state and is mainly caused by lack of communication between operators. As the operators are unaware of the intentions and actions of other operators, the operator of the larger equipment will move the equipment before the smaller equipment is clear. An In-rest GPS proximity warning system with transmission lock will prevent the larger equipment from moving before the area is clear. It will also provide operators of the large equipment with the location of smaller equipment that may not be visible to them.

To optimize a system that utilizes a transmission lock, several studies will need to be conducted. Tests must be performed to determine the optimal proximity zone or bubble for the equipment involved in the system. The proximity zone will be based on the equipment size and type. It will also be possible to dynamically adjust the proximity zone for certain conditions, including the current weather, the location of the vehicles, and the time of the day. For example, when visibility is lower, the proximity zone should be increased to compensate based on historical information analyzed by GIS software.

The software developed for this project has shown the potential to develop a full working system that would incorporate changes in condition real time to adjust the proximity zone. The client software design allows for easy modification to add features or to obtain other data to return to the server software. The server software will need to be altered to accept a larger number of vehicles since for the server simulation; the clients were limited to only two in order to reduce the amount of simulated data needed.

While the software and hardware for this effort are currently stable, the possibility of system failure must be analyzed. A failure of the lockout system does not necessarily mean that the truck operator could run over a pickup or a person. The chances that system failure and the potential for a run over incident would occur at the same time are small. There are three main scenario failures associated with this system. The first
scenario occurs when the system activates and locks the transmission when it should not, resulting in a positive-proximity signal. The result of this failure is a loss in production time. In an industry where production time is money, a failure of this nature could cost a company thousands of dollars with each failure. The second scenario occurs when the system does not activate when it should to prevent a run over incident, i.e. a vehicle is within the proximity zone of the haul truck and no positive signal for proximity is registered resulting in a run over incident. The results of a failure such as this are potentially tragic and may cost a company up to one million dollars. The third scenario of failure would result if the nearby vehicle is not equipped with the system, meaning that the equipment is not able to detect the nearby vehicle. To prevent this failure, all equipment will need to be integrated into the system, including visiting vehicles.

The accuracy of the system is based upon the GPS accuracy. If DGPS or RTK GPS units are used, sub meter accuracy is possible. The consistency of obtaining usable GPS signals will affect the accuracy of the system. If a GPS position is not obtainable due to obstructions, then the system will not function properly. The ability to obtain a GPS signal will depend on the GPS satellite configuration and obstructions, such as buildings and high walls, which will change with each site that this system is installed in. When advancements are made in GPS accuracy and signal usability, the advancements may be implemented into this system with little change to the overall system by simply upgrading the GPS units used.

Depending on the needs of the operation, a GIS based approach can increase the data obtained while increasing the ability to manipulate more data. GIS is a time and cost effective method when utilizing large data sets, especially spatial data. GIS software can provide the tools necessary to optimize equipment production while increasing worker safety.

The primary focus of this work was to determine the possibility of utilizing a GPS based Proximity Warning System with a transmission locking mechanism for large mobile equipment and the benefits from using a system such as this. The success of a GPS
proximity warning system with a transmission locking mechanism for in-rest vehicles depends on several factors. First, the cost of the system must be low, as well as operating costs. Without a low cost solution, there is no great desire to implement a system such as this. The main costs incurred are the initial startup and installation fees, capital cost for the equipment and software, and operating costs on the system, with the capital costs being the highest cost incurred. Second, the system must not interfere with the equipment operation except in cases where there is risk for a potential run over that could occur at the in-rest condition. Any unneeded interference in the daily operation of equipment can result in slowed operation or lost production. The third factor is the information obtained from using a GPS based system. In addition to providing proximity warning system to the mobile equipment operation, production and equipment tracking is possible from a central site. This provides fast and reliable data for production reports on every piece of mobile equipment in use at an operation.

With the ever increasing costs associated with run-over incidents, fatal and non-fatal alike, the overall savings and benefits from using a GPS proximity system and transmission locking mechanism could easily make this system a worthwhile investment for all large mobile equipment operations. The information provided by using a system such as this is also valuable and can aid in optimizing operations or processes at a site where large mobile equipment is used. When both of these are considered, the benefits for using a GPS based proximity warning system will easily outweigh the associated costs.

6.2 FUTURE WORK

In addition to utilizing a transmission locking mechanism, it is also be possible to implement additional safety systems. A device that could retard the throttle of the larger equipment by interrupting the flow of fuel to the engine could be used to prevent the equipment from moving. Another device that could be used is an ignition kill switch, which would shut the engine down and keep the equipment stationary. These types of mechanisms may not be desirable as most large equipment is left running continuously;
however, they may be the only device that will work on certain equipment types and sizes.

Visiting vehicles can be integrated into the system by utilizing a device that contains a GPS receiver, a wireless networking device, and a small onboard computer that will handle the data that will be sent to the server. In this case, the visiting operator’s device will not receive the data and the positions of the equipment from the server. This setup is only for small vehicles that have the potential for being run over, as they have very little potential for running over other equipment. Visiting vehicles will be easy to accommodate into the system as they only require placing a GPS receiver and wireless network device on the roof of the vehicle using a magnet or suction device and providing power to the device though a standard cigarette power adapter, found in most vehicles today.

ESRI has recently released a software package that allows for real-time tracking of GPS based equipment, called Tracking Server. Tracking server can be customized to perform the same operations as the Server Software created for this project. The benefits to utilizing Tracking Server would be that the data would be stored in a GIS spatial format and would not have to be converted from a CSV file. Information can be easily accessed and manipulated in a GIS format. The vehicles could also be tracked in real-time from the server, allowing for an overview of the entire operation from a central site. The ability to optimize an operation would take less time with an integrated GIS based data server and storage program.
REFERENCES


Boldt, C.M.K. “Improved Visibility for Operating Large Haulage Equipment” 
Civil Engineer, Spokane Research Center, Spokane, WA (2005) 


Presented at 29th APCOM International Conference Proceedings, Beijing, China (2001).


NMEA. NMEA 2005 <http://www.nmea.org/> 


Seymour, Chris, “GPS BASED MACHINE Guidance In The Mining Industry” 2004


APPENDICES
APPENDIX A: NMEA SENTENCE INFORMATION

$GPGGA,123519,4807.038,N,01131.000,E,1,08,0.9,545.4,M,46.9,M,,*47$

Where:

GGA          Global Positioning System Fix Data
123519       Fix taken at 12:35:19 UTC
4807.038,N   Latitude 48 deg 07.038' N
01131.000,E  Longitude 11 deg 31.000' E
1            Fix quality: 0 = invalid
             1 = GPS fix (SPS)
             2 = DGPS fix
             3 = PPS fix
             4 = Real Time Kinematic
             5 = Float RTK
             6 = estimated (dead reckoning) (2.3 feature)
             7 = Manual input mode
             8 = Simulation mode
08           Number of satellites being tracked
0.9          Horizontal dilution of position
545.4,M      Altitude, Meters, above mean sea level
46.9,M       Height of geoid (mean sea level) above WGS84
(empty field) time in seconds since last DGPS update
(empty field) DGPS station ID number
*47          the checksum data, always begins with *

$GPGSA,A,3,04,05,,09,12,,,24,,,,,2.5,1.3,2.1*39$

Where:

GSA      Satellite status
A        Auto selection of 2D or 3D fix (M = manual)
3        3D fix - values include: 1 = no fix
         2 = 2D fix
         3 = 3D fix
04,05... PRNs of satellites used for fix (space for 12)
2.5      PDOP (dilution of precision)
1.3      Horizontal dilution of precision (HDOP)
2.1      Vertical dilution of precision (VDOP)
*39      the checksum data, always begins with *

$GPGSV,2,1,08,01,40,083,46,02,17,308,41,12,07,344,39,14,22,228,45*75$

Where:

GSV      Satellites in view
2        Number of sentences for full data
1 sentence 1 of 2
08 Number of satellites in view

01 Satellite PRN number
40 Elevation, degrees
083 Azimuth, degrees
46 SNR - higher is better
for up to 4 satellites per sentence
*75 the checksum data, always begins with *

$GPRMC,123519,A,4807.038,N,01131.000,E,022.4,084.4,230394,003.1,W*6A

Where:
  RMC Recommended Minimum sentence C
  123519 Fix taken at 12:35:19 UTC
  A Status A=active or V=Void.
  4807.038,N Latitude 48 deg 07.038' N
  01131.000,E Longitude 11 deg 31.000' E
  022.4 Speed over the ground in knots
  084.4 Track angle in degrees True
  230394 Date - 23rd of March 1994
  003.1,W Magnetic Variation
*6A The checksum data, always begins with *

$GPVTG,054.7,T,034.4,M,005.5,N,010.2,K

where:
  VTG Track made good and ground speed
  054.7,T True track made good
  034.4,M Magnetic track made good
  005.5,N Ground speed, knots
  010.2,K Ground speed, Kilometers per hour
APPENDIX B: SERVER SOFTWARE CODE

Form1.vb

Imports System.Windows.Forms
Imports System.Threading
Imports System.Net.Sockets
Imports System.IO
Imports System.Net

Public Class frmGPS
Inherits Form

Private connection As Socket
Private connection2 As Socket
Private readThread As Thread
Private readThread2 As Thread

Private socketStream As NetworkStream
Private socketStream2 As NetworkStream

Private writer As BinaryWriter
Private writer2 As BinaryWriter
Private reader As BinaryReader
Private reader2 As BinaryReader

Private thrConnection1 As Thread
Private thrConnection2 As Thread
Private thrProcess As Thread

'Setup the Stream Writers

Dim output1 As StreamWriter
Dim output2 As StreamWriter
Dim output3 As StreamWriter

'Setup the Stream Readers

Dim input1 As StreamReader
Dim input2 As StreamReader

'Setup File Names
Dim filename1 As String = "C:\Truck01.txt"
Dim filename2 As String = "C:\Truck02.txt"
Dim filename3 As String = "C:\CompositeData.txt"

'Setup Loop Stop
Dim iStop As Integer = 0
Dim iStop2 As Integer = 0

'Setup Data transfer
Dim data1 As String
Dim data2 As String
Dim data1old As String
Dim data2old As String

#Region " Windows Form Designer generated code "

Public Sub New()
    MyBase.New()

    'This call is required by the Windows Form Designer.
    InitializeComponent()

    'Add any initialization after the InitializeComponent() call
    thrConnection1 = New Thread(AddressOf Client1)
    thrConnection2 = New Thread(AddressOf Client2)
    thrProcess = New Thread(AddressOf ProcessSendData)
End Sub

'Form overrides dispose to clean up the component list.
Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)
    If disposing Then
        If Not (components Is Nothing) Then
            components.Dispose()
        End If
    End If
    MyBase.Dispose(disposing)
End Sub

'Required by the Windows Form Designer
Private components As System.ComponentModel.IContainer
'NOTE: The following procedure is required by the Windows Form Designer
'It can be modified using the Windows Form Designer.
'Do not modify it using the code editor.
Friend WithEvents btnExit As System.Windows.Forms.Button
Friend WithEvents btnStart As System.Windows.Forms.Button
Friend WithEvents txtDisplay As System.Windows.Forms.TextBox
<System.Diagnostics.DebuggerStepThrough()> Private Sub InitializeComponent()
    Me.txtDisplay = New System.Windows.Forms.TextBox
    Me.SuspendLayout()

    'btnExit
    Me.btnExit.Location = New System.Drawing.Point(416, 224)
    Me.btnExit.Name = "btnExit"
    Me.btnExit.Size = New System.Drawing.Size(88, 32)
    Me.btnExit.TabIndex = 1
    Me.btnExit.Text = "E&xit"

    'btnStart
    Me.btnStart.Location = New System.Drawing.Point(8, 224)
    Me.btnStart.Name = "btnStart"
    Me.btnStart.Size = New System.Drawing.Size(80, 32)
    Me.btnStart.TabIndex = 2
    Me.btnStart.Text = "&Start"

    'txtDisplay
    Me.txtDisplay.Location = New System.Drawing.Point(8, 8)
    Me.txtDisplay.Multiline = True
    Me.txtDisplay.Name = "txtDisplay"
    Me.txtDisplay.Size = New System.Drawing.Size(496, 208)
    Me.txtDisplay.TabIndex = 3
    Me.txtDisplay.Text = ""

    'frmGPSTS
    Me.AutoScaleBaseSize = New System.Drawing.Size(5, 13)
    Me.ClientSize = New System.Drawing.Size(512, 262)
    Me.Controls.Add(Me.txtDisplay)
    Me.Controls.Add(Me.btnStart)
    Me.Controls.Add(Me.btnExit)
Me.Name = "frmGPSTS"
Me.Text = "GPS Tracking Server 1.0"
Me.ResumeLayout(False)

End Sub

#End Region

Private Sub frmGPSTS_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
    Try
        output1 = New StreamWriter(filename1)
        output2 = New StreamWriter(filename2)
        output3 = New StreamWriter(filename3)
    Catch ex As Exception
    End Try

End Sub

Private Sub btnExit_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnExit.Click
    Me.Close()
End Sub

Private Sub btnStart_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnStart.Click
    btnStart.Enabled = False
    thrConnection1.Start()
    thrConnection2.Start()
    thrProcess.Start()
End Sub

'Begin Connection Threads
Public Sub Client1()
    'First Connection
    Dim listener As TcpListener

    Dim counter As Integer = 1
    Dim IPAdd As IPAddress

    IPAdd = IPAddress.Parse("192.168.1.100")
'Second COnnection

Try

    listener = New TcpListener(IPAdd, 4000)

    listener.Start()

'Wait for Client 1 connect
While True
    txtDisplay.Text = "Waiting for Client." & vbCrLf

        ' accept an incoming connection
        connection = listener.AcceptSocket()

        ' create NetworkStream object associated with socket
        socketStream = New NetworkStream(connection)

        ' create objects for reading and writing across stream
        writer = New BinaryWriter(socketStream)

        reader = New BinaryReader(socketStream)

        txtDisplay.Text &= "Connection 1 received." & vbCrLf

        ' inform client that connection was successful
        writer.Write("SERVER>>> Connection successful")

        'txtSend.ReadOnly = False
        Dim theReply As String = ""

        ' Read String data sent from server
        Try

            ' loop until client signals termination
            Do
                theReply = reader.ReadString() ' read data
            'Write to file

            'txtSend.Text = theReply & vbCrLf
            'cbReadData.Text = "Client said: " & theReply & vbCrLf
        Loop

        ' Write to file

    'txtDisplay.Text &= "Connection 1 lost." & vbCrLf
    End While
output1.WriteLine(theReply)
output1.Flush()
'Pass incoming data to be processed
data1 = theReply
' display message
txtDisplay.Text &= vbCrLf & "CLIENT1>>> " & theReply

Loop While (theReply <> "CLIENT1>>> TERMINATE" _
AndAlso connection.Connected)

' handle exception if error reading data
Catch inputOutputException As IOException
    MessageBox.Show("Client application closing")

' close connections
Finally

    txtDisplay.Text &= vbCrLf & _
        "User terminated connection"

    'txtSend.ReadOnly = True

    ' Step 5: close connection
    writer.Close()
    reader.Close()
    socketStream.Close()
    connection.Close()
    iStop = iStop + 1

    counter += 1
End Try

End While

' handle exception if error occurs in establishing connection
Catch inputOutputException As IOException
    'MessageBox.Show("Server application closing")
End Try
End Sub

Public Sub Client2()
    'Second Connection
    Dim listener2 As TcpListener

    Dim counter As Integer = 1
    Dim IPAdd As IPAddress
    IPAdd = IPAddress.Parse("192.168.1.100")
    'Second Connection

    Try

        listener2 = New TcpListener(IPAdd, 4001)

        listener2.Start()

        ' Step 3: establish connections upon client request
        While True
            txtDisplay.Text = "Waiting for Client." & vbCrLf
            ' accept an incoming connection
            connection2 = listener2.AcceptSocket()
            ' create NetworkStream object associated with socket
            socketStream2 = New NetworkStream(connection2)
            ' create objects for reading and writing across stream
            writer2 = New BinaryWriter(socketStream2)
            reader2 = New BinaryReader(socketStream2)

            txtDisplay.Text &= "Connection 2 received." & vbCrLf
            ' inform client that connection was successfull
            writer2.Write("SERVER>>> Connection successfull")
'txtSend.ReadOnly = False
Dim theReply As String = ""

' Step 4: read String data sent from server
Try

' loop until client signals termination
Do
    theReply = reader2.ReadString() ' read data
    'Write to File
    output2.WriteLine(theReply)
    output2.Flush()
    'Pass incoming data to be processed
    data2 = theReply
    ' display message
    txtDisplay.Text &= vbCrLf & "CLIENT2>>> " & theReply
Loop While (theReply <> "CLIENT2>>> TERMINATE" AndAlso connection.Connected)

' handle exception if error reading data
Catch inputOutputException As IOException
    MessageBox.Show("Client application closing")

' close connections
Finally
    txtDisplay.Text &= vbCrLf & "User terminated connection"

'txtSend.ReadOnly = True

' Step 5: close connection
writer2.Close()
reader2.Close()
socketStream2.Close()
connection2.Close()
iStop = iStop + 1
counter += 1
End Try
End While

' handle exception if error occurs in establishing connection
Catch InputOutputException As IOException
' MessageBox.Show("Server application closing")
End Try

End Sub
' End Connection Threads

Public Sub ProcessSendData()

Dim data12() As String
Dim data22() As String
Dim returnData As String

Do Until iStop2 = 2
' Wait for both clients to send data

Do Until data1 <> "" And data2 <> "" And data1 <> data1old And data2 <> data2old
Loop

' Read Data
data1old = data1
data2old = data2

' Split up data into an array
data12 = data1.Split(",")
data22 = data1.Split(",")

returnData = "02," & data12(1) & "," & data12(2) & "," & data12(3) & "," & data12(4) & "," & data12(5) & "," & data22(0) & "," & data22(3) & "," & data22(4) & "," & data22(5)

output3.WriteLine(returnData)
output3.Flush()

txtDisplay.Text & vbCrLf & "SERVER>>> " & returnData

writer.Write(returnData)
writer2.Write(returnData)

Loop

End Sub

End Class
APPENDIX C: CLIENT SOFTWARE CODE

The Client Software Consists of the following parts:

Form1.vb

Imports System.Threading
Imports System.Drawing
Imports System.IO
Imports System.Math
Imports System.Windows.Forms
Imports System.Net.Sockets
Imports System.Net

Public Class frmGPSTrack
    Inherits System.Windows.Forms.Form
    ' Private Memebers
    Private WithEvents GPSData As Rs232

    'Globals
    Dim intPort As Integer = 3 ' Change this Later so it can be set in the options
    Dim strGPSData() As String
    Dim strProcessedGPSData() As String
    Dim strECEFLocal() As String
    Dim strECEFOther(,) As String
    Dim strGPSData2 As String
    Dim strECEFOtherPass(5) As String
    Dim strECEFCalculated(5) As String
    Dim strDistance() As String
    Dim strDistanceTo As String = "The Distance to vehicle "
    Dim strSPDDIR(1) As String
    Dim blnStop As Boolean = False
    Dim blnRestart As Boolean = False
    Dim strLocalID As String = "01"
    Dim intNumOfVehicles As Integer
    Dim strTemp() As String
    Dim intCount As Integer = 0
    Dim x2 As Double
    Dim y2 As Double
    Dim z2 As Double

    'Client Variables
    Public blnStop2 As Boolean = False

    Private output As NetworkStream
Private writer As BinaryWriter
Private reader As BinaryReader
Private message As String = ""

Private readThread As New Thread(AddressOf RunClient)

'Threads
Dim thrDataCollect As New System.Threading.Thread(AddressOf DataRead)

#Region " Windows Form Designer generated code "
Public Sub New()
    MyBase.New()

    'This call is required by the Windows Form Designer.
    InitializeComponent()

    'Add any initialization after the InitializeComponent() call

End Sub

'Form overrides dispose to clean up the component list.
Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)
    If disposing Then
        If Not (components Is Nothing) Then
            components.Dispose()
        End If
    End If
    MyBase.Dispose(disposing)
End Sub

'Required by the Windows Form Designer
Private components As System.ComponentModel.IContainer

'NOTE: The following procedure is required by the Windows Form Designer
'It can be modified using the Windows Form Designer.
'Do not modify it using the code editor.
Friend WithEvents stbStatus As System.Windows.Forms.StatusBar
Friend WithEvents NotifyIcon1 As System.Windows.Forms.NotifyIcon
Friend WithEvents btnStart As System.Windows.Forms.Button
Friend WithEvents btnStop As System.Windows.Forms.Button
Friend WithEvents btnExit As System.Windows.Forms.Button
Friend WithEvents lblInfo As System.Windows.Forms.Label
Friend WithEvents PicBox As System.Windows.Forms.PictureBox
Friend WithEvents Panel1 As System.Windows.Forms.Panel
<System.Diagnostics.DebuggerStepThrough()> Private Sub InitializeComponent()
    Me.components = New System.ComponentModel.Container
    Me.stbStatus = New System.Windows.Forms.StatusBar
    Me.NotifyIcon1 = New System.Windows.Forms.NotifyIcon(Me.components)
    Me.PicBox = New System.Windows.Forms.PictureBox
    Me.SuspendLayout()
    
    Me.stbStatus.Location = New System.Drawing.Point(0, 374)
    Me.stbStatus.Name = "stbStatus"
    Me.stbStatus.Size = New System.Drawing.Size(672, 24)
    Me.stbStatus.TabIndex = 0
    
    Me.NotifyIcon1.Text = "NotifyIcon1"
    Me.NotifyIcon1.Visible = True
    
    Me.btnStart.Location = New System.Drawing.Point(392, 232)
    Me.btnStart.Name = "btnStart"
    Me.btnStart.Size = New System.Drawing.Size(72, 32)
    Me.btnStart.TabIndex = 1
    Me.btnStart.Text = "&Start"
    
    Me.btnStop.Location = New System.Drawing.Point(488, 232)
    Me.btnStop.Name = "btnStop"
    Me.btnStop.Size = New System.Drawing.Size(72, 32)
    Me.btnStop.TabIndex = 2
    Me.btnStop.Text = "S&top"
    
    Me.btnExit.Location = New System.Drawing.Point(584, 232)
    Me.btnExit.Name = "btnExit"
    
    Me.ResumeLayout()
End Sub
Me.btnExit.Size = New System.Drawing.Size(72, 32)
Me.btnExit.TabIndex = 3
Me.btnExit.Text = "E&xit"

'MblInfo

Me.lblInfo.Location = New System.Drawing.Point(384, 24)
Me.lblInfo.Name = "lblInfo"
Me.lblInfo.Size = New System.Drawing.Size(272, 184)
Me.lblInfo.TabIndex = 4

'PicBox

Me.PicBox.Location = New System.Drawing.Point(168, 168)
Me.PicBox.Name = "PicBox"
Me.PicBox.Size = New System.Drawing.Size(32, 32)
Me.PicBox.TabIndex = 5
Me.PicBox.TabStop = False

'Panel1

Me.Panel1.Location = New System.Drawing.Point(0, 0)
Me.Panel1.Name = "Panel1"
Me.Panel1.TabIndex = 6

'frmGPSTrack

Me.AutoScale = False
Me.AutoScaleBaseSize = New System.Drawing.Size(5, 13)
Me.ClientSize = New System.Drawing.Size(672, 398)
Me.Controls.Add(Me.Panel1)
Me.Controls.Add(Me.lblInfo)
Me.Controls.Add(Me.btnExit)
Me.Controls.Add(Me.btnStop)
Me.Controls.Add(Me.btnStart)
Me.Controls.Add(Me.stbStatus)
Me.Controls.Add(Me.PicBox)
Me.Name = "frmGPSTrack"
Me.Text = "GPS Tracker 1.0"
Me.ResumeLayout(False)
End Sub
' Protected Overrides Sub OnPaint(ByVal paintEvent As PaintEventArgs)

' Graphics()
' Dim graphicsObject As Graphics

' graphicsObject = Panel1.CreateGraphics

' Dim BrushGreen As SolidBrush = New SolidBrush(Color.Green)
' Dim BrushYellow As SolidBrush = New SolidBrush(Color.Yellow)
' Dim BrushRed As SolidBrush = New SolidBrush(Color.Red)
' Dim PenBlack As Pen = New Pen(Color.Black)

' Dim brush As SolidBrush = New SolidBrush(Color.Black)

' Draw(Circles)
' graphicsObject.DrawLine(PenBlack, 93, 93, 164, 164)
' graphicsObject.DrawLine(PenBlack, 307, 93, 236, 164)
' graphicsObject.DrawLine(PenBlack, 164, 236, 93, 307)
' graphicsObject.DrawLine(PenBlack, 236, 236, 307, 307)

' Draw Truck Icon
' PicBox.Image = System.Drawing.Image.FromFile("c:\temp\Truck.bmp")
' PicBox.SizeMode = PictureBoxSizeMode.StretchImage
' PicBox.BorderStyle = BorderStyle.FixedSingle
' PicBox.BringToFront()

' PicBox.Height = 36
' PicBox.Width = 20

' PicBox.Location = New Point(190, 182)

' Draws basic lines for grid
' PenBlack.Width = 1

' graphicsObject.DrawLine(PenBlack, 93, 93, 164, 164)
' graphicsObject.DrawLine(PenBlack, 307, 93, 236, 164)
' graphicsObject.DrawLine(PenBlack, 164, 236, 93, 307)
' graphicsObject.DrawLine(PenBlack, 236, 236, 307, 307)
Private Sub frmGPSTrack_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
    'Set Com Port Settings

    'Set Other GPS Locations

    'Button Defaults

    btnStop.Enabled = False

    Dim g As Graphics = Panel1.CreateGraphics()

    Dim imgZones As Image = Image.FromFile("c:\temp\zones2.tif")
    g.DrawImage(imgZones, 0, 0, imgZones.Width, imgZones.Height)
    g.Dispose()

    'Graphics

End Sub

Private Sub btnStart_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnStart.Click

    'Test Port
    Dim PortTest As New Rs232
    Try
        If PortTest.IsPortAvailable(intPort) Then
            MessageBox.Show("Port Is Avaiable", "Port Check", MessageBoxButtons.OK, MessageBoxIcon.Information)
            stbStatus.Text = "Port is Avaiable"
        Else
            MessageBox.Show("Port Is NOT Avaiable", "Port Check", MessageBoxButtons.OK, MessageBoxIcon.Exclamation)
        End If
    Catch ex As Exception
        MessageBox.Show("Port Test FAILED", "Port Check", MessageBoxButtons.OK, MessageBoxIcon.Error)
    End Try

End Sub
End Try

'Open Port
stbStatus.ForeColor = Color.Black
Dim intPortOpen As Integer = 0

GPSData = New Rs232
Try
' Setup Info
With GPSData
  ' Set By User
  .Port = intPort
  .BaudRate = 4800
  .DataBit = 8
  .StopBit = 1

  ' Set but may be changed later
  .Parity = Rs232.DataParity.Parity_None
  .Timeout = 5000
  .BufferSize = 100
End With

' Initialize Port
GPSData.Open()

' Set DTR and RTS
GPSData.Dtr = True
GPSData.Rts = True

Catch EX As Exception
  MessageBox.Show(EX.Message, "Connection Error", MessageBoxButtons.OK)
  intPortOpen = 1
End Try

Finally

  If intPortOpen = 0 Then
    stbStatus.Text = "Com Port " & intPort & " is open."
    btnStart.Enabled = False
    btnStop.Enabled = True
  End If
End Try

'Start Data Collection
If thrDataCollect.ThreadState = ThreadState.Suspended Then
    thrDataCollect.Resume()
Else
    thrDataCollect.Start()
End If

If readThread.ThreadState = ThreadState.Running Then
    readThread.Start()
Else
    End If

End Sub

Private Sub btnExit_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnExit.Click
    'Abort all threads and exit program
    blnRestart = True
    blnStop = True

    If thrDataCollect.ThreadState = ThreadState.Suspended Then
        thrDataCollect.Resume()
    End If
    thrDataCollect.Abort()
    Me.Close()
End Sub

Private Sub btnStop_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnStop.Click
    'Stop Threads
    'Reset Buttons
    btnStart.Enabled = True
    btnStop.Enabled = False

    'Display Stop Message
    stbStatus.Text = "Data Collection Has Stopped."
    blnStop = True
GPSData.Close()

End Sub

Sub DataRead()

Do Until blnRestart = True
    Do Until blnStop = True
        Try
            'Get GPS Data from Local Vehicle
            strGPSData = DataCollect.GPSPass(GPSData)

            'Process The Data Local Data into Usable Information Only
            strProcessedGPSData = ProcessData.ProcessLocal(strGPSData(0), strGPSData(1), strLocalID)
            ' Returns (ID,DATE,TIME,LAT,ALT,DIRECTION,SPEED)
            strSPDDIR(0) = strProcessedGPSData(6)
            strSPDDIR(1) = strProcessedGPSData(7)

            'Get GPS Data from Other Vehicles
            '---XX,DATE,TIME,V1,LAT,ALT,V2,LAT,ALT
            '---DATE is the GPS Date ddmmyy
            '---TIME is the 6 digit GPS Time to nearest second
            '---XX Is the number of vehicles
            '---V1 and V2 are the vehicle IDs
            '---LAT,ALT are the coordinates of the vehicles.
            '---There are coordinates for each vehicle
            strGPSData2 = DataCollect2.GPSPass2() 'Use only for pseudo data
            strGPSData2 = message

            'Convert to ECEFxyz for all GPS Data
            ' For Local

            strECEFLocal = ProcessData.Convert2ECEF(strProcessedGPSData)

            ' For Other Vehicles
            ' The Other Vehicle Data Must Be Parsed from a String
            intNumOfVehicles = CInt(strGPSData2.Chars(0) & strGPSData2.Chars(1))
ReDim strECEFOther(intNumOfVehicles - 1, 5)

strTemp = strGPSData2.Split(",")

For intCount = 1 To intNumOfVehicles
    strECEFOtherPass(0) = strTemp((4 * intCount) - 1)
    strECEFOtherPass(1) = strTemp(1)
    strECEFOtherPass(2) = strTemp(2)
    strECEFOtherPass(3) = strTemp(4 * intCount)
    strECEFOtherPass(4) = strTemp(4 * intCount + 1)
    strECEFOtherPass(5) = strTemp(4 * intCount + 2)

    strECEFCalculated = ProcessData.Convert2ECEF(strECEFOtherPass)

    strECEFOther(intCount - 1, 0) = strECEFCalculated(0)
    strECEFOther(intCount - 1, 1) = strECEFCalculated(1)
    strECEFOther(intCount - 1, 2) = strECEFCalculated(2)
    strECEFOther(intCount - 1, 3) = strECEFCalculated(3)
    strECEFOther(intCount - 1, 4) = strECEFCalculated(4)
    strECEFOther(intCount - 1, 5) = strECEFCalculated(5)

Next

"Calculate Distances between XYZ of Local and Other vehicles"
ReDim strDistance(intNumOfVehicles - 1)

For intCount = 0 To intNumOfVehicles - 1
    x2 = (CDbl(strECEFLocal(3)) - CDbl(strECEFOther(intCount, 3))) ^ 2
    y2 = (CDbl(strECEFLocal(4)) - CDbl(strECEFOther(intCount, 4))) ^ 2
    z2 = (CDbl(strECEFLocal(5)) - CDbl(strECEFOther(intCount, 5))) ^ 2

    strDistance(intCount) = Math.Sqrt(x2 + y2 + z2)
    strDistance(intCount) = Math.Sqrt(x2 + y2) ' + z2) Use X and Y Only

Next

lblInfo.ResetText()
' Writes The Calculated info to the Label
For intCount = 0 To intNumOfVehicles - 1
    lblInfo.Text = lblInfo.Text & strDistanceTo & strECEFOther(intCount, 0)
    & " is " & strDistance(intCount) & " meters." & ChrW(10)
Next

' Calls Function to Redraw Positions of Other Vehicles
' Need to Fix Still 1/24/05
Visual_Calc(strECEFLocal, strECEFOther, strDistance, strSPDDIR, intNumOfVehicles)
Catch Ex As Exception
    MessageBox.Show(Ex.Message, "ERROR", MessageBoxButtons.OK)
End Try

Loop
    thrDataCollect.Suspend()
    blnStop = False
End Loop

End Sub

Sub Visual_Calc(ByVal strECEFLocal As Array, ByVal strECFOther As Array,
    ByVal strDistance As Array, ByVal strSPDDIR As Array, ByVal intVehicles As Integer)
    'This Sub Updates the Location of the other vehicles on the main form
    'Date: 1/17/2005
    'Inputs local Data Array
    'Inputs the ID, Date, TIME, X, Y, Z and then Distance from Local for each vehicle

    'Graphics
    Dim g As Graphics = Panel1.CreateGraphics()

    'Clears the Panel
    g.Clear(Me.BackColor)

    'Draw Background Image
    Dim imgZones As Image = Image.FromFile("c:\temp\zones2.tif")
    g.DrawImage(imgZones, 25, 25, imgZones.Width, imgZones.Height)

    'For other Truck Locations
    Dim strVehicleLoc(intVehicles - 1, 4) As String 'ID, Screen X, Screen Y, Dist(r),
    Theta(degrees)
    Dim i As Integer = 0
    Dim D2 As Double 'Used to calculate the angle
    Dim x As Integer
    Dim y As Integer
    Dim BlackPen As Pen = New Pen(Color.Black)
    Dim Bluebrush As SolidBrush = New SolidBrush(Color.Blue)
    Dim Blackbrush As SolidBrush = New SolidBrush(Color.Black)
'Font For North Arrow
Dim arial As Font = New Font(New FontFamily("Arial"), 12, FontStyle.Bold)

If CDbl(strSPDDIR(1)) > 1 Then ' Change this later to a speed of greater than 1

'If the speed is greater than 1, then change the direction of North Arrow.

'Change the North Arrow Display
y = 165 - CInt(150 * Math.Cos(CDbl(strSPDDIR(0))))
x = 165 - CInt(150 * Math.Sin(CDbl(strSPDDIR(0))))

g.DrawString("N", arial, Blackbrush, x, y)

End If

For i = 0 To intVehicles - 1
strVehicleLoc(i, 0) = strECEFOther(i, 0) 'ID
strVehicleLoc(i, 1) = CStr(CDbl(strECEFLocal(3)) - CDbl(strECEFOther(i, 3)))
'X in screen coords
strVehicleLoc(i, 2) = CStr(CDbl(strECEFLocal(4)) - CDbl(strECEFOther(i, 4)))
'Y in screen coords
strVehicleLoc(i, 3) = strDistance(i)
Console.WriteLine(Math.Sqrt(CDbl(strVehicleLoc(i, 1) ^ 2 +
CDbl(strVehicleLoc(i, 2) ^ 2))))

D2 = Math.Sqrt(CDbl(strVehicleLoc(i, 1)) ^ 2 + (CDbl(strVehicleLoc(i, 2) +
150)) ^ 2)

strVehicleLoc(i, 4) = CStr((180 / 3.14) * (Math.Acos(((22500 - D2 ^ 2 +
CDbl(strDistance(i)) ^ 2)) / (2 * 150 * CDbl(strDistance(i))))) * 2))'Converts from radians to

Degrees

'Rotate Theta according to the direction of the local and the speed involved.
'Then convert from polar back to local screen x,y.  R must not be greater than
150m on the screen.

If CDbl(strSPDDIR(1)) > 1 Then ' Change this later to a speed of greater than 1

'If the speed is greater then 1, then change the direction.
strVehicleLoc(i, 4) = CStr(CDbl(strVehicleLoc(i, 4)) + CDbl(strSPDDIR(0)))
'If the Distance is Greater then 150, set it to 150 for display
If strVehicleLoc(i, 3) < 150 Then
    x = 175 - CDbl(strVehicleLoc(i, 3)) * Math.Cos(CDbl(strVehicleLoc(i, 4)))
    y = 175 - CDbl(strVehicleLoc(i, 3)) * Math.Sin(CDbl(strVehicleLoc(i, 4)))
Else
    x = 175 - CInt(150 * Math.Cos(CDbl(strVehicleLoc(i, 4))))
    y = 175 - CInt(150 * Math.Sin(CDbl(strVehicleLoc(i, 4))))
End If

'Draws the Ellipse for the Vehicle Location
g.DrawEllipse(BlackPen, x - 5, y - 5, 10, 10)
g.FillEllipse(Bluebrush, x - 5, y - 5, 10, 10)

End If

Next

g.Dispose()

End Sub

Public Sub RunClient()
    'Reads data sent from Server
    Dim client As TcpClient
    Dim IPAdd As IPAddress
    IPAdd = IPAddress.Parse("192.168.1.100")
    Try
        stbStatus.Text = "Connecting" & vbCrLf
        client = New TcpClient
        client.Connect(IPAdd, 4000)
        output = client.GetStream()
        writer = New BinaryWriter(output)
        reader = New BinaryReader(output)
    End Try
stbStatus.Text &= "Got I/O Streams" & vbCrLf
'txtSend.ReadOnly = False

Try
    Do
        message = reader.ReadString
        'stbStatus.Text &= message & vbCrLf
    Loop While blnStop2 = False

Catch inputOutPutexception As IOException
    MessageBox.Show("Client closing")

Finally
    stbStatus.Text &= "Closing Connection."
    writer.Close()
    reader.Close()
    output.Close()
    client.Close()
End Try

Application.Exit()

Catch inputOutPutexception As IOException
    MessageBox.Show("Client closing")
End Try
End Sub

End Class

CR232.vb – By Corrado Cavalli – Used For Com Port Data Collection Only
*All code is freely redistributable

Imports System.Runtime.InteropServices
Imports System.Text
Imports System.Threading
Imports System.ComponentModel
Imports System.Windows.Forms

Imports System.Runtime.InteropServices
Imports System.Text
Imports System.Threading
Imports System.ComponentModel
Imports System.Windows.Forms
#Region "RS232"
Public Class Rs232 : Implements IDisposable
'===================================================
Module Rs232
Description : Class for handling RS232 communication with VB.Net
Created : 10/08/2001 - 8:45:25
Author : Corrado Cavalli (corrado@mvps.org)
WebSite : www.codeworks.it/net/index.htm
Notes :
'===================================================

* Revisions *

  02/12/2000  First internal alpha version built on framework beta1

  1st Public release Beta2 (10/08/2001)

  Rev.1 (28.02.2002)
  1. Added ResetDev, SetBreak and ClearBreak to the EscapeCommFunction constants
  2. Added the overloaded Open routine.
  3. Added the modem status routines, properties and enum.
  4. If a read times out, it now returns a EndOfStreamException (instead of a simple Exception).
  5. Compiled with VS.Net final

  Rev.2 (01.03.2002)
  Added Async support

  Rev.3 (07.04.2002)
  Minor bugs fixed

  Rev.3 (05/05/2002)
  Fixed BuildCommDCB problem

  Rev.4 (24/05/2002)
  Fixed problem with ASCII Encoding truncating 8th bit

  Rev.5 (27/05/2002)
  Added IDisposable / Finalize implementation
' Rev.6 (14/03/2003)
'   Fixed problem on DCB fields Initialization
'
' Rev.7 (26/03/2003)
'   Added XON/XOFF support
'
' Rev.8 (12/07/2003)
'    Added support to COM port number greater than 4
'
'   Rev.9 (15/07/2003)
'    Added CommEvent to detect incoming chars/events
'    Updated both Tx/Rx method from Non-Ovelapped to Overlapped mode
'    Removed unused Async methods and other stuff.
'
'   Rev.10 (21/07/2003)
'    Fixed incorrect character handling when using EnableEvents()
'
' Rev.11 (12/08/2003)
'   Fixed some bugs signaled by users
'
'   Rev.12 (01/09/2003)
'    Removed AutoReset of internal buffers and added PurgeBuffer() method
'
'   Rev.13 (02/09/2003)
'    Removed GetLastErrorUse in favour of Win32Exception()
'
'   Rev.14 (14/09/2003)
'    Added IsPortAvailable() function
'    Revised some API declaration
'    Fixed problem with Win98/Me OS
'
' Rev.15 (24/09/2003)
'   Fixed bug introduced on Rev.14
'
' Rev.16 (12/10/2003)
'   Added SetBreak/ClearBreak() methods
'
' Rev.17 (02/11/2003)
'   Fixed field on COMMCONFIG
'
'===================================================================
''// Class Members
   Private mhRS As IntPtr = New IntPtr(0)   '//' Handle to Com Port

Private miPort As Integer = 1   '//' Default is COM1
Private miTimeout As Int32 = 70   '//' Timeout in ms
Private miBaudRate As Int32 = 9600
Private meParity As DataParity = 0
Private meStopBit As DataStopBit = 0
Private miDataBit As Int32 = 8
Private miBufferSize As Int32 = 512 ' // Buffers size default to 512 bytes
Private mabtRxBuf As Byte() ' // Receive buffer
Private meMode As Mode ' // Class working mode
    Private moThreadTx As Thread
    Private moThreadRx As Thread
    Private moEvents As Thread
Private miTmpBytes2Read As Int32
Private meMask As EventMasks
Private mbDisposed As Boolean
    Private mbUseXonXoff As Boolean
    Private mbEnableEvents As Boolean
Private miBufThreshold As Int32 = 1
Private muOvlE As OVERLAPPED
Private muOvlW As OVERLAPPED
Private muOvlR As OVERLAPPED

#Region " Enums"
    ' // Parity Data
    Public Enum DataParity
        Parity_None = 0
        Parity_Odd
        Parity_Even
        Parity_Mark
    End Enum
    ' // StopBit Data
    Public Enum DataStopBit
        StopBit_1 = 1
        StopBit_2
    End Enum
    <Flags()> Public Enum PurgeBuffers
        RXAbort = &H2
        RXClear = &H8
        TxAbort = &H1
        TxClear = &H4
    End Enum
    Private Enum Lines
        SetRts = 3
        ClearRts = 4
        SetDtr = 5
        ClearDtr = 6
        ResetDev = 7 ' // Reset device if possible
SetBreak = 8 ' // Set the device break line.
ClearBreak = 9 ' // Clear the device break line.

'// Modem Status
<Flags()> Public Enum ModemStatusBits
    ClearToSendOn = &H10
    DataSetReadyOn = &H20
    RingIndicatorOn = &H40
    CarrierDetect = &H80
End Enum

'// Working mode
Public Enum Mode
    NonOverlapped
    Overlapped
End Enum

'// Comm Masks
<Flags()> Public Enum EventMasks
    RxChar = &H1
    RXFlag = &H2
    TxBufferEmpty = &H4
    ClearToSend = &H8
    DataSetReady = &H10
    CarrierDetect = &H20
    Break = &H40
    StatusError = &H80
    Ring = &H100
End Enum

#End Region
#Region "Structures"
<StructLayout(LayoutKind.Sequential, Pack:=1)> Private Structure DCB
    Public DCBlength As Int32
    Public BaudRate As Int32
    Public Bits1 As Int32
    Public wReserved As Int16
    Public XonLim As Int16
    Public XoffLim As Int16
    Public ByteSize As Byte
    Public Parity As Byte
    Public StopBits As Byte
    Public XonChar As Char
    Public XoffChar As Char
    Public ErrorChar As Char
    Public EofChar As Char
    Public EvtChar As Char
    Public wReserved2 As Int16

105
End Structure
<StructLayout(LayoutKind.Sequential, Pack:=1)> Private Structure COMMTIMEOUTS
  Public ReadIntervalTimeout As Int32
  Public ReadTotalTimeoutMultiplier As Int32
  Public ReadTotalTimeoutConstant As Int32
  Public WriteTotalTimeoutMultiplier As Int32
  Public WriteTotalTimeoutConstant As Int32
End Structure
<StructLayout(LayoutKind.Sequential, Pack:=8)> Private Structure COMMCONFIG
  Public dwSize As Int32
  Public wVersion As Int16
  Public wReserved As Int16
  Public dcbx As DCB
  Public dwProviderSubType As Int32
  Public dwProviderOffset As Int32
  Public dwProviderSize As Int32
  Public wcProviderData As Int16
End Structure
<StructLayout(LayoutKind.Sequential, Pack:=1)> Public Structure OVERLAPPED
  Public Internal As Int32
  Public InternalHigh As Int32
  Public Offset As Int32
  Public OffsetHigh As Int32
  Public hEvent As IntPtr
End Structure
<StructLayout(LayoutKind.Sequential, Pack:=1)> Private Structure COMSTAT
  Dim fBitFields As Int32
  Dim cbInQue As Int32
  Dim cbOutQue As Int32
End Structure

#Region "Constants"
Private Const PURGE_RXABORT As Integer = &H2
Private Const PURGE_RXCLEAR As Integer = &H8
Private Const PURGE_TXABORT As Integer = &H1
Private Const PURGE_TXCLEAR As Integer = &H4
Private Const GENERIC_READ As Integer = &H80000000
Private Const GENERIC_WRITE As Integer = &H40000000
Private Const OPEN_EXISTING As Integer = 3
Private Const INVALID_HANDLE_VALUE As Integer = -1
Private Const IO_BUFFER_SIZE As Integer = 1024
Private Const FILE_FLAG_OVERLAPPED As Int32 = &H40000000
Private Const ERROR_IO_PENDING As Int32 = 997
Private Const WAIT_OBJECT_0 As Int32 = 0
Private Const ERROR_IO_INCOMPLETE As Int32 = 996
Private Const WAIT_TIMEOUT As Int32 = &H102&
Private Const INFINITE As Int32 = &HFFFFFFFF

#End Region
#Region "Win32API"
'// Win32 API
<DllImport("kernel32.dll", SetLastError:=True)> Private Shared Function
SetCommState(ByVal hCommDev As IntPtr, ByRef lpDCB As DCB) As Int32
End Function
<DllImport("kernel32.dll", SetLastError:=True)> Private Shared Function
GetCommState(ByVal hCommDev As IntPtr, ByRef lpDCB As DCB) As Int32
End Function
<DllImport("kernel32.dll", SetLastError:=True, CharSet:=CharSet.Auto)> Private
Shared Function BuildCommDCB(ByVal lpDef As String, ByRef lpDCB As DCB) As Int32
End Function
<DllImport("kernel32.dll", SetLastError:=True)> Private Shared Function
SetupComm(ByVal hFile As IntPtr, ByVal dwInQueue As Int32, ByVal dwOutQueue As
Int32) As Int32
End Function
<DllImport("kernel32.dll", SetLastError:=True)> Private Shared Function
GetCommTimeouts(ByVal hFile As IntPtr, ByRef lpCommTimeouts As COMMTIMEOUTS) As Int32
End Function
<DllImport("kernel32.dll", SetLastError:=True)> Private Shared Function
ClearCommError(ByVal hFile As IntPtr, ByRef lpErrors As Int32, ByRef lpComStat As
COMSTAT) As Int32
End Function
<DllImport("kernel32.dll", SetLastError:=True)> Private Shared Function
PurgeComm(ByVal hFile As IntPtr, ByVal dwFlags As Int32) As Int32
End Function
<DllImport("kernel32.dll", SetLastError:=True)> Private Shared Function
EscapeCommFunction(ByVal hFile As IntPtr, ByVal ifunc As Long) As Boolean
End Function
<DllImport("kernel32.dll", SetLastError:=True)> Private Shared Function
WaitCommEvent(ByVal hFile As IntPtr, ByRef Mask As EventMasks, ByRef lpOverlap
As OVERLAPPED) As Int32
End Function
<DllImport("kernel32.dll", SetLastError:=True)> Private Shared Function
WriteFile(ByVal hFile As IntPtr, ByVal Buffer As Byte(), ByVal
nNumberOfBytesToWrite As Integer, ByRef lpNumberOfBytesWritten As Integer, ByRef lpOverlapped As OVERLAPPED) As Integer
End Function

<DllImport("kernel32.dll", SetLastError:=True)> Private Shared Function ReadFile(ByVal hFile As IntPtr, <Out()> ByVal Buffer As Byte(), ByVal nNumberOfBytesToRead As Integer, ByRef lpNumberOfBytesRead As Integer, ByRef lpOverlapped As OVERLAPPED) As Integer
End Function

<DllImport("kernel32.dll", SetLastError:=True, CharSet:=CharSet.Auto)> Private Shared Function CreateFile(ByVal lpFileName As String, ByVal dwDesiredAccess As Integer, ByVal dwShareMode As Integer, ByVal lpSecurityAttributes As Integer, ByVal dwCreationDisposition As Integer, ByVal dwFlagsAndAttributes As Integer, ByVal hTemplateFile As Integer) As IntPtr
End Function

<DllImport("kernel32.dll", SetLastError:=True)> Private Shared Function CloseHandle(ByVal hObject As IntPtr) As Boolean
End Function

<DllImport("kernel32.dll", SetLastError:=True)> Public Shared Function GetCommModemStatus(ByVal hFile As IntPtr, ByRef lpModemStatus As Int32) As Boolean
End Function

<DllImport("kernel32.dll", SetLastError:=True)> Private Shared Function SetEvent(ByVal hEvent As IntPtr) As Boolean
End Function

<DllImport("kernel32.dll", SetLastError:=True, CharSet:=CharSet.Auto)> Private Shared Function CreateEvent(ByVal lpEventAttributes As IntPtr, ByVal bManualReset As Integer, ByVal bInitialState As Integer, ByVal lpName As String) As IntPtr
End Function

<DllImport("kernel32.dll", SetLastError:=True)> Private Shared Function WaitForSingleObject(ByVal hHandle As IntPtr, ByVal dwMilliseconds As Int32) As Int32
End Function

<DllImport("kernel32.dll", SetLastError:=True)> Private Shared Function GetOverlappedResult(ByVal hFile As IntPtr, ByRef lpOverlapped As OVERLAPPED, ByRef lpNumberOfBytesTransferred As Int32, ByVal bWait As Int32) As Int32
End Function

<DllImport("kernel32.dll", SetLastError:=True)> Private Shared Function SetCommMask(ByVal hFile As IntPtr, ByVal lpEvtMask As Int32) As Int32
End Function

<DllImport("kernel32.dll", SetLastError:=True, CharSet:=CharSet.Auto)> Private Shared Function GetDefaultCommConfig(ByVal lpName As String, ByRef lpCC As COMMCONFIG, ByRef lpdwSize As Integer) As Boolean
End Function

<DllImport("kernel32.dll", SetLastError:=True)> Private Shared Function SetCommBreak(ByVal hFile As IntPtr) As Boolean
End Function
<DllImport("kernel32.dll", SetLastError:=True)> Private Shared Function ClearCommBreak(ByVal hFile As IntPtr) As Boolean
End Function

#Region "Events"
Public Event CommEvent As CommEventHandler
#End Region
#Region "Delegates"
Public Delegate Sub CommEventHandler(ByVal source As Rs232, ByVal Mask As EventMasks)
#End Region

Public Property Port() As Integer
    '===================================================
    ' Description : Comunication Port
    ' Created : 21/09/2001 - 11:25:49
    ' *Parameters Info*
    ' Notes :
    '===================================================
    Get
        Return miPort
    End Get
    Set(ByVal Value As Integer)
        miPort = Value
    End Set
End Property

Public Sub PurgeBuffer(ByVal Mode As PurgeBuffers)
    '=======================================
    ' Description : Purge Communication Buffer
    ' Created : 01/09/03 - 10:37:39
    ' Author : Corrado Cavalli
    ' *Parameters Info*
    ' Notes : This method will clear any character into buffer, use TxAbort/RxAbort
    '©2003 ALSTOM FIR S.p.A All rights reserved
any pending overlapped Tx/Rx operation.

If (mhRS.ToInt32 <> 0) Then PurgeComm(mhRS, Mode)

Public Overridable Property Timeout() As Integer

Description: Comunication timeout in seconds
Created: 21/09/2001 - 11:26:50

*Parameters Info*

Notes:

Get
Return miTimeout
End Get
Set(ByVal Value As Integer)
    miTimeout = CInt(IIf(Value = 0, 500, Value))
    '// If Port is open updates it on the fly
    pSetTimeout()
End Set

End Property

Public Property Parity() As DataParity

Description: Comunication parity
Created: 21/09/2001 - 11:27:15

*Parameters Info*

Notes:

Get
Return meParity
End Get
Set(ByVal Value As DataParity)
    meParity = Value
End Set

End Property

Public Property StopBit() As DataStopBit
Description: Comunication StopBit
Created : 21/09/2001

Get
Return meStopBit
End Get
Set(ByVal Value As DataStopBit)
   meStopBit = Value
End Set
End Property

Public Property BaudRate() As Integer

Description: Comunication BaudRate
Created : 21/09/2001

Get
Return miBaudRate
End Get
Set(ByVal Value As Integer)
   miBaudRate = Value
End Set
End Property

Public Property DataBit() As Integer

Description : Comunication DataBit
Created : 21/09/2001
Public Property miDataBit() As Integer
    Get
        Return miDataBit
    End Get
    Set(ByVal Value As Integer)
        miDataBit = Value
    End Set
End Property

Public Property BufferSize() As Integer
    Get
        Return miBufferSize
    End Get
    Set(ByVal Value As Integer)
        miBufferSize = Value
    End Set
End Property

Public Overloads Sub Open()
    Dim uDcb As DCB, iRc As Int32
    meMode = Mode.Overlapped
Dim iMode As Int32 = Convert.ToInt32(IIf(meMode = Mode.Overlapped, FILE_FLAG_OVERLAPPED, 0))
'// Initializes Com Port
If miPort > 0 Then
    Try
        '// Creates a COM Port stream handle
        mhRS = CreateFile("\\.\COM" & miPort.ToString, GENERIC_READ Or GENERIC_WRITE, 0, 0, OPEN_EXISTING, iMode, 0)
        If (mhRS.ToInt32 <> 0) Then
            '// Clear all communication errors
            Dim lpErrCode As Int32
            iRc = ClearCommError(mhRS, lpErrCode, New COMSTAT)
            '// Clears I/O buffers
            iRc = PurgeComm(mhRS, PurgeBuffers.RXClear Or PurgeBuffers.TxClear)
            '// Gets COM Settings
            iRc = GetCommState(mhRS, uDcb)
            '// Updates COM Settings
            Dim sParity As String = "NOEM"
            sParity = sParity.Substring(meParity, 1)
            '// Set DCB State
            Dim sDCBState As String = String.Format("baud={0} parity={1} data={2} stop={3}", miBaudRate, sParity, miDataBit, CInt(meStopBit))
            iRc = BuildCommDCB(sDCBState, uDcb)
            uDcb.Parity = CByte(meParity)
            '// Set Xon/Xoff State
            If mbUseXonXoff Then
                uDcb.Bits1 = 768
            Else
                uDcb.Bits1 = 0
            End If
            iRc = SetCommState(mhRS, uDcb)
            If iRc = 0 Then
                Dim sErrTxt As String = New Win32Exception().Message
                Throw New CIOChannelException("Unable to set COM state " & sErrTxt)
            End If
        End If
    Catch
        '// Setup Buffers (Rx,Tx)
        iRc = SetupComm(mhRS, miBufferSize, miBufferSize)
    End Try
    '// Set Timeouts
    pSetTimeout()
    '// Enables events if required
If mbEnableEvents Then Me.EnableEvents()
Else
  '// Raise Initialization problems
  '//Throw New CIOChannelException("Unable to open COM" & miPort.ToString)
  Throw New Win32Exception
End If
Catch Ex As Exception
  '// Generica error
  Throw New CIOChannelException(Ex.Message, Ex)
End Try
Else
  '// Port not defined, cannot open
  Throw New ApplicationException("COM Port not defined, use Port property to set it before invoking InitPort")
End If
End Sub

Public Overloads Sub Open(ByVal Port As Integer, ByVal BaudRate As Integer, ByVal DataBit As Integer, ByVal Parity As DataParity, ByVal StopBit As DataStopBit, ByVal BufferSize As Integer)
  '=================================================================
  'Description:       Opens comunication port
  'Created            :    21/09/2001 - 11:33:40
  '*Parameters Info*
  'Notes
  '=================================================================
  Me.Port = Port
  Me.BaudRate = BaudRate
  Me.DataBit = DataBit
  Me.Parity = Parity
  Me.StopBit = StopBit
  Me.BufferSize = BufferSize
  Open()
End Sub

Public Sub Close()
  '=================================================================
  'Description:       Close comunication channel
  'Created            :    21/09/2001 - 11:38:00
  '=================================================================
If mhRS.ToInt32 <> 0 Then
    If mbEnableEvents = True Then
        Me.DisableEvents()
    End If
    Dim ret As Boolean = CloseHandle(mhRS)
    If Not ret Then Throw New Win32Exception
    mhRS = New IntPtr(0)
End If
End Sub

ReadOnly Property IsOpen() As Boolean

    Description: Returns Port Status
    Created: 21/09/2001 - 11:38:51

Get
    Return CBool(mhRS.ToInt32 <> 0)
End Get

End Property

Public Overloads Sub Write(ByVal Buffer As Byte())

    Description: Transmit a stream
    Created: 21/09/2001 - 11:39:51

Dim iRc, iBytesWritten As Integer

muOvlW = New Overlapped
If mhRS.ToInt32 = 0 Then
Throw New ApplicationException("Please initialize and open port before using this method")

Else

    '// Creates Event
    Try
        muOvlW.hEvent = CreateEvent(Nothing, 1, 0, Nothing)
        If muOvlW.hEvent.ToInt32 = 0 Then Throw New ApplicationException("Error creating event for overlapped writing")
        '// Clears IO buffers and sends data
        iRc = WriteFile(mhRS, Buffer, Buffer.Length, 0, muOvlW)
        If iRc = 0 Then
            If Marshal.GetLastWin32Error <> ERROR_IO_PENDING Then
                Throw New ApplicationException("Write command error")
            Else
                '// Check Tx results
                If GetOverlappedResult(mhRS, muOvlW, iBytesWritten, 1) = 0 Then
                    Throw New ApplicationException("Write pending error")
                Else
                    '// All bytes sent?
                    If iBytesWritten <> Buffer.Length Then
                        Throw New ApplicationException("Write Error - Bytes Written " & iBytesWritten.ToString & " of " & Buffer.Length.ToString)
                    End If
                End If
            End If
        End If
    End Try

Finally

    '// Closes handle
    CloseHandle(muOvlW.hEvent)

End Try

End Sub

Public Overloads Sub Write(ByVal Buffer As String)

    '===================================================
    ' Description : Writes a string to RS232
    ' Created      : 04/02/2002 - 8:46:42
    '
    *Parameters Info*
    '
    Notes         : 24/05/2002 Fixed problem with ASCII Encoding

    '====================================================
Dim oEncoder As New System.Text.ASCIIEncoding
Dim oEnc As Encoding = oEncoder.GetEncoding(1252)
'-------------------------------------------------------------
Dim aByte() As Byte = oEnc.GetBytes(Buffer)
Me.Write(aByte)
End Sub

Public Function Read(ByVal Bytes2Read As Integer) As Integer
'===================================================
'*Parameters Info*
' Bytes2Read : Bytes to read from port
' Returns : Number of readed chars
'Notes : '===================================================
Dim iReadChars, iRc As Integer, bReading As Boolean
'-------------------------------------------------------------
'// If Bytes2Read not specified uses Buffersize
If Bytes2Read = 0 Then Bytes2Read = miBufferSize
muOvlR = New Overlapped
If mhRS.ToInt32 = 0 Then
    Throw New ApplicationException("Please initialize and open port before using this method")
Else
    '// Get bytes from port
    Try
        muOvlR.hEvent = CreateEvent(Nothing, 1, 0, Nothing)
        If muOvlR.hEvent.ToInt32 = 0 Then Throw New ApplicationException("Error creating event for overlapped reading")
        '// Clears IO buffers and reads data
        ReDim mabtRxBuf(Bytes2Read - 1)
        iRc = ReadFile(mhRS, mabtRxBuf, Bytes2Read, iReadChars, muOvlR)
        If iRc = 0 Then
            If Marshal.GetLastWin32Error() <> ERROR_IO_PENDING Then
                Throw New ApplicationException("Read pending error")
            Else
                'ERROR_IO_PENDING Then
            End If
        End If
    End Try
End If
Else
End If
/\ Wait for characters
iRc =
WaitForSingleObject(muOvlR.hEvent, miTimeout)
Select Case iRc
Case WAIT_OBJECT_0
  '// Some data received...
  If
  GetOverlappedResult(mhRS, muOvlR, iReadChars, 0) = 0 Then
      Throw New ApplicationException("Read pending error.")
    Else
      Return iReadChars
    End If
Case WAIT_TIMEOUT
  Throw New IOTimeoutException("Read Timeout.")
Case Else
  Throw New ApplicationException("General read error.")
End Select
End If
Else
  Return (iReadChars)
End If
Finally
  '//Closes handle
  CloseHandle(muOvlR.hEvent)
End Try
End If
End Function
Overridable ReadOnly Property InputStream() As Byte()
',
'=====================================================
| Description: | Returns received data as Byte() |
| Created: | 21/09/2001 - 11:45:06 |
| Notes: | |
'================================================================
Get
    Return mabtRxBuf
End Get
End Property
Overridable ReadOnly Property InputStreamString() As String

'===================================================
' Description : Return a string containing received data
' Created : 04/02/2002 - 8:49:55
' *Parameters Info*
' Notes :
'===================================================

Get
Dim oEncoder As New System.Text.ASCIIEncoding
Dim oEnc As Encoding = oEncoder.GetEncoding(1252)
'===================================================
Return oEnc.GetString(Me.InputStream)
End Get
End Property

Public Sub ClearInputBuffer()

'===================================================
' Description: Clears Input buffer
' Created : 21/09/2001 - 11:45:34
' *Parameters Info*
' Notes : Gets all character until end of buffer
'===================================================

If Not mhRS.ToInt32 = 0 Then
    PurgeComm(mhRS, PURGE_RXCLEAR)
End If
End Sub

Public WriteOnly Property Rts() As Boolean

'===================================================
' Description: Set/Resets RTS Line
' Created : 21/09/2001 - 11:45:34
' *Parameters Info*
' Notes :
'===================================================
Set(ByVal Value As Boolean)
    If Not mhRS.ToInt32 = 0 Then
        If Value Then
            EscapeCommFunction(mhRS, Lines.SetRts)
        Else
            EscapeCommFunction(mhRS, Lines.ClearRts)
        End If
    End If
End Set
End Property

Public WriteOnly Property Dtr() As Boolean
    '===================================================
    ' Description: Set/Resets DTR Line
    ' Created : 21/09/2001 - 11:45:34
    ' *Parameters Info*
    ' Notes :
    '===================================================
    Set(ByVal Value As Boolean)
        If Not mhRS.ToInt32 = 0 Then
            If Value Then
                EscapeCommFunction(mhRS, Lines.SetDtr)
            Else
                EscapeCommFunction(mhRS, Lines.ClearDtr)
            End If
        End If
End Set
End Property

Public ReadOnly Property ModemStatus() As ModemStatusBits
    '===================================================
    ' Description : Gets Modem status
    ' Created : 28/02/2002 - 8:58:04
    ' *Parameters Info*
    ' Notes :
    '===================================================
    Get
        If mhRS.ToInt32 = 0 Then
            Throw New ApplicationException("Please initialize and open port before using this method")
        End If
End Get
Else
    '// Retrieve modem status
    Dim lpModemStatus As Int32
    If Not GetCommModemStatus(mhRS, lpModemStatus)
    Then
        Throw New ApplicationException("Unable to get modem status")
    Else
        Return CType(lpModemStatus, ModemStatusBits)
    End If
End If
End Get
End Property

Public Function CheckLineStatus(ByVal Line As ModemStatusBits) As Boolean
    '===================================================
    ' Description : Check status of a Modem Line
    ' Created : 28/02/2002 - 10:25:17
    ' *Parameters Info*
    ' Notes : 
    '===================================================
    Return Convert.ToBoolean(ModemStatus And Line)
End Function

Public Property UseXonXoff() As Boolean
    '===================================================
    ' Description : Set XON/XOFF mode
    ' Created : 26/05/2003 - 21:16:18
    ' *Parameters Info*
    ' Notes :
    '===================================================
    Get
        Return mbUseXonXoff
    End Get
    Set(ByVal Value As Boolean)
        mbUseXonXoff = Value
    End Set
End Property

Public Sub EnableEvents()
    '===================================================
    ' Description : Enables monitoring of incoming events

121
If mhRS.ToInt32 = 0 Then
    Throw New ApplicationException("Please initialize and open port before using this method")
Else
    If moEvents Is Nothing Then
        mbEnableEvents = True
        moEvents = New Thread(AddressOf pEventsWatcher)
        moEvents.Start()
    End If
End If
End Sub
Public Sub DisableEvents()
    If mbEnableEvents = True Then
        SyncLock Me
        mbEnableEvents = False
    End SyncLock
    This should kill the thread
    '// Let WaitCommEvent exit...
    If muOvlE.hEvent.ToInt32 <> 0 Then SetEvent(muOvlE.hEvent)
    moEvents = Nothing
End If
End Sub
Public Property RxBufferThreshold() As Int32
    Description : Numer of characters into input buffer
    Created : 16/07/03 - 9:00:57
    Author : Corrado Cavalli
Public Function IsPortAvailable(ByVal portNumber As Int32) As Boolean

    Dim cfg As COMMCONFIG
    Dim cfgsize As Int32 = Marshal.SizeOf(cfg)
    cfg.dwSize = cfgsize
    Dim ret As Boolean = GetDefaultCommConfig("COM" + portNumber.ToString, cfg, cfgsize)
    Return ret

End Function

Public Sub SetBreak()

    ©2003 www.codeworks.it All rights reserved

    ' Description : Set COM in break modem
    ' Created : 12/10/03 - 10:00:57
    ' Author : Corrado Cavalli

End Sub
Public Sub ClearBreak()
    If Not mhRS.ToInt32 = 0 Then
        If SetCommBreak(mhRS) = False Then Throw New Win32Exception
    End If
End Sub

#Region "Finalize"
Protected Overrides Sub Finalize()
    If Not mhRS.ToInt32 = 0 Then
        If ClearCommBreak(mhRS) = False Then Throw New Win32Exception
    End If
End Sub
If Not mbDisposed Then
  If mbEnableEvents Then Me.DisableEvents()
  Close()
End If
Finally
  MyBase.Finalize()
End Try
End Sub
#End Region

#Region "Private Routines"
Private Sub pSetTimeout()
'===================================================
  Description:  Set communication timeouts
  Created:      21/09/2001 - 11:46:40

  *Parameters Info*

  Notes:
'===================================================

Dim uCtm As COMMTIMEOUTS
'// Set ComTimeout
If mhRS.ToInt32 = -1 Then
  Exit Sub
Else
  '// Changes setup on the fly
  With uCtm
    .ReadIntervalTimeout = 0
    .ReadTotalTimeoutMultiplier = 0
    .ReadTotalTimeoutConstant = miTimeout
    .WriteTotalTimeoutMultiplier = 10
    .WriteTotalTimeoutConstant = 100
  End With
  SetCommTimeouts(mhRS, uCtm)
End If
End Sub

Private Sub pDispose() Implements IDisposable.Dispose
'===================================================
  Description:  Handles correct class disposing Write
  Created:      27/05/2002 - 19:03:06

  *Parameters Info*
'
Notes :
===================================================
If (Not mbDisposed AndAlso (mhRS.ToInt32 <> -1)) Then
    '// Closes Com Port releasing resources
    Try
        Me.Close()
    Finally
        mbDisposed = True
        '// Suppress unnecessary Finalize overhead
        GC.SuppressFinalize(Me)
    End Try
End If

End Sub
Private Sub pEventsWatcher()
    '===================================================
    '©2003 www.codeworks.it All rights reserved
    'Description : Watches for all events raising events when they arrive to the port
    'Created : 15/07/03 - 11:45:13
    'Author : Corrado Cavalli
    'Notes :
    '===================================================
    '// Events to watch
        EventMasks.StatusError
    Dim lRetMask As EventMasks, iBytesRead, iTotBytes, iErrMask As Int32, iRc As Int32, aBuf As New ArrayList
    Dim uComStat As COMSTAT
    '-----------------------------------
    '// Creates Event
    muOvlE.hEvent = CreateEvent(Nothing, 1, 0, Nothing)
    If muOvlE.hEvent.ToInt32 = 0 Then Throw New ApplicationException("Error creating event for overlapped reading")
    '// Set mask
    SetCommMask(mhRS, lMask)
    '// Looks for RxChar

126
While mbEnableEvents = True
    WaitCommEvent(mhRS, lMask, muOvlE)
Select Case WaitForSingleObject(muOvlE.hEvent, INFINITE)
    Case WAIT_OBJECT_0
        '// Event (or abort) detected
        If mbEnableEvents = False Then Exit While
        If (lMask And EventMasks.RxChar) > 0 Then
            '// Read incoming data
            ClearCommError(mhRS, iErrMask, uComStat)
            If iErrMask = 0 Then
                Dim ovl As New Overlapped
                ReDim mabtRxBuf(uComStat.cbInQue - 1)
                If ReadFile(mhRS, mabtRxBuf, uComStat.cbInQue, iBytesRead, ovl) > 0 Then
                    If iBytesRead > 0 Then
                        '// Some bytes read, fills temporary buffer
                        If iTotBytes < miBufThreshold Then
                            aBuf.AddRange(mabtRxBuf)
                            iTotBytes += iBytesRead
                        End If
                        '// Threshold reached?, raises event
                        If iTotBytes >= miBufThreshold Then
                            '// Copies temp buffer into Rx buffer
                            ReDim mabtRxBuf(iTotBytes - 1)
                            aBuf.CopyTo(mabtRxBuf)
                            '// Raises event
                            Try
                                Me.OnCommEventReceived(Me, lMask)
                            Finally
                                iTotBytes = 0
                                aBuf.Clear()
                            End Try
                        End If
                    End If
                End If
            Else
                '// Simply raises OnCommEventHandler event
                Me.OnCommEventReceived(Me, lMask)
            End If
        Case Else
            Dim sErr As String = New Win32Exception().Message
            Throw New ApplicationException(sErr)
        End Select
    End If
End While
    '// Release Event Handle
CloseHandle(muOvlE.hEvent)
muOvlE.hEvent = IntPtr.Zero

End Sub
#End Region

#Region "Protected Routines"
Protected Sub OnCommEventReceived(ByVal source As Rs232, ByVal mask As EventMasks)
'===================================================
' Description : Raises CommEvent
' Created : 15/07/03 - 15:09:50
' Author : Corrado Cavalli
' *Parameters Info*
' Notes :
'===================================================
RaiseEvent CommEvent(Me, mask)

End Sub
#End Region

End Class
#End Region

#Region "Exceptions"
Public Class CIOChannelException : Inherits ApplicationException
'===================================================
' Module : CChannelException
' Description: Customized Channel Exception
' Created : 17/10/2001 - 10:32:37
' Notes : This exception is raised when NACK error found
'===================================================
Sub New(ByVal Message As String)
    MyBase.New(Message)
End Sub
Sub New(ByVal Message As String, ByVal InnerException As Exception)
    MyBase.New(Message, InnerException)
End Sub

End Class
Public Class IOTimeoutException : Inherits CIOChannelException
Description : Timeout customized exception

Created : 28/02/2002 - 10:43:43

*Parameters Info*

Notes :

---
Sub New(ByVal Message As String)
    MyBase.New(Message)
End Sub
Sub New(ByVal Message As String, ByVal InnerException As Exception)
    MyBase.New(Message, InnerException)
End Sub
End Class

#End Region

---

Constants.vb – Used For ECEF Calculations

Public Class Constants
    Public Const mdblMu As Double = 398600500000000.0
    'meters^3/second^2
    Public Const mdblAA As Double = 6378137.0
    'meters
    Public Const mdblBB As Double = 6356752.31425
    'meters
    Public Const mdblEsquare As Double = (mdblAA ^ 2 - mdblBB ^ 2) / mdblAA ^ 2
    Public Const mdblOmegaE As Double = 0.00007292115
    'radians/second
    Public Const mdblC As Double = 299792458
    'meters/second
    Public Const mdblDgrad As Double = 3.14159265358979 / 180.0
    'seconds
    Public Const mintLeapSeconds As Integer = 13
    Public Const mdblF0 As Double = 10230000.0
    'Hertz
    Public Const mdblF As Double = 154 * mdblF0
    'Hertz
    Public Const mdblLambda As Double = mdblC / mdblF
    'meters
End Class
ECEF.vb – Calculates ECEF xyz from Latitude, Longitude, and Altitude

Imports System.Math
Module ECEF

' this GPS module converts a WGS-84 latitude-longitude-altitude
' position into ECEF coordinates
'
' input: WGS-84 coordinates are passed to ECEF.vb from the main form
' lat(deg), long(deg), alt(m)
'
' output: 'dblECEFcoords' Array which contains the same position
' specified by ECEF coordinates (meters)
'
Function ECEFxyz(ByVal dblLLA() As Double)
    Dim dblLatPass As Double = dblLLA(0)
    Dim dblLonPass As Double = dblLLA(1)
    Dim dblAltPass As Double = dblLLA(2)
    Dim dbllat As Double
    Dim dbllon As Double
    Dim dblNN As Double
    Dim dblECEFcoords(2) As Double
    ' convert to radians
    dbllat = dblLatPass * Constants.mdblDegrad   ' latitude in radians
    dbllon = dblLonPass * Constants.mdblDegrad   ' longitude in radians
    ' computes the ECEF coordinates
    dblNN = (Constants.mdblAA) ^ 2 / ((Constants.mdblAA * Cos(dbllat)) ^ 2 +
            (Constants.mdblBB / Constants.mdblAA) ^ 2 +
            (Constants.mdblBB * Sin(dbllat)) ^ 2) ^ 0.5
    dblECEFcoords(0) = (dblNN + dblAltPass) * Cos(dbllat) * Cos(dbllon)  ' meters
    dblECEFcoords(1) = (dblNN + dblAltPass) * Cos(dbllat) * Sin(dbllon)  ' meters
    dblECEFcoords(2) = (dblNN * (Constants.mdblBB / Constants.mdblAA) ^ 2 +
                        dblAltPass) * Sin(dbllat)  ' meters
    Return dblECEFcoords
End Function

End Module
DataCollect.vb – Collects and Processes the GPS Data

'Date: 01/10/05
'This Module Collects GPS Data and Stores it in a String to be passed on to be processed.
Module DataCollect

Function GPSPass(ByVal GPSData As Rs232) As Array

'Begin Data Collection

Dim chrChar As Char = Nothing
Dim intEndloop As Integer = 0
Dim intEndloop2 As Integer = 0
Dim intEndloop3 As Integer = 0
Dim strIsGPRMC As String = Nothing
Dim strIsGPGGA As String = Nothing
Dim strIsGPVTG As String = Nothing
Dim strNMEA As String = Nothing
Dim strRMCGGA(2) As String

'Write Program to Read Characters and Write them to a String only if it is the complete
'SGPRMC string. Discard all other information for now

'//New Code******
Try

GPSData.Read(1)
chrChar = GPSData.InputStreamString

'Reads in GPS Data Until it Gets GPGGA

Do Until intEndloop = 1
Do Until chrChar = ChrW(10)
strNMEA = strNMEA & CStr(chrChar)
GPSData.Read(1)
chrChar = GPSData.InputStreamString
Loop

If strNMEA.Chars(4) = "G" And strNMEA.Chars(0) = "$" Then
strIsGPGGA = strNMEA

GPSData.Read(1)
chrChar = GSData.InputStreamString
strNMEA = Nothing
intEndloop = 1

Else
    GSData.Read(1)
    chrChar = GSData.InputStreamString
    strNMEA = Nothing
End If

Loop

'Reads in until it gets GPRMC

Do Until intEndloop2 = 1
    Do Until chrChar = ChrW(10)
        strNMEA = strNMEA & CStr(chrChar)
        GSData.Read(1)
        chrChar = GSData.InputStreamString
    Loop

    If strNMEA.Chars(5) = "C" And strNMEA.Chars(0) = "$" Then
        strIsGPRMC = strNMEA

        GSData.Read(1)
        chrChar = GSData.InputStreamString
        strNMEA = Nothing
        intEndloop2 = 1
    Else
        GSData.Read(1)
        chrChar = GSData.InputStreamString
        strNMEA = Nothing
    End If

Loop

'Reads in until it gets GPVTG

Do Until intEndloop3 = 1
    Do Until chrChar = ChrW(10)
        strNMEA = strNMEA & CStr(chrChar)
        GSData.Read(1)
        chrChar = GSData.InputStreamString
    Loop
If strNMEA.Chars(5) = "G" And strNMEA.Chars(0) = "$" Then
    strIsGPVTG = strNMEA
    GPSData.Read(1)
    chrChar = GPSData.InputStreamString
    strNMEA = Nothing
    intEndloop3 = 1
Else
    GPSData.Read(1)
    chrChar = GPSData.InputStreamString
    strNMEA = Nothing
End If
Loop

' Loop
strRMCGGA(0) = strIsGPRMC
strRMCGGA(1) = strIsGPGGA
strRMCGGA(2) = strIsGPVTG

'Return the $GPRMC  and $GPGGA Sentences
GPSPass = strRMCGGA

Catch ex As Exception
    'There will be a read pending error. Can ignore for now.
End Try

Return GPSPass

End Function

End Module
Module ProcessData

Function ProcessLocal(ByVal strGPRMC As String, ByVal strGPGGA As String, ByVal strGPVTG As String, ByVal strLocalID As String) As Array

'Processes the GPRMC and GPGGA into one string with ID,DATE,TIME,LAT,LONG,ALT,DIRECTION,SPEED

Dim strLocalData(7) As String

'(ID,DATE,TIME,LAT,LONG,ALT,DIRECTION,SPEED
Dim strChar As String = Nothing
Dim strTemp As String
Dim strConvert As String
Dim count As Integer
Dim strGPRMC2() As String
Dim strGPGGA2() As String
Dim strGPVTG2() As String

Try

strLocalData(0) = strLocalID

'Split Sentences into Arrays
strGPRMC2 = strGPRMC.Split(,,)
strGPGGA2 = strGPGGA.Split(,,)
strGPVTG2 = strGPVTG.Split(,,)

'---Gets the GPS Time
strLocalData(2) = strGPRMC2(1)

If strGPRMC2(2) = "A" Then

'---Gets the Latitude from the Array

strLocalData(3) = strGPRMC2(3)

'---Converts from ddm.mmm to dd.dddd
strTemp = Nothing
strTemp = strLocalData(3).Remove(2, 7)
strConvert = strLocalData(3).Remove(0, 2)
strConvert = CStr((CDbl(strConvert)) / 60)
strLocalData(3) = strTemp & strConvert.Remove(0, 1)

'---Checks for N or S - Sets S equal to negative Latitude
If strGPRMC2(4) = "S" Then
    strLocalData(3) = CStr(CDb(strLocalData(3)) * -1)
End If

'---Gets the Longitude from the string
strLocalData(4) = strGPRMC2(5)

'---Converts from dddmm.mmm to ddd.dddd
strTemp = Nothing
strTemp = strLocalData(4).Remove(3, 7)
strConvert = strLocalData(4).Remove(0, 3)
strConvert = CStr((CDbl(strConvert)) / 60)
strLocalData(4) = strTemp & strConvert.Remove(0, 1)

'---Checks for E or W - Sets W equal to negative Longitude
If strGPRMC2(6) = "W" Then
    strLocalData(4) = CStr(CDb(strLocalData(4)) * -1)
End If

'--- Gets UTC Date
strLocalData(1) = strGPRMC2(9)

'---Gets Altitude from GPGGA
strLocalData(5) = strGPGGA2(9)

Else
    strLocalData(3) = "00.0000"
    strLocalData(4) = "000.0000"
    strLocalData(5) = "0.0"

End If

If strGPVTG2(1) = "" Then
    strLocalData(6) = "0.0"
    strLocalData(7) = "0.0"
Else
    strLocalData(6) = strGPVTG2(1)
    strLocalData(7) = strGPVTG2(5)
End If

Catch Ex As Exception

End Try

'Returns the Local Data (ID,DATE,TIME,LAT,LONG,ALT,DIRECTION,SPEED)
Return strLocalData

End Function

'Calculate

Function Convert2ECEF(ByVal strData() As String) As Array
    'Converts incoming ID,DATE,TIME,LAT,LONG,ALT to ID, Date, Time, X ,Y, Z
    Dim ID As String
    Dim datDate As String
    Dim Time As String
    Dim Latitude As String
    Dim Longitude As String
    Dim Altitude As String
    Dim dblLLA(2) As Double
    Dim dblXYZ(2) As Double
    Dim strDataReturn(5) As String

    ID = strData(0)
    datDate = strData(1)
    Time = strData(2)
    Latitude = strData(3)
    Longitude = strData(4)
    Altitude = strData(5)
dblLLA(0) = CDbl(Latitude)
dblLLA(1) = CDbl(Longitude)
dblLLA(2) = CDbl(Altitude)

dblXYZ = ECEF.ECEFxyz(dblLLA)

strDataReturn(0) = ID
strDataReturn(1) = datDate
strDataReturn(2) = Time
strDataReturn(3) = CStr(dblXYZ(0))
strDataReturn(4) = CStr(dblXYZ(1))
strDataReturn(5) = CStr(dblXYZ(2))

'Returns ID, Date, Time, X, Y, Z Array
Return strDataReturn

End Function

End Module

ActivateSystem.vb

' This Module Controls the Actions if the conditions are met for the system to be activated.

Module ActivateSystem

Function Lock(ByVal strDistance() As String) As Array

    Dim i As Integer = 0
    Dim j As Integer

    j = strDistance.Length

    Do Until i = j - 1

        If strDistance(i) < 25 Then
            'LOCK Transmission
            'Insert Method to send signal to lock transmission
        End If

        i = i + 1

    Do Until i = j - 1

        If strDistance(i) < 25 Then
            'LOCK Transmission
            'Insert Method to send signal to lock transmission
        End If

        i = i + 1

    End Function

End Module
Loop
End Function
End Module
APPENDIX D: LATLONGDIST.CAL VBA SCRIPT

Calculates the distance between two points in Latitude and Longitude by using the great circle method.

Dim a As Double
Dim b As Double
Dim c As Double
Dim PI As Double
Dim RadiusEarth As Double

PI = 3.14159265358979
RadiusEarth = 6371000


c = Sin( [truck1.LAT] * PI / 180) * Sin([truck2.LAT] * PI / 180)

If (a + b + c) >= 1 Or (a + b + c) <= -1 Then
    CalculateDistance = 0
Else
    CalculateDistance = Atn((Sqr(1 - (a + b + c)^2) / (a + b + c))) * RadiusEarth
End If

__esri_field_calculator_splitter__
CalculateDistance
STEPHEN J. MILLER

Education:

Virginia Tech - Blacksburg, VA
M.S. Mining and Minerals Engineering (December 2005)
B.S. Mining and Minerals Engineering (May 2003)
B.S. Geophysics (May 2003)

Previous Employment:

Graduate Research Assistant, Blacksburg, VA, August 2003 – May 2005
• Conduct Research involving GPS and GIS projects.

Teachers Assistant, Blacksburg, VA, January 2002 - May 2003
• Aided in the design and testing of a multi-channel borehole geophone array for the Department of Energy.
• Aided in developing software and equipment for Computer Aided Tomography of objects.

Engineering Assistant, Marshall Miller & Associates, Bluefield, VA, May 2001 – August 2001
• Assist senior engineers and geologists with:
  Design of surface and underground mines, surface water runoff calculations, permitting process for the expansion of a surface quarry near a residential area, calculations for numerous mine reclamations.

• Assisted in the underground surveying of a coal seam to assist in the computer visualization to optimize the longwall mining operation.

Activities:

President of Burkhart Mining Society, Virginia Tech Student Chapter of SME 2002-2003
Member of the Geology Club at Virginia Tech 2002-2003
Treasurer of Burkhart Mining Society 2001-2002
Student Member of SME 1999 – Present
Member of Burkhart Mining Society 2000-2005
Member of Mining Competition Team 2001 and 2002