Wireless Internet Based GPS Tracking System
(Intended for use by the University of Utah Shuttle System)

Richard Wells
Amany El Gouhary
December 17, 2006
# Table of Contents

Abstract ............................................................................................................................... 3  
Motivation ........................................................................................................................... 3  
Background Information .................................................................................................. 4  
Functionality ....................................................................................................................... 5  
    Use Cases....................................................................................................................... 5  
        Program Installation................................................................................................. 6  
        Optional Programs – .............................................................................................. 7  
Hardware Design ................................................................................................................ 7  
    Computing Platform ..................................................................................................... 7  
    GPS Receiver ............................................................................................................... 8  
    Wireless Internet ......................................................................................................... 9  
    Power Circuit .............................................................................................................. 10  
    Enclosure .................................................................................................................... 11  
Software Components ....................................................................................................... 12  
    Website ....................................................................................................................... 12  
    Google Maps ............................................................................................................... 12  
    SQL Importer ............................................................................................................. 12  
    Database Poller .......................................................................................................... 12  
    Graphical GPS Viewer .............................................................................................. 12  
    Random Point Generator ......................................................................................... 13  
    Route Simulator ......................................................................................................... 13  
    Serial Emulation ........................................................................................................ 13  
    WPA Supplicant .......................................................................................................... 14  
    GPSd .......................................................................................................................... 14  
    GPSTest ..................................................................................................................... 14  
Problems encountered ....................................................................................................... 15  
    Embedded Active Antenna Short............................................................................. 15  
    JavaScript, SQL, ASP.Net 2.0 Learning Curves ......................................................... 15  
Equipment Configuration and Capabilities ....................................................................... 15  
    “Ubiquitous” Campus Wireless Network ................................................................... 16  
Technical Specifications ................................................................................................... 16  
    Bill of Materials ......................................................................................................... 17  
Project Future .................................................................................................................... 18  
Conclusion ......................................................................................................................... 18  
References ......................................................................................................................... 19  
Acknowledgements ........................................................................................................... 19
Abstract

The purpose of this project is to design and construct a hand-held wireless GPS tracking device that can be tracked from the Internet. The project consists of three parts. The first part is a mobile device with an embedded GPS and wireless Internet connection to transmit its current location. The second part is a web server that will receive the data, parse it, and store it for access over the Internet. The third component is the user interface that will allow others to visually see where the hand-held GPS device is and has been. To view its location, one could use any device that can connect to the Internet such as a desktop computer, laptop, PDA, or cell phone. The data available through a browser includes a scalable map of the surrounding area, latitude, longitude, speed, and altitude of the hand-held device. The system is intended to be a general purpose tracking device; however, the user interface will be tailored to the university shuttle system.

Motivation

The intended application for our wireless GPS tracking device is the University of Utah shuttle system. As our group was formulating ideas for our project, we came to the conclusion that both of us were frustrated with the university shuttle system. We had several complaints in common: the shuttles didn’t come often enough; they were often late leaving us out in rain, snow, and heat; and worst of all, sometimes they never showed up.

In an informal study of the punctuality of the university shuttle system by group members, it was found that on average the shuttle was three and one-half minutes late. The distribution of the shuttle departure times is presented in the following table.

<table>
<thead>
<tr>
<th>Time Frame of shuttle departure (in minutes)</th>
<th>Percent of departures</th>
</tr>
</thead>
<tbody>
<tr>
<td>Early by 1-5</td>
<td>15%</td>
</tr>
<tr>
<td>On time - 4 late</td>
<td>38%</td>
</tr>
<tr>
<td>Late by 5 – 10</td>
<td>16%</td>
</tr>
<tr>
<td>Late by &gt; 10</td>
<td>23%</td>
</tr>
<tr>
<td>Never came</td>
<td>8%</td>
</tr>
</tbody>
</table>

*Table 1: University shuttle punctuality*

Each of us has many “horror” stories from our shuttle riding experience. We have seen a shuttle arrive 50 minutes late when it was scheduled to come every 10 minutes. It was snowing, but every shuttle that came would provide us with the update that it would “be there shortly.” We have experienced drivers running ten minutes late and going into the Union building or hospital for five minutes to get a drink. We have encountered shuttles that don’t come at all. Shuttle drivers are aware that they miss stops, but the problem continues. Being students with tight time schedules, the unreliability of the
shuttle system can greatly affect us. We have been late to class and almost had our lack of punctuality affect our class grade.

We and many other university students have thought, “I wish I knew when the shuttle was coming.” Our device is engineered to address that question. It will allow anyone with an Internet connection to track the shuttle and know if it is early, on time, or late. With this information students can adapt their schedule to meet the projected shuttle arrival times.

**Background Information**

The basic idea of any satellite positioning system is to calculate the distance between a satellite and the user’s current location. The position of each satellite is known. Using the calculated distance from four satellites, one can narrow their current position to exactly one place on earth’s surface. The accuracy of the positioning depends on how accurately the distance is measured and how precisely the position of the satellite is known.

In 1973, the Department of Defense funded the Navigation Technology Program that resulted in Navigation System and Ranging Global Positioning System (NAVSTAR GPS), now known as GPS. The current GPS system consists of satellites in 6 different orbits. At present, there are 29 active satellites circling the globe at an altitude of 20,200 km. They are arranged to provide at least four satellites within the line of sight of any point on the globe.

GPS satellites broadcast three kinds of data. First is the almanac data. It sends the course time information along with status information about the satellites. The second is the ephemeris data which contains highly accurate orbital information about the satellites. Each GPS satellite is continually updated from continuous measurements made from Earth. The time information consists of the course acquisition code (C/A), a pseudo random code which repeats every millisecond.

The GPS receiver calculates its position from the timing information. It compares the C/A code against its internal crystal oscillator clock and the C/A code the receiver generates. This timing mechanism is highly accurate. An error of a microsecond yields an error of 30 meters. Position can be determined to about 1% of a bit time or 3 meters under optimal conditions. Based on the timing calculations and the ephemeris data, the GPS receiver can calculate its current position.

During late 2005, the first of the next-generation of GPS satellites was launched. One of the main civilian benefits is a second signal named L2C that will yield increased accuracy and precision. Additional civilian signals are in the works. Additional measures such as the Wide Area Augmentation System (WAAS) and Differential GPS (DGPS) are available to increase the accuracy of the GPS readings. Improvements to increase the accuracy of GPS position readings continue.
Functionality

Use Cases

The two main use cases for this project tracking the University Shuttle system include an end-user, such as a student, accessing a web page on an Internet capable device. The user sees a map and a column on both sides. Clicking on the route name in the column on the left displays the route information and position. The column on the right has a drop down box which allows the user to choose which route they want the map to center on and follow. Below the tracking selection box are checkboxes that allow the user to select the route or routes of which they want to see the recent path displayed.

There are three version of this screen available: widescreen, standard, and PDA. They are optimized to fit various screen sizes.

From the website the student can create a user account and log in. When logged in, the three screen sizes are still available. User-specific functions were an addition to the project and only partially implemented with plans to complete the implementation shortly. From here the user can select times that they would like to be alerted of the shuttle’s progress. Another feature that will be implemented in the future is keeping track of the state of the map display, including the route to center on, routes to show current information about, and tracks to show. This would be restored to the last setting whenever a user logs in.

Specific users can be designated as administrators. Administrators have the option to view and modify all users. There are several additional features we have planned to implement here including administering the alerts settings, sending email to users, uploading routes, etc.

The second use case involves a user deciding that they want to use their favorite GPS program to follow the tracking unit. They would start up the serial emulator, start up their favorite GPS program, connect their favorite GPS program to the com port the
serial emulator is feeding data to, and use their favorite GPS program as normal from anywhere in the world.

**Program Installation**

This project consists of a chain of about 5-10 applications, depending on configuration. Prerequisites for the installation process include a working instance of SQL Server 2005 and Microsoft’s .NET Framework 2.0. Currently the project comes in a ZIP file. Unzip the files into an easily accessible folder. More detailed information about the functionality of these programs is found later on in the software components section.

1) **GPSData Database** – Using the SQL scripts, create a database named GPSData with two tables titled GPSPoints and UnitOnRoute. Also create the RouteStatus view using the SQL script provided. The GPSPoints table contains all the latitude, longitude, speed, and altitude measurements received from the tracking units. The UnitOnRoute table contains information about the routes that the tracking units are simulating. The information includes route name, route number, driver, etc.

2) **ASP Membership Database** - To configure user accounts, the ASP membership database needs to be created. It can be done by going to the SQL Server command line and typing the command aspnet_regsql.exe -E -S localhost -A mr to create the database. Administrators can be configured by using Visual Studio’s Web Administration Tool (SWAT).

3) **Website** - Copy the website code to the virtual directory desired. Configure it through Internet Information Services to run as an application.

4) **Google Maps API Key** - Register for a Google Maps API Key at http://www.google.com/apis/maps. Insert that key into the loggedOut.master and loggedIn.master files.

5) **RouteCreator** - Use the RouteCreator to create routes that you want to emulate. This involves running the program, entering in the information about the route that you are going to simulate, and driving along the route desired. The MAC for the route and the tracking unit has to be identical.

6) **Database Poller** – The program extracts the information from the SQL database and outputs it into the format that Google Maps requires. Configure the outputXML.exe.config file with the parameters desired. The parameters include: time between polling of the database, file path to write the XML files too, and the connection string to the database.

7) **SQLImport** – This program needs to have input the IP address and port that you desire to listen for tracking units communicating on. It also requires the database connection string. Once these parameters are set, run the program and it will receive incoming connections from a tracking unit.
Optional Programs –

1) **GPSGraphical** – The program is a utility to diagnose the strength of the GPS fix, wireless Internet connection, basic status of the Linux operating system on the tracking unit, and to compare for accuracy against a commercial GPS receiver. Input in the port that the serial GPS is on, if desired. Run the program. Input the IP address of the tracking unit into the Gumstix column and then click the radio button next to “From Gumstix.” Readings will come over the Internet.

2) **Serial Emulation** – This process allows you to access the tracking unit over the Internet with any program that will connect to a serial GPS. The first step is to install com0com, an open source null terminal emulator. Two modifications have to be made using regedit. The ports have to be renamed to comX depending on the current com ports of the computer. The baud rate emulation parameter also has to be set. At this point the data should be available over the output com port that was just renamed.

**Hardware Design**

On conception of this project, we realized that we would need more than a traditional microcontroller. We knew we needed to run several programs simultaneously on the tracking unit. We also knew that we could not implement some of the tasks by ourselves, such as WPA authentication. We chose to purchase a board that would support a minimal operating system and that would have a suite of programs available.

**Computing Platform**

In our search for a platform that would support these characteristics, we discovered a company called Gumstix. They manufactured a board about the size of a stick of gum. It has on it an XScale PXA255 chip which runs at 400MHz, 64Mb of RAM, and 16Mb of flash. It is capable of running a stripped down version of Linux. The basic footprint was under 4Mb. It was able to fit in this amount of space because it uses a busybox implementation of many of the common Linux programs. We knew that would leave us plenty of room to install additional programs and to hold the programs we would write. The platform turned out to be highly stable and with a little research everything we desired was able to be accomplished with this platform.
**GPS Receiver**

During our project we realized it would be difficult to obtain our planned GPS engine because the company tailored to OEMs. Also the documentation left much to be desired. While we were searching for a suitable GPS receiver, Gumstix developed a new expansion board with a GPS engine. The board interfaces with the processor through a TTL serial connection. The GPS expansion board is only slightly bigger than the Gumstix board with the processor, flash, and RAM.

The expansion board had a remarkable GPS engine. It is the U-blox LEA-4H. The GPS Engine itself is programmable and has an additional feature called SuperSense which integrates the GPS signal over time to allow it to track GPS signals down to -158dB. It measures just 17 x 22 mm. Its features include antenna short circuit detection, antenna open circuit detection, 16 channel receiver, low noise amplifier, USB output, SPI, serial output, and its own ARM7TDMI processor. It supports an active or a passive antenna. It has a software customization kit. It can also support a backup battery to maintain the ephemeris and almanac data.

After we had received the GPS expansion board with the U-blox LEA-4H GPS engine module on it, we endeavored to find an antenna. We found that the GPS engine board was only configured to support a passive antenna. Another characteristic our antenna needed was to be small and easily embeddable. We could not find a passive antenna that met those characteristics. We did find a couple of active embedded antennas. Using the engine module’s system integration manual and the accounts of a few people who have previously modified their U-blox modules to support active antennas, we modified our board to support an active antenna. This included connecting the V_ANT pin to Vcc through a 10Ω resistor to prevent excessive current. The engine module filtered the power and then provided the antenna with the voltage and current necessary to power the antenna’s LNA. With this configuration we saw more sensitivity.
and better tracking of satellites than with any other GPS system we had previously used.

One problem that we ran into is that intermittently we would power the unit on and we would not be able to track satellites. The problem was traced to the active antenna modification. The problem was the pin that connected Vcc was extremely close to a ground post of the SMA antenna connector. Intermittently it would touch the antenna connector. It would draw excessive current and the engine module’s short circuit detection would shut it down. We inserted a small layer of insulation between the ground post and the wire. It solved the problem.

**Wireless Internet**

To obtain a wireless Internet connection, our original plan consisted of using a Compact Flash based wireless Internet card. However, it had driver incompatibilities with the newer version of Linux we were running. Connection with the university’s wireless Internet was done with an expansion board from Gumstix. The expansion board has the Marvell® 88W8385 module and a couple of chips to interface with the Gumstix using the PCMCIA protocol. The Marvell module implements the 802.11g standard for a bandwidth of 54 Mbps. The open source driver supported 802.1x, WPA, WPA2, and WEP so we could connect to practically any access point for testing. The wireless Internet module had adjustable power settings allowing us to tune the parameter to save battery life or have maximum range. We found the connection to be superior to older laptop wireless Internet connections and comparable with current wireless Internet connections.

Since the proposal of the project, the university decided to phase out the old secure.utah.edu network and replace it with a new secure wireless network called uconnect.utah.edu. The old secure.utah.edu network used dynamic WEP for encryption and 802.1x for authentication. The uconnect.utah.edu network uses 802.1x for authentication and WPA or 802.11i for encryption. The new encryption standards are a lot more processor intensive and not as well supported or suited for embedded systems. We had initially planned to utilize Xsupplicant, an open source program that was developed mainly by a staff member here at the University of Utah. However, because that had never been tested on an ARM based processor, we decided to go with a better known and deployed supplicant called WPA Supplicant. It had also been tested and was compatible with the open source driver that interfaces with the Marvell wireless chipset.

The university supports preconfigured clients for Windows and Macintosh. They do not support Linux or publish the technical specifications of the network to allow someone to connect easily using Linux. We did a lot of research and pulled pieces of
information from many sources to figure out what standards were used in the network. Authentication to uconnect.utah.edu uses EAP/TTLS with the second phase using PAP. It also took a lot of work to find a certificate that was in the proper format for Linux to use. Once we had the correct information, connecting with WPA Supplicant was not trivial. We had to get the right format and order of the configuration file. We also had to trace through the debug logs to find several errors. The most surprising of which was we had to set the time on the tracking unit before it would authenticate the certificate. The certificate had a beginning date and on initial power-up the tracking unit has a date from 1970 because there is no battery backup on the system clock.

**Power Circuit**

Constructing the circuit to power the tracking module proved to be more difficult than we anticipated. Initially, we had designed a switching power supply. However, due to the number of external components required and their relative size, we decided to switch to a power circuit based on linear voltage regulators. Another benefit of that choice is that the GPS module is extremely sensitive to high frequency noise in the power supply and does not perform well with a switching power supply.

The circuit was designed with two parallel input channels for an external power source and a battery. The external power source was designed to function with a car adapter we constructed out of an old cell phone charger. It also functioned off of AC power with the help of an adapter that would rectify the power and lower its voltage to 7.5V. Each channel would pass through a linear voltage regulator that had its output fixed at 5V. A couple of capacitors were present at the input and output of the voltage regulators to smooth the output power.

The voltage of the output of the battery powered a Maxim 703 microprocessor power supervisory chip.
The chip had a voltage comparator circuit. It was utilized to detect a low battery condition. From the battery input, a voltage divider was constructed to a level that indicated a low battery condition when the center of the voltage divider was at 1.25V. High resistance resistors were used to minimize quiescent power. When the voltage at the center of the voltage divider went below 1.25V, an output pin was set low. The output of the voltage comparator circuit went through a chip which implemented a not function and powered an LED when the battery voltage was low.

The output of both voltage regulators were fed into a power switching circuit. Using the output of the external power adapter channel, another microprocessor supervisory chip was powered. When it sensed that the voltage was too low or non-existent, it opened the gate of a 1A P channel power MOSFET for power from the battery to flow through. Otherwise, the chip closed the gate on the transistor and power from the external power adapter powered the circuit. The output had a power adapter that fit into the power adapter plug on the Gumstix making it removable. Our unit also has an on and off switch which is implemented with a double throw single pole switch to connect or break both input channels simultaneously.

In initial assembly of the power supply circuit, we found that the low battery detection was functioning while the circuit was running on an external power supply. We found that we were trying to get too much functionality out of our power management chip. This is what led us to use two Maxim 703 chips because we only wanted the one that controls the low battery detection to be powered when we were powering the tracking unit with the battery.

Choosing the proper battery proved to be more difficult than expected. We had chosen to use a NiMH 9V battery in the initial design. Upon the completion of the construction of the tracking module, we tested its power consumption. We found it consumed .41A while actively processing and .3A normally. Since the battery was rated at 150 mAh we expected it to last 20-30 minutes. We found that it lasted about 6-10 minutes. In researching NiMH batteries, we found that under high current drain they do not last as long as under low current drain. We went searching for another battery. We found a camcorder battery that provided 8V. That lasted for over thirty minutes. However, in testing the output voltage level from the voltage regulator started varying by about .5V after fifteen minutes. We tracked down the cause of that to the voltage regulators having a high dropout voltage. It was about 2.5V, 0.5V higher than the datasheet claimed. We replaced the linear voltage regulators with low dropout voltage regulators (NTE 1951) and it fixed the problem.

**Enclosure**

Our enclosure consisted of a 4” x 6” x 2” ABS plastic box. With our battery, power circuit on separation posts, and Gumstix it proved to be about the right size. We made openings in the case for the switch and external power adapter. However, with the knowledge of a prior implementation and fabrication of a circuit board, the tracking unit could be made significantly smaller.
Software Components

Website

The website will be the software component that the end user interacts with. It is written in ASP.Net 2.0 using C# as the code behind language. It makes significant use of JavaScript which Google Maps is based off of. Its functionality was described in the use cases section above.

Google Maps

Google provides a well documented API for Google Maps. In order to use it, a user has to obtain a Google Maps API Key. Google Maps can be interfaced with using JavaScript. Some of the functionality of Google Maps that was implemented on the website includes: icons, polylines, map panning, satellite view, and AJAX to refresh the shuttle history and current position.

SQL Importer

The SQL Importer is a multithreaded application written in C#. It opens up a listening socket on a port (currently port 6000) and waits for the GPS tracking units to connect to it. When it receives data from a unit in the form of an XML file, it parses the file into individual readings. It is written so that units may send an arbitrary number of readings at a time. It then takes each individual measurement and extracts the latitude, longitude, speed, altitude, and MAC address from the XML file. It then puts a row into the SQL database for each position measurement from the unit.

Database Poller

Database poller’s overall purpose is to extract information from the database for each route and to output a XML file containing past and present points for Google Maps to use to display current position and a track history. This application is written in C#. From the UnitsOnRoute table in the database, it reads the routes that are available. For each route, it extracts a finite time amount of the route’s track history from the GPSPoints table in the database. It takes the information and formats it into an XML file and stores it on the disk on the Web Server. The XML file is then used by the Google Maps API in an AJAX manner to dynamically update the current shuttle position and track history without refreshing the page.

Graphical GPS Viewer

During the early stages of system integration testing, we saw a few points of weakness. First, we did not have a way to tell the strength of the wireless Internet signal.
between an access point and our unit. Second, Gpsd did not have a way to display the signal strength, azimuth, and elevation of each GPS satellite that the GPS engine was currently tracking. We did not know which direction was best for the antenna to point, if enough satellites were available for a fix, and how strong our GPS signals were. Additionally, we saw some inaccuracies and jumps once we plotted the data we received from the unit. We needed a standard to compare it too. Another commercial GPS unit was available and was used to compare our readings against. This program was designed to solve those problems.

GPS Graphical was designed to connect over the Internet to the unit and retrieve its current GPS readings. It would display the latitude, longitude, speed, altitude, time, PDOP, VDOP, and HDOP from the unit. Over a serial connection another GPS receiver can be connected and its readings displayed for the various measurements. If both have a GPS fix, the distance between the two measurements is calculated.

Some variations on this are integrated into the program. The serial GPS can record its measurements to the database, thus simulating a tracking unit. The Gumstix can use the latest values direct from the database to compare against the serial GPS.

To deal with signal strength problems we parsed and displayed the satellite number, elevation, azimuth, and signal to noise ratio (SNR) of each satellite that the GPS unit was receiving. This allowed us to compare relative signal strengths and evaluate if improvements helped us receive the GPS satellite signals better.

Over the Internet, the program also queries the tracking unit for various system status measurements. It reports the connection status, link quality, load average, free memory, bytes sent over the network, and bytes received from the network.

**Random Point Generator**

Quickly it was discovered that testing of the user interface required having an active GPS tracking unit. With varying weather conditions, ability to move equipment outside, and hardware assembly still in progress it was deemed necessary to write a program to simulate a tracking unit. The random point generator is written in C#. It is given a MAC address and a starting point and using random numbers, it will move a small amount in a random direction each second. Each movement is independent of the last. It would insert the data points into the SQL database as a traditional tracking unit would. This allowed testing on the front-end Internet interface.

**Route Simulator**

To be able to demonstrate the capabilities of the system on demo day and hopefully to the University Shuttle System, a way was needed to demonstrate the capacity of this system to handle more than one shuttle. Route Simulator was written to fill the need. It is written in C#. This program has a table representing each shuttle route. It also polls a controller table to decide what routes there are to simulate. It simulates the actual traffic that the shuttle routes will generate.

**Serial Emulation**

Many useful and very well written programs have been coded for GPS units. However GPS units are assumed to be within a serial or USB connection of the computer. Several statistics and graphical representations of data that were generated by other
programs were desired. The general functionality of being able to use any program written for a GPS unit was seen to be beneficial.

The implementation that was used to overcome this problem involved two software components. The first component was an open source null terminal emulation program, com0com. It allowed serial data to be input into a virtual COM port and then output over another virtual COM port.

The second software component involved in this solution was a program that was written in C#. It configures GPSd to make available the raw NMEA data. The raw NMEA data is then transmitted over the Internet. The program forwards it into one end of the null terminal emulation and on the other end of the virtual null cable the user is free to attach their favorite GPS program. A benefit of this approach is that as long as the tracking unit is within wireless Internet range, the arbitrary GPS program can be used from anywhere in the world.

**WPA Supplicant**

WPA Supplicant is an open source supplicant for WPA and 802.1x systems. It supports EAP/TTLS and PAP authentication during phase two. These are the standards that the uconnect.utah.edu network implements. With some patches provided by Gumstix to make the WPA supplicant code compatible with ARM systems, we compiled the code and installed it on the tracking unit. We obtained the certificate needed from the university. With some trial and error in the configuration file and setting the current time, we were able to connect to the uconnect.utah.edu network.

**GPSd**

GPSd is a daemon program that attaches to a serial port and parses the NMEA data coming over the serial port and it makes it available externally through a socket. When a client program connects to it over a socket, it accepts one letter commands and will return the latest information it has from the GPS receiver that corresponds to the command. For example, sending it an “a” over the network port will return the current altitude.

**GPSTest**

This is the program that we implemented on the tracking device. It is written in C. GPSTest is a client program for GPSd. At a fixed interval, it will poll GPSd through a socket for the information that we are interested in. Currently that information includes latitude, longitude, speed, and altitude. It appends the MAC address to the information so SQLImport can uniquely identify tracking units. GPSTest stores each reading in an
XML format for transfer over the network. It stores the last ten seconds of readings in a file.

**Problems encountered**

*Embedded Active Antenna Short*

The weekend before demo day, we hooked up our unit as normal and our software did not show that the GPS engine was tracking any satellites. This caused us great concern. We initially thought that the problem was related to the active antenna modification. We verified that the wire that was providing power for the active antenna was not shorted. To confirm what we thought was the diagnosis we found a way to access the NMEA data sentences that were coming from the GPS engine module. It seemed to support our diagnosis. It would report that the unit’s antenna status was okay to start with, but it would short as the antenna was connected. We thought that we saw possible beads of solder bridging pins. We cleaned the area around the solder connection with no success. We removed and soldered the active antenna modification. That did not help. Al then measured the resistance of the embedded active antenna and found out that it was shorted. When the problem was discovered, we did not have time to order anything else over the Internet and the parts needed were not available in Utah. The best thing that we could find was a large wide band antenna. We tried it and the antenna did not short the GPS engine out. That confirmed our new diagnosis. We went with our back up plan. To obtain the NMEA data we used a previously acquired commercial GPS unit. We connected to our tracking module an expansion board that would do the voltage level conversion from RS232 to TTL. We connected the serial output of the GPS to this board and pointed the GPSd program to the serial port we had just connected and started getting the NMEA data.

*JavaScript, SQL, ASP.Net 2.0 Learning Curves*

As we were preparing the project proposal, we discovered that some of the software tools and technologies best suited to the project, we had not used previously. However, we were willing and excited to learn them. They included JavaScript, SQL, and ASP.Net 2.0. JavaScript was the language that was necessary to interface with Google Maps. A SQL database was needed to store all the data the tracking module would create. A database was also helpful to be able to customize the data we extracted from it. ASP.Net 2.0 was needed over generic HTML to provide extra functionality, security, and membership accounts. With each of these technologies there was a learning curve, but with proper reference manuals we were able to figure out exactly how to use these technologies the way we desired in our project.

*Equipment Configuration and Capabilities*

For our senior project we were given a Windows and a Linux computer to use. We used the Linux computer for the custom buildroot toolchain to cross compile the software and kernel for the tracking module. It was also used to maintain our weekly logs. We used the Windows machine to serve as our web server. However, because of security restrictions, these machines could not be accessed from outside the School of
Computing network without a vpn connection. This made testing from home impossible. We tried to keep a copy of our latest software on the machines, but they were too slow to make it practical. We decided to use our own laptops for portability and speed. Another thing that was noticed is that others would change the configuration of the computer, breaking our software. We were able to track down some of the changes others made, while some we were not.

“Ubiquitous” Campus Wireless Network

When we formulated the idea for our project, we researched the campus wireless network. They stated that they would have the wireless network to the state of “ubiquitous” around campus by the time our project was scheduled to be complete. Along with everything else in the wireless network, that changed. Now the scheduled completion date for the campus “ubiquitous” wireless network is 2008. That makes our project forward thinking. While it would work in some areas and we have programming measures to mask a lost connection, it would not provide the smooth real-time updates that the system is designed to provide.

Technical Specifications

**Embedded System**

<table>
<thead>
<tr>
<th>Processor</th>
<th>PXA 255 (400 MHz XScale)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM</td>
<td>64Mb</td>
</tr>
<tr>
<td>Flash</td>
<td>16Mb</td>
</tr>
<tr>
<td>Connectors</td>
<td>1 HiRose, 1 92pin</td>
</tr>
<tr>
<td>Max Current</td>
<td>.4A</td>
</tr>
</tbody>
</table>

**Embedded Software**

<table>
<thead>
<tr>
<th>Linux Kernel</th>
<th>2.6.17</th>
</tr>
</thead>
<tbody>
<tr>
<td>Other used software packages</td>
<td>WPA_supplicant, Ntp, Gpsd</td>
</tr>
<tr>
<td>Frequency embedded software queries position</td>
<td>2 sec</td>
</tr>
<tr>
<td>Frequency between position transmit</td>
<td>10 sec</td>
</tr>
<tr>
<td>Language of embedded programs</td>
<td>C</td>
</tr>
</tbody>
</table>

**Power System**

<table>
<thead>
<tr>
<th>Max Current</th>
<th>1A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Typical Load</td>
<td>.3A - .4A</td>
</tr>
<tr>
<td>Input Voltage</td>
<td>6V - 20V</td>
</tr>
<tr>
<td>Output voltage</td>
<td>4.2V</td>
</tr>
<tr>
<td>Battery Model</td>
<td>Canon BP-508</td>
</tr>
<tr>
<td>Battery Chemistry</td>
<td>Li-Ion</td>
</tr>
<tr>
<td>Battery Voltage</td>
<td>Rated at 7.4V</td>
</tr>
<tr>
<td>Battery Capacity</td>
<td>800 mAh</td>
</tr>
<tr>
<td>Low Battery Detection Voltage</td>
<td>Detected at 6.8V</td>
</tr>
<tr>
<td>Available Power Sources</td>
<td>Battery, Jack for power from an AC or car adapter</td>
</tr>
</tbody>
</table>
### Wireless Internet

<table>
<thead>
<tr>
<th>Feature</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chipset</td>
<td>Marvell® 88W8385</td>
</tr>
<tr>
<td>Security Mechanisms available</td>
<td>None, WEP, WPA, WPA2, 802.1x</td>
</tr>
<tr>
<td>Transmit Power</td>
<td>100mW, variable</td>
</tr>
<tr>
<td>Wireless Standard</td>
<td>802.11g</td>
</tr>
<tr>
<td>Antenna Connector</td>
<td>SMA female</td>
</tr>
</tbody>
</table>

### Gpsstix

<table>
<thead>
<tr>
<th>Feature</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Engine</td>
<td>U-blox LEA-4H</td>
</tr>
<tr>
<td>Channels</td>
<td>16 Parallel Channels</td>
</tr>
<tr>
<td>Output Format</td>
<td>NMEA 2.3</td>
</tr>
<tr>
<td>Output Baud Rate</td>
<td>9600 baud</td>
</tr>
<tr>
<td>Tracking Sensitivity</td>
<td>-158dBm</td>
</tr>
<tr>
<td>Antenna Connector</td>
<td>SMA female connector</td>
</tr>
<tr>
<td>Frequency Band</td>
<td>L1 (1575.42 MHz), C/A code</td>
</tr>
<tr>
<td>Cold Start Time</td>
<td>36 s</td>
</tr>
<tr>
<td>Hot Start Time</td>
<td>&lt;3.5 s</td>
</tr>
</tbody>
</table>

### GPS Active Antenna

<table>
<thead>
<tr>
<th>Feature</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td>V.Torch G050A</td>
</tr>
<tr>
<td>Gain</td>
<td>26dB</td>
</tr>
<tr>
<td>Voltage</td>
<td>3.3V +/- 0.5V</td>
</tr>
<tr>
<td>Current</td>
<td>12 mA</td>
</tr>
<tr>
<td>Weight</td>
<td>18 g</td>
</tr>
</tbody>
</table>

### Bill of Materials

<table>
<thead>
<tr>
<th>Item</th>
<th>Cost</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gumstix</td>
<td>$129</td>
<td>Gumstix</td>
</tr>
<tr>
<td>Gpsstix</td>
<td>$130</td>
<td>Gumstix</td>
</tr>
<tr>
<td>Active Antenna – V.torch G050A</td>
<td>$11.95</td>
<td>Spark Fun Electronics</td>
</tr>
<tr>
<td>Right angle SMA adapter</td>
<td>$5.50</td>
<td>Ra-Elco</td>
</tr>
<tr>
<td>Wifistix and antenna</td>
<td>$79.00</td>
<td>Gumstix</td>
</tr>
<tr>
<td>Screws and Spacers</td>
<td>$4.00</td>
<td>Gumstix</td>
</tr>
<tr>
<td>2 NTE 1951 voltage regulators</td>
<td>$8.60</td>
<td>Ra-Elco</td>
</tr>
<tr>
<td>2 MAX703</td>
<td>$10.04</td>
<td>Maxim</td>
</tr>
<tr>
<td>1 NOT chip</td>
<td>$.50</td>
<td>BYU Electronics</td>
</tr>
<tr>
<td>1 LED</td>
<td>$0.50</td>
<td>Ra-Elco</td>
</tr>
<tr>
<td>1 SPDT Switch</td>
<td>$3.00</td>
<td>Allied.com</td>
</tr>
<tr>
<td>Wire</td>
<td>Essentially Free</td>
<td>Already owned</td>
</tr>
<tr>
<td>2” x 3” Component PC Board</td>
<td>$1.79</td>
<td>Radio Shack</td>
</tr>
<tr>
<td>Wire wrap wires</td>
<td>Essentially Free</td>
<td>Junior Hardware Lab</td>
</tr>
</tbody>
</table>
(2) Wire wrap 8 pin dip sockets $1.00 Ra-Elco

Wire wrap 16 pin dip socket $1.00 Ra-Elco

1 1N4001 diode $0.10 Ra-Elco

1A power mosfet FDS8433A $0.68 Digi-key www.digikey.com

BP-508 battery $19.99 NexTag www.nextag.com

Enclosure $4.99 Radio Shack

M size coaxial power adapter $2.79 Radio Shack

---

### Project Future

This project has taught us a lot about embedded hardware, circuits, software, and networking. The technologies we are using are a lot of fun to work with. We plan to continue working on the project on our own time. During the course of the project, many students and faculty have remarked that this is something that the University Shuttle system needs badly. There are also many technologies such as AJAX, SQL Server Notification Services, SQL Server Reporting Services, WAAS, SMS, and WAP that would enhance our project and we have a desire to learn them. The tracking unit also needs a small LCD to report basic information.

The next phase of the project is enhancing the website. Here we plan to finish off the membership functions by providing each user with a profile of preferred display options. Also users will be able to set alerts. This will notify the user of the current state of the shuttle at times specified. Administrative functions of the website will be enhanced.

The second phase would be to add the LCD and fabricate a circuit board for the power supply. Once the hardware is finalized, a better battery connection socket can be found and the unit can be made much smaller.

The third phase would be experimenting with the new technologies previously mentioned to enhance the usability and functionality of the tracking system.

Upon completion of the project, we plan to present it to the University Shuttle system to seek funding and permission to implement a tracking system on every shuttle. If possible, Internet capable devices such as ultra mobile PCs would be deployed at major shuttle stops so students without a mobile Internet accessible device could be updated on the status of the shuttles.

### Conclusion

We believe that this project has been a success. We have been able to accomplish our baseline goals and implement some of our extras. Even though along the way we ran into many problems, our project was flexible enough to adapt to the problems we encountered. We were able to build a successful tracking unit and implement the software to track it. In the process we learned a lot about hardware and software that will enable us to be better computer engineers.
References


Acknowledgements

We would like to thank Al Davis for his continued support, technical help, debugging, and sessions where he taught us topics that aren’t covered in books in school, but require hands on training.
#include <stdio.h>
#include <netdb.h>
#include <netinet/in.h>
#include <errno.h>
#include <time.h>
#include <fcntl.h>
#include <string.h>
#include <pthread.h>

typedef struct sockaddr SA;

//Pound define constants
#define MAXLINE 8192
#define RIO_BUFSIZE 8192

typedef struct {
    int rio_fd;
    int rio_cnt;
    char *rio_bufptr;
    char rio_buf[RIO_BUFSIZE];
} rio_t;

//Function Prototypes
int open_clientfd(char *hostname, int portno);
void rio_readinitb(rio_t *rp, int fd);
void Rio_readinitb(rio_t *rp, int fd);
char *Fgets(char *prt, int n, FILE *stream);
void app_error(char *msg);
ssize_t rio_writen(int fd, void *usrbuf, size_t n);
void Rio_writen(int fd, void *usrbuf, size_t n);
void unix_error(char *msg);
ssize_t rio_readlineb(rio_t *rp, void *usrbuf, size_t maxlen);
ssize_t Rio_readlineb(rio_t *rp, void *usrbuf, size_t maxlen);
ssize_t rio_readn(int fd, void *usrbuf, size_t n);
void Fputs(const char *ptr, FILE *stream);
void *sendFile(void *vargp);
//int sendFile(void *vargp);
void *timeOut(void *vargp);

char buf1[MAXLINE];
int main(int argc, char **argv)
{

// Variable Declarations
int clientfd, port, n, f2;
char *host, buf[MAXLINE];
rio_t rio;
FILE *f1;
time_t t1,t2,*temp;
pthread_t tid,tid1;
if(argc != 5) {
    fprintf(stderr, "usage: %s <host> <port> <host> <port>\n", argv[0]);
    exit(0);
}
host = argv[1];
port = atoi(argv[2]);
n = 0;
// connect to gpd server
clientfd = open_clientfd(host, port);
while (n<1000) {
    Rio_readinitb(&rio, clientfd);

    f1 = fopen ("gpsStatus.xml", "wt");
    /* gpsStatus.xml is the name of the file */
    /* "wt" means the file is a write only text file */
    fprintf(f1,"<ROOT>\n");
    t1 = time (temp);
    t2 = t1;
    char* mac = "00:0B:6B:0B:A6:AC";
    while(difftime(t2,t1) < 10) {
        char buf2[] = "o"; // full report /time/position/velocity
        Rio_writen(clientfd, buf2, strlen(buf2));
        Rio_readlineb(&rio, buf, MAXLINE);
        if (buf[7]!='?')        // if data is available
            {
                fprintf(f1," <Point ");
                fprintf(f1," Latitude = "buf[31];
                fprintf(f1," Longitude= "buf[41];
                fprintf(f1," Speed = "buf[85];
                fprintf(f1," Altitude = "buf[53];
                fprintf(f1," MAC = "mac);
                fprintf(f1," />
\n");
            
        
    
    
}
sleep(2);
t2 = time(temp);
}
fprintf(f1,"</ROOT>n");
fclose(f1);
// copy gpsStatus to buffer to send
f2 = open("gpsStatus.xml", O_RDONLY,0);
Rio_readinitb(&rio, f2);
rio_readfileb(&rio, buf1, MAXLINE);
close(f2);
//if (sendFile(argv)<0)
//      Fputs("error sending file",stdout);
pthread_create(&tid, NULL, sendFile, argv);
pthread_create(&tid1,NULL, timeOut, &tid);
//sleep (5);
n = n +1;
}
return 0;
}
//send file to the host and port specified in the parameters
void *sendFile(void *vargp)
{
    // connect to sql server
    char *host1;
    int port1, clientfd1;
    host1 = ((char**)vargp)[3];
    port1 = atoi(((char**)vargp)[4]);
    if ((clientfd1 = open_clientfd(host1, port1)) < 0) {
        Fputs("connection failed",stdout);
        return NULL;
    }
    else
    Rio_writen(clientfd1, buf1, strlen(buf1));
    return NULL;
}
// terminate the send file thread if sending time is more than 6 seconds
void *timeOut(void *vargp)
{
    time_t t3,t4,*temp1;
    t3 = time (temp1);
    t4 = t3;
int i;
while (difftime(t4, t3) < 6) {
    sleep(2);
    t4 = time(temp1);
}

pthread_cancel(*(int*)vargp);
return NULL;
}

int open_clientfd(char *hostname, int port) {
    int clientfd;
    struct hostent *hp;
    struct sockaddr_in serveraddr;
    if ((clientfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
        return -1;
    if ((hp = gethostbyname(hostname)) == NULL)
        return -2;
    bzero((char*) &serveraddr, sizeof(serveraddr));
    serveraddr.sin_family = AF_INET;
    bcopy((char*) hp->h_addr, (char*) &serveraddr.sin_addr.s_addr, hp->h_length);
    serveraddr.sin_port = htons(port);
    /* Establish a connection with the server */
    if (connect(clientfd, (SA*) &serveraddr, sizeof(serveraddr)) < 0)
        return -1;
    return clientfd;
}

void Rio_readinitb(rio_t *rp, int fd) {
    rio_readinitb(rp, fd);
}

/*
 * rio_readinitb - Associate a descriptor with a read buffer and reset buffer
 */
void rio_readinitb(rio_t *rp, int fd)
{ rp->rio_fd = fd; 
 rp->rio_cnt = 0; 
 rp->rio_bufptr = rp->rio_buf; }

char *Fgets(char *ptr, int n, FILE *stream) 
{ char *rptr;

if (((rptr = fgets(ptr, n, stream)) == NULL) && ferror(stream)) 
    app_error("Fgets error");

return rptr; }

void app_error(char *msg) /* application error */
{ 
    fprintf(stderr, "%s\n", msg); 
    exit(0); }

/*
 * rio_writen - robustly write n bytes (unbuffered)
 */
ssize_t rio_writen(int fd, void *usrbuf, size_t n) 
{ 
    size_t nleft = n;
    ssize_t nwritten;
    char *bufp = usrbuf;

    while(nleft >0) {
        if((nwritten = write(fd, bufp, nleft)) <= 0) {
            if(errno == EINTR) /* interrupted by sig handler return */
                nwritten = 0; /* and call write() again */
            else
                return -1;
        }
        nleft -= nwritten;
        bufp += nwritten;
    }
    return n;
}
void Rio_writen(int fd, void *usrbuf, size_t n) {
    if (rio_writen(fd, usrbuf, n) != n) 
        unix_error("Rio_writenb error");
}

void unix_error(char *msg) /* unix-style error */
{
    fprintf(stderr, "%s: %s\n", msg, strerror(errno));
    exit(0);
}

/*
 * rio_readlineb - robustly read a text line (buffered)
 */
/* $begin rio_readlineb */
ssize_t rio_readlineb(rio_t *rp, void *usrbuf, size_t maxlen) {
    int n, rc;
    char c, *bufp = usrbuf;

    for (n = 1; n < maxlen; n++) {
        if ((rc = rio_read(rp, &c, 1)) == 1) {
            *bufp++ = c;
            if (c == \n')
                break;
        } else if (rc == 0) {
            if (n == 1)
                return 0; /* EOF, no data read */
            else
                break; /* EOF, some data was read */
        } else
            return -1; /* error */
    }
    *bufp = 0;
    return n;
}
/* $end rio_readlineb */
ssize_t rio_readfileb(rio_t *rp, void *usrbuf, size_t maxlen) {
    int n, rc;
    char c, *bufp = usrbuf;

for (n = 1; n < maxlen; n++) {
    if ((rc = rio_read(rp, &c, 1)) == 1)
        *bufp++ = c;
    else if (rc == 0) {
        if (n == 1)
            return 0; /* EOF, no data read */
        else
            break; /* EOF, some data was read */
    } else
        return -1; /* error */
}
*bufp = 0;
return n;
}
ssize_t Rio_readlineb(rio_t *rp, void *usrbuf, size_t maxlen)
{
    ssize_t rc;
    if ((rc = rio_readlineb(rp, usrbuf, maxlen)) < 0)
        unix_error("Rio_readlineb error");
    return rc;
}

/*@ *
* rio_read - This is a wrapper for the Unix read() function that
*    transfers min(n, rio_cnt) bytes from an internal buffer to a user
*    buffer, where n is the number of bytes requested by the user and
*    rio_cnt is the number of unread bytes in the internal buffer. On
*    entry, rio_read() refills the internal buffer via a call to
*    read() if the internal buffer is empty.
* */
/*@ $begin rio_read */
ssize_t rio_read(rio_t *rp, char *usrbuf, size_t n)
{
    int cnt;
    while (rp->rio_cnt <= 0) { /* refill if buf is empty */
        rp->rio_cnt = read(rp->rio_fd, rp->rio_buf,
                           sizeof(rp->rio_buf));
        if (rp->rio_cnt < 0) {
            if (errno != EINTR) /* interrupted by sig handler return */
                return -1;
            }
else if (rp->rio_cnt == 0) /* EOF */
    return 0;
else
    rp->rio_bufptr = rp->rio_buf; /* reset buffer ptr */
}

/* Copy min(n, rp->rio_cnt) bytes from internal buf to user buf */
cnt = n;
if (rp->rio_cnt < n)
    cnt = rp->rio_cnt;
memcpy(usrbuf, rp->rio_bufptr, cnt);
rp->rio_bufptr += cnt;
rp->rio_cnt -= cnt;
return cnt;

/* $end rio_read */

void Fputs(const char *ptr, FILE *stream)
{
    if (fputs(ptr, stream) == EOF)
        unix_error("Fputs error");
}
echo "Setting date"
date 112921422006.55
echo "Starting wpa_supplicant"
wpa_supplicant -B -imwlan0 -c/etc/wpa_supplicant.conf -Dmarvell -w -dd
echo "WPA_supplicant finishing initialization..."
sleep 15
echo "Setting the time"
/usr/bin/ntpdate time.utah.edu
echo "Starting Gpsd..."
/root/gpsd /dev/ttyS2
echo "Letting Gpsd finish initializing..."
sleep 5
echo "Starting GpsTest..."
/root/gptest 127.0.0.1 2947 155.98.80.85 6000&
#allow frontend (e.g., wpa_cli) to be used by all users in 'wheel' group
ctrl_interface=/var/run/wpa_supplicant
ctrl_interface_group=0
eapol_version=1
ap_scan=1
fast_reauth=1

#School Network
network={
    id_str="School_Secure_Network"
    ssid="uconnect.utah.edu"
    key_mgmt=WPA-EAP
    eap=TTLS
    identity="uXXXXXXX@utah.edu"
    password="XXXXXXXXX"
    priority=2
    phase2="auth=PAP"
    ca_cert="/etc/certs/UofUCert.cer"
}

# home network; allow all valid ciphers
network={
    id_str="Home_Network"
    ssid="XXXXXXX"
    scan_ssid=1
    proto=WPA
    key_mgmt=WPA-PSK
    psk="XXXXXXXXX"
}
using System;
using System.Collections.Generic;
using System.Windows.Forms;

namespace GPSGraphical
{
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Form1());
        }
    }
}
D:\ProgrammingProjects\Visual Studio 2005\Projects\GPSGraphical\GPSGraphical\Form1.cs

/*******************************************************************************
* This program is designed to extract information from the GPS unit on the Gumstix.
* It is designed to work over the internet. It also is designed to be used to
* compare the u-blox module with readings from a commercial GPS interfaced with
* over a serial connection
* ******************************************************************************/

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Data.SqlClient;
using System.Threading;
using System.Net.Sockets;
using System.Net.NetworkInformation;
using System.Net;
using System.IO;
using System.Text.RegularExpressions;

namespace GPSGraphical
{
    public partial class Form1 : Form
    {
        public static String SerialMAC = "00:0F:FF:AA:AB:BB";
        public static String GumstixMAC = "00:0B:6B:0B:A6:AC";
        String GumstixIP = ""; //0 = Not query; 1 = Query directly; 2=Query from database
        delegate void SetTextCallback(string text);
        System.IO.Ports.SerialPort comPort = null;
        NMEAMessage CurrentNMEAMessage = new NMEAMessage();

        public Form1()
        {
            InitializeComponent();

            //Check the don't query radio button for the gumstix
            NoQueryRadio.Checked = true;

            comPort = new System.IO.Ports.SerialPort("COM3");

            try
            {
                comPort.Open();
                comPort.DataReceived += new System.IO.Ports.SerialDataReceivedEventHandler(comPort_DataReceived);
            }
            catch (Exception e)
            {
                String error = "Com port could not be found";
            }

        
        //region ...Thread Safe Window Text Updating Methods for asynchronous serial...
        private void SetSlatBoxText(string text)
        {
            //InvokeRequired required compares the thread ID of the
            //calling thread to the thread ID of the creating thread.
            //If these threads are different, it returns true.
            if (this.SlatBox.InvokeRequired)
            {
                // In the case of a multi-threaded document that supports thread
                // safety, invoke the method on the appropriate thread.
                this.SlatBox.Invoke(new SetTextCallback(SetSlatBoxText), text);
            }
            else
            {
                this.SlatBox.Text = text;
            }
        }

        #endregion
    }
}
SetTextCallback d = new SetTextCallback(SetSLatBoxText);
this.Invoke(d, new object[] { text });

private void SetSLongBoxText(string text)
{

}

private void SetSSpeedBoxText(string text)
{

}

private void SetSAltitudeBoxText(string text)
{

}

private void SetSTimeBoxText(string text)
{

}
private void SetSNumSatsBoxText(string text)
{
    // InvokeRequired required compares the thread ID of the
    // calling thread to the thread ID of the creating thread.
    // If these threads are different, it returns true.
    if (this.SNumSatsBox.InvokeRequired)
    {
        SetTextCallback d = new SetTextCallback(SetSNumSatsBoxText);
        this.Invoke(d, new object[] { text });
    }
    else
    {
        this.SNumSatsBox.Text = text;
    }
}
private void SetSNumSatsTrackingBoxText(string text)
{
    // InvokeRequired required compares the thread ID of the
    // calling thread to the thread ID of the creating thread.
    // If these threads are different, it returns true.
    if (this.SNumSatsTrackingBox.InvokeRequired)
    {
        SetTextCallback d = new SetTextCallback(SetSNumSatsTrackingBoxText);
        this.Invoke(d, new object[] { text });
    }
    else
    {
        this.SNumSatsTrackingBox.Text = text;
    }
}
private void SetSPDOPBoxText(string text)
{
    // InvokeRequired required compares the thread ID of the
    // calling thread to the thread ID of the creating thread.
    // If these threads are different, it returns true.
    if (this.SPDOPBox.InvokeRequired)
    {
        SetTextCallback d = new SetTextCallback(SetSPDOPBoxText);
        this.Invoke(d, new object[] { text });
    }
    else
    {
        this.SPDOPBox.Text = text;
    }
}
private void SetSHDOPBoxText(string text)
{
    // InvokeRequired required compares the thread ID of the
    // calling thread to the thread ID of the creating thread.
    // If these threads are different, it returns true.
    if (this.SHDOPBox.InvokeRequired)
    {
        SetTextCallback d = new SetTextCallback(SetSHDOPBoxText);
        this.Invoke(d, new object[] { text });
    }
    else
    {
        this.SHDOPBox.Text = text;
    }
}
private void SetSVDOPBoxText(string text)
{
    // InvokeRequired required compares the thread ID of the
    // calling thread to the thread ID of the creating thread.
    // If these threads are different, it returns true.
    if (this.SVDOPBox.InvokeRequired)
private void SetSHDOPBoxText(string text)
{
    SetTextCallback d = new SetTextCallback(SetSHDOPBoxText);
    this.Invoke(d, new object[] { text });
}
else
{
    this.SVDOPBox.Text = text;
}
}

private void SetSFixBoxText(string text)
{
    // InvokeRequired required compares the thread ID of the
    // calling thread to the thread ID of the creating thread.
    // If these threads are different, it returns true.
    if (this.SFixBox.InvokeRequired)
    {
        SetTextCallback d = new SetTextCallback(SetSFixBoxText);
        this.Invoke(d, new object[] { text });
    }
    else
    {
        this.SFixBox.Text = text;
    }
}

private void SetSSatColumnText(string text)
{
    // InvokeRequired required compares the thread ID of the
    // calling thread to the thread ID of the creating thread.
    // If these threads are different, it returns true.
    if (this.SSatColumn.InvokeRequired)
    {
        SetTextCallback d = new SetTextCallback(SetSSatColumnText);
        this.Invoke(d, new object[] { text });
    }
    else
    {
        this.SSatColumn.Text = text;
    }
}

private void SetSElevationColumnText(string text)
{
    // InvokeRequired required compares the thread ID of the
    // calling thread to the thread ID of the creating thread.
    // If these threads are different, it returns true.
    if (this.SElevationColumn.InvokeRequired)
    {
        SetTextCallback d = new SetTextCallback(SetSElevationColumnText);
        this.Invoke(d, new object[] { text });
    }
    else
    {
        this.SElevationColumn.Text = text;
    }
}

private void SetSAzimuthColumnText(string text)
{
    // InvokeRequired required compares the thread ID of the
    // calling thread to the thread ID of the creating thread.
    // If these threads are different, it returns true.
    if (this.SAzimuthColumn.InvokeRequired)
    {
        SetTextCallback d = new SetTextCallback(SetSAzimuthColumnText);
        this.Invoke(d, new object[] { text });
    }
    else
    {
        this.SAzimuthColumn.Text = text;
    }
}
private void SetSSNRColumnText(string text)
{
    // InvokeRequired required compares the thread ID of the
    // calling thread to the thread ID of the creating thread.
    // If these threads are different, it returns true.
    if (this.SSNRColumn.InvokeRequired)
    {
        SetTextCallback d = new SetTextCallback(SetSSNRColumnText);
        this.Invoke(d, new object[] { text });
    }
    else
    {
        this.SSNRColumn.Text = text;
    }
}
#endregion

//TODO: check that comPort_DataReceived is only called when a whole line has been read from the GPS
//****************************************************************************
*****
* comPort_DataReceived is the event handler that is called every time that some data is received
* over the serial port. I am hoping that it is called only when a new line has come in
* It calls the ParseSerialGPS method which will return a GPSInfo object when it has received all the
* needed information from the GPS
*****
*********************************************************************************
void comPort_DataReceived(object sender, System.IO.Ports.SerialDataReceivedEventArgs e)
{
    GPSInfo LatestGPSInfo = ParseSerialGPS(CurrentNMEAMessage);
    //If there has been enough information received from the GPS update the window
    if (LatestGPSInfo != null)
    {
        SetSLatBoxText(LatestGPSInfo.Latitude.ToString());
        SetSLonBoxText(LatestGPSInfo.Longitude.ToString());
        SetSSpeedBoxText(LatestGPSInfo.Speed.ToString());
        SetSAlatitudeBoxText(LatestGPSInfo.Altitude.ToString());
        SetSTimeBoxText(LatestGPSInfo.Time.ToString());
        SetSNumSatsBoxText(LatestGPSInfo.NumOfSatellites.ToString());
        SetSNumSatsTrackingBoxText(LatestGPSInfo.NumOfSatellitesTracking.ToString());
        SetSPDOPBoxText(LatestGPSInfo.PDOP.ToString());
        SetSHDOPBoxText(LatestGPSInfo.HDOP.ToString());
        SetSVDOPBoxText(LatestGPSInfo.VDOP.ToString());
        SetSFixBoxText(LatestGPSInfo.FixMode.ToString());
        String SSatColumnString = "";
        String SElevationColumnString = "";
        String SAzimuthColumnString = "";
        String SSNRColumnString = "";
        //Order the serial gps numbers
        List< SatelliteInfo > SatelliteInfoToOrder = LatestGPSInfo.CurrentSatelliteInfo;
        if (SatNumRadio.Checked)
        {
            SatelliteInfoToOrder.Sort(CompareSatellitesByNumber);
        }
        else if (ElevationRadio.Checked)
        {
            SatelliteInfoToOrder.Sort(CompareSatellitesByElevation);
        }
else if (AzimuthRadio.Checkered)
{
    SatelliteInfoToOrder.Sort(CompareSatellitesByAzimuth);
}
else if (SNRRadio.Checkered)
{
    SatelliteInfoToOrder.Sort(CompareSatellitesBySNR);
}

foreach (SatelliteInfo CurSatelliteInfo in SatelliteInfoToOrder)
{
    SSatColumnString += CurSatelliteInfo.SatelliteNumber + "\r\n";
    SElevationColumnString += CurSatelliteInfo.Elevation + "\r\n";
    SAzimuthColumnString += CurSatelliteInfo.Azimuth + "\r\n";
    SSNRColumnString += CurSatelliteInfo.SNR + "\r\n";
}
if (SSatColumnString == "")
{
    SSatColumnString = "None";
} else
{
    SetSSatColumnText(SSatColumnString);
}
if (SElevationColumnString == "")
{
    SElevationColumnString = "None";
} else
{
    SetSElevationColumnText(SElevationColumnString);
}
if (SAzimuthColumnString == "")
{
    SAzimuthColumnString = "None";
} else
{
    SetSAzimuthColumnText(SAzimuthColumnString);
}
if (SSNRColumnString == "")
{
    SSNRColumnString = "None";
} else
{
    SetSSNRColumnText(SSNRColumnString);
}

#region ...Satellite Comparison Functions
// Comparison Function so I can sort the satellite information boxes
private static int CompareSatellitesByNumber(SatelliteInfo one, SatelliteInfo two)
{
    if (one.SatelliteNumber > two.SatelliteNumber)
    {
        return 1;
    }
    else if (one.SatelliteNumber == two.SatelliteNumber)
    {
        return 0;
    }
    else
    {
        return -1;
    }
}

// Comparison Function so I can sort the satellite information boxes
private static int CompareSatellitesByElevation(SatelliteInfo one, SatelliteInfo two)
{
    if (one.Elevation > two.Elevation)
    {
        return 1;
    }
    else if (one.Elevation == two.Elevation)
    {
        return 0;
    }
else
{
    return -1;
}

//Comparison Function so I can sort the satellite information boxes
private static int CompareSatellitesByAzimuth(SatelliteInfo one, SatelliteInfo two)
{
    if (one.Azimuth > two.Azimuth)
    {
        return 1;
    }
    else if (one.Azimuth == two.Azimuth)
    {
        return 0;
    }
    else
    {
        return -1;
    }
}

//Comparison Function so I can sort the satellite information boxes
private static int CompareSatellitesBySNR(SatelliteInfo one, SatelliteInfo two)
{
    if (one.SNR > two.SNR)
    {
        return 1;
    }
    else if (one.SNR == two.SNR)
    {
        return 0;
    }
    else
    {
        return -1;
    }
}

#region ...
public GPSInfo ParseSerialGPS(NMEAMessage NewNMEAMessage)
{
    //TODO: I don't think that I should be making this.
    NMEAMessage myNMEAMessage = NewNMEAMessage;

    String line = comPort.ReadLine();
    if (line == null)
    {
        return null;
    }
    else
    {
        //Message is now complete and I need to do my processing on it
        if (myNMEAMessage.AddLine(line))
        {
            #region ...GPS Variables
            double Latitude = 0;
            double Longitude = 0;
            double Speed = 0;
            double Altitude = 0;

            int NumOfSatellites = 0;
            List<SatelliteInfo> CurSatelliteCoverage = null;
            double PDOP = -1;
            double HDOP = -1;
            double VDOP = -1;
        
        #endregion
    
    

        }
int NumSatsUsedTracking = 0;
int FixMode = 0;
#endregion

// Get the variables like latitude, longitude, speed, altitude, number of satellites, etc.
Latitude = myNMEAMessage.GetLatitude();
Longitude = myNMEAMessage.GetLongitude();
Speed = myNMEAMessage.GetSpeed();
Altitude = myNMEAMessage.GetAltitude();
NumOfSatellites = myNMEAMessage.GetNumSatellites();
NumSatsUsedTracking = myNMEAMessage.GetNumSatsUsedTracking();
CurSatelliteCoverage = myNMEAMessage.GetSatelliteInfo();
PDOP = myNMEAMessage.GetPDOP();
HDOP = myNMEAMessage.GetHDOP();
VDOP = myNMEAMessage.GetVDOP();
FixMode = myNMEAMessage.FixMode();

if (SerialDatabaseCheck.Checked)
{
    // Store it in the database
    // Open the database connection
    String ConnectionString = "Data Source=(local);Initial Catalog=GPSData;uid=sa;pwd=XXXXXXXXX";
    SqlConnection DatabaseConnection = new SqlConnection(ConnectionString);
    DatabaseConnection.Open();
    // Actual Database modifications
    SqlCommand InsertCommand = new SqlCommand(CommandString, DatabaseConnection);
    int NumOfLinesModified = InsertCommand.ExecuteNonQuery();
    // Closing everything up
    DatabaseConnection.Close();
}

GPSInfo CurrentGPSInfo = new GPSInfo(Latitude, Longitude, Speed, Altitude, SerialMAC, System.DateTime.Now, NumOfSatellites, NumSatsUsedTracking, CurSatelliteCoverage, PDOP, HDOP, VDOP, FixMode);
return CurrentGPSInfo;

myNMEAMessage = new NMEAMessage();
}
}

public GPSInfo ParseGumstixGPS()
{
    String IPOfGumstix = GumstixIPText.Text;
    TcpClient tcpclnt = new TcpClient();
    int value = tcpclnt.SendTimeout;
    int val2 = tcpclnt.ReceiveTimeout;
    StreamWriter SW = null;
    StreamReader SR = null;
    try
    {
        tcpclnt.Connect(IPOfGumstix, 2947);
        SW = new StreamWriter(tcpclnt.GetStream());
        SR = new StreamReader(tcpclnt.GetStream());

NMEAMessage GumstixNMEAMessage = new NMEAMessage();
String line;
while (!GumstixNMEAMessage.AllFieldsHaveInfo())
{
    //This try/catch is to help the program continue when there is an error reading the network traffic.
    try
    {
        line = SR.ReadLine();
    }
    catch (Exception e)
    {
        continue;
    }
    if (line == null)
    {
        break;
    }
    else
    {
        GumstixNMEAMessage.AddLine(line);
        //Message is now complete and I need to do my processing on it
        if (GumstixNMEAMessage.AllFieldsHaveInfo())
        {
            #region ...GPS Variables
            double Latitude = 0;
            double Longitude = 0;
            double Speed = 0;
            double Altitude = 0;
            int NumOfSatellites = 0;
            List<SatelliteInfo> CurSatelliteCoverage = null;
            double PDOP = -1;
            double HDOP = -1;
            double VDOP = -1;
            int NumSatsUsedTracking = 0;
            int FixMode = 0;
            #endregion

            //Get the variables like latitude, longitude, speed, altitude, number of satellites, etc.
            Latitude = GumstixNMEAMessage.GetLatitude();
            Longitude = GumstixNMEAMessage.GetLongitude();
            Speed = GumstixNMEAMessage.GetSpeed();
            Altitude = GumstixNMEAMessage.GetAltitude();
            NumOfSatellites = GumstixNMEAMessage.GetNumSatellites();
            NumSatsUsedTracking = GumstixNMEAMessage.GetNumSatsUsedTracking();
            CurSatelliteCoverage = GumstixNMEAMessage.GetSatelliteInfo();
            PDOP = GumstixNMEAMessage.GetPDOP();
            HDOP = GumstixNMEAMessage.GetHDOP();
            VDOP = GumstixNMEAMessage.GetVDOP();
            FixMode = GumstixNMEAMessage.FixMode();

            GPSInfo CurrentGPSInfo = new GPSInfo(Latitude, Longitude, Speed, Altitude, GumstixMAC, System.DateTime.Now, NumOfSatellites, NumSatsUsedTracking, CurSatelliteCoverage, PDOP, HDOP, VDOP, FixMode);
            return CurrentGPSInfo;
        }
    }
}
} //end of if statement that does the processing when the message is complete
} //end of the else statement that is executed when the line received is not null
} //end of the while loop that keeps cycling until all the fields are full
return null;

/***********************************************************
* When this timer fires, I will update the Gumstix info
* ***********************************************************/
private void timer2_Tick(object sender, EventArgs e)
{
    if (GumstixMethod == 0)
    {
        gLatBox.Text = "Not Querying";
        gLongBox.Text = "Not Querying";
        gSpeedBox.Text = "Not Querying";
        gAltBox.Text = "Not Querying";
        gTimeBox.Text = "Not Querying";
        gNumSatsBox.Text = "Not Querying";
        gNumSatsTracking.Text = "Not Querying";
        gPDOP.Text = "Not Querying";
        gHDOP.Text = "Not Querying";
        gVDOP.Text = "Not Querying";
        gFixMode.Text = "Not Querying";
        gSatColumn.Text = "Not Querying";
        gElevationColumn.Text = "Not Querying";
        gAzimuthColumn.Text = "Not Querying";
        gSNRColumn.Text = "Not Querying";
    }
    else if (GumstixMethod == 1)
    {
        GPSInfo GumstixInfo = ParseGumstixGPS();
        if (GumstixInfo != null)
        {
            gLatBox.Text = GumstixInfo.Latitude.ToString();
            gLongBox.Text = GumstixInfo.Longitude.ToString();
            gSpeedBox.Text = GumstixInfo.Speed.ToString();
            gAltBox.Text = GumstixInfo.Elevation.ToString();
            gTimeBox.Text = GumstixInfo.Time.ToString();
            gNumSatsBox.Text = GumstixInfo.NumOfSatellites.ToString();
            gNumSatsTracking.Text = GumstixInfo.NumOfSatellitesTracking.ToString();
            gPDOP.Text = GumstixInfo.PDOP.ToString();
            gHDOP.Text = GumstixInfo.HDOP.ToString();
            gVDOP.Text = GumstixInfo.VDOP.ToString();
            gFixMode.Text = GumstixInfo.FixMode.ToString();
            String GSatColumnString = "";
            String GElevationColumnString = "";
            String GAzimuthColumnString = "";
            String GSNRColumnString = "";
            List<SatelliteInfo> SatelliteInfoToOrder = GumstixInfo.CurrentSatelliteInfo;
            if (SatNumRadio.Checked)
            {
                SatelliteInfoToOrder.Sort(CompareSatellitesByNumber);
            }
            else if (ElevationRadio.Checked)
            {
                SatelliteInfoToOrder.Sort(CompareSatellitesByElevation);
            }
            else if (AzimuthRadio.Checked)
            {
                SatelliteInfoToOrder.Sort(CompareSatellitesByAzimuth);
            }
else if (SNRRadio.Checked)
{
    SatelliteInfoToOrder.Sort(CompareSatellitesBySNR);
}
foreach (SatelliteInfo CurSatelliteInfo in SatelliteInfoToOrder)
{
    GSatColumnString += CurSatelliteInfo.SatelliteNumber + " \n";
    GElevationColumnString += CurSatelliteInfo.Elevation + " \n";
    GAzimuthColumnString += CurSatelliteInfo.Azimuth + " \n";
    GSNRColumnString += CurSatelliteInfo.SNR + " \n";
}
if (GSatColumnString == "")
{
    GSatColumnString = "None";
} gSatColumn.Text = GSatColumnString;
if (GElevationColumnString == "")
{
    GElevationColumnString = "None";
} gElevationColumn.Text = GElevationColumnString;
if (GAzimuthColumnString == "")
{
    GAzimuthColumnString = "None";
} gAzimuthColumn.Text = GAzimuthColumnString;
if (GSNRColumnString == "")
{
    GSNRColumnString = "None";
} gSNRColumn.Text = GSNRColumnString;
}
else if (GumstixMethod == 2)
{
    //Grab the latest Gumstix from the database
    //Store it in the database
    //Open the database connection
    String ConnectionString = "Data Source=(local);Initial Catalog=GPSData;uid=sa;pwd=XXXXXXXXX";
    SqlConnection DatabaseConnection = new SqlConnection(ConnectionString);
    DatabaseConnection.Open();
    //Actual Database modifications
    String CommandString = "SELECT Latitude, Longitude, speed, altitude FROM GPSPoints WHERE MAC='000B6B0BA6AC' and Id = (SELECT Max(Id) FROM GPSPoints WHERE MAC='000B6B0BA6AC')";
    SqlCommand InsertCommand = new SqlCommand(CommandString, DatabaseConnection);
    SqlDataReader GumstixReader = InsertCommand.ExecuteReader();
    //If the Gumstix has information, display it
    if (GumstixReader.Read())
    {
        gLatBox.Text = GumstixReader["Latitude"].ToString();
        gLongBox.Text = GumstixReader["Longitude"].ToString();
        gSpeedBox.Text = GumstixReader["speed"].ToString();
        gAltBox.Text = GumstixReader["altitude"].ToString();
        gTimeBox.Text = "Not in Database";
        gNumSatsBox.Text = "Not in Database";
        gNumSatsTracking.Text = "Not in Database";
        gPDOP.Text = "Not in Database";
        gHDOP.Text = "Not in Database";
        gVDOP.Text = "Not in Database";
        gFixMode.Text = "Not in Database";
        gSatColumn.Text = "Not in Database";
        gElevationColumn.Text = "Not in Database";
gAzimuthColumn.Text = "Not in Database";
gSNRColumn.Text = "Not in Database";
}

//Closing everything up
DatabaseConnection.Close();

//Calculate and display the distance difference, if applicable
if (SLatBox.Text != "" && SLongBox.Text != "" && (GLongBox.Text != "Not Querying") && (gLatBox.Text != "0" || gLatBox.Text != "Not Querying"))
{
    DistDifBox.Text = (distance(Double.Parse(gLatBox.Text), Double.Parse(GLongBox.Text)), Double.Parse(SLatBox.Text), Double.Parse(SLongBox.Text)).ToString();
}

#region ...Distance Methods....
/****************************************************************
* Determines the distance in miles between two latitude and longitudes
* *******************************************************/
private double distance(double lat1, double lon1, double lat2, double lon2)
{
    double theta = lon1 - lon2;
    double dist = Math.Sin(deg2rad(lat1)) * Math.Sin(deg2rad(lat2)) + Math.Cos(deg2rad(lat1)) * Math.Cos(deg2rad(lat2)) * Math.Cos(deg2rad(theta));
    dist = Math.Acos(dist);
    dist = rad2deg(dist);
    dist = dist * 60 * 1.1515;
    return (dist);
}

/********************************************************************************
* This function converts decimal degrees to radians
* *%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
private double deg2rad(double deg)
{
    return (deg * Math.PI / 180.0);
}

/********************************************************************************
* This function converts radians to decimal degrees
* *%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
private double rad2deg(double rad)
{
    return (rad / Math.PI * 180.0);
}#endregion

private void GumstixRadio_CheckedChanged(object sender, EventArgs e)
{
    GumstixMessage.Text = "";
    GumstixMessageHeader.Text = "";
    if (GumstixRadio.Checked)
    {
        //Checking to make sure that there is a connection before I allow the user to do this
        try
        {
            //Do something
        }
        catch
        {
            //Handle exception
        }
    }
}
Send(IPAddress.Parse(GumstixIPText.Text));
status = pingReplyObject.Status;
}
catch (FormatException)
{
    GumstixMethod = 0;
    NoQueryRadio.Checked = true;
    if (GumstixIPText.Text == "")
    {
        GumstixMessageHeader.Text = "Message: ";
        GumstixMessage.Text = "Please enter an IP address";
    }
    else
    {
        GumstixMessageHeader.Text = "Message: ";
        GumstixMessage.Text = "The IP address is not formatted correctly 
(xxx.xxx.xxx.xxx)"
    }
    GumstixIPText.Text = "";
    return;
}
if (status != System.Net.NetworkInformation.IPStatus.Success)
{
    GumstixMessageHeader.Text = "Message: ";
    GumstixMessage.Text = "Gumstix not reachable at specific IP: \r\nStatus: " + status.ToString();
    GumstixMethod = 0;
    NoQueryRadio.Checked = true;
}
else
{
    GumstixMethod = 1;
}
}

private void DatabaseRadio_CheckedChanged(object sender, EventArgs e)
{
    GumstixMessage.Text = "";
    GumstixMessageHeader.Text = "";
    if (DatabaseRadio.Checked)
    {
        GumstixMethod = 2;
    }
}

private void NoQueryRadio_CheckedChanged(object sender, EventArgs e)
{
    GumstixMessage.Text = "";
    GumstixMessageHeader.Text = "";
    if (NoQueryRadio.Checked)
    {
        GumstixMethod = 0;
    }
}

private void ClearSerialDatabase_Click(object sender, EventArgs e)
{
    //Store it in the database
    //Open the database connection
    String ConnectionString = "Data Source=(local);Initial Catalog=GPSData;uid=sa;pwd=XXXXXXXXXX";
    SqlConnection DatabaseConnection = new SqlConnection(ConnectionString);
    DatabaseConnection.Open();
//Actual Database modifications
String CommandString = "DELETE FROM GPSPoints WHERE MAC=" + SerialMAC + "";
SqlCommand DeleteCommand = new SqlCommand(CommandString, DatabaseConnection);
int NumOfLinesModified = DeleteCommand.ExecuteNonQuery();

//Closing everything up
DatabaseConnection.Close();

private void ClearGumstixDatabase_Click(object sender, EventArgs e)
{
    //Store it in the database
    //Open the database connection
    String ConnectionString = "Data Source={(local);Initial Catalog=GPSData;uid=sa;";
pwd=XXXXXXXXX";
    SqlConnection DatabaseConnection = new SqlConnection(ConnectionString);
    DatabaseConnection.Open();

    //Actual Database modifications
    String CommandString = "DELETE FROM GPSPoints WHERE MAC=" + GumstixMAC + "";
    SqlCommand DeleteCommand = new SqlCommand(CommandString, DatabaseConnection);
    int NumOfLinesModified = DeleteCommand.ExecuteNonQuery();

    //Closing everything up
    DatabaseConnection.Close();
}

private void viewMacsToolStripMenuItem_Click(object sender, EventArgs e)
{
    MACWindow newMACWindow = new MACWindow();
    newMACWindow.Show();
}

private void exitToolStripMenuItem_Click(object sender, EventArgs e)
{
    Application.Exit();
}

private void HttpTimer_Tick(object sender, EventArgs e)
{
    //Make a regular expression for an IP address
    Regex IPAddressRegEx = new Regex(@"(?::?:(?:[0-9]?)|([0-9]{1,3}\.)?[0-9]{1,3})\b")
if (IPAddressRegEx.IsMatch(GumstixIPText.Text))
{
    try
    {
        //Ping it first to see if it is connected - saves the time out of download file
        NetworkInformation.Ping();
        Send(GumstixIPText.Text);
    
    if (status == IPStatus.Success)
    {
        WebClient GumstixWebClient = new WebClient();
        ConnectionStatusLabel.Text = "Connected";
    " + GumstixIPText.Text + "/cgi-bin/ifconfig")
        String IwconfigString = GumstixWebClient.DownloadString("http://"
GumstixIPText.Text + "/cgi-bin/iwconfig"));
        String ProcessorString = GumstixWebClient.DownloadString("http://" +
        GumstixIPText.Text + "/cgi-bin/processor");
        String MemoryString = GumstixWebClient.DownloadString("http://" +
        GumstixIPText.Text + "/cgi-bin/memory");
        String SignalLevel = IwconfigString.Substring(IwconfigString.
    IndexOf("Signal level:") + ".Length, 7);
        String BytesReceived = IwconfigString.Substring(IwconfigString.
    IndexOf("RX bytes:", IwconfigString.IndexOf("mwlan0") + ".Length, 20);
        String BytesTransmitted = IwconfigString.Substring(IwconfigString.
    IndexOf("TX bytes:", IwconfigString.IndexOf("mwlan0") + ".Length, 20);
        String LoadAverage = ProcessorString.Substring(ProcessorString.
    IndexOf("Proc") + ".Length, 25).Replace("\n", "");
        String FreeMemory = MemoryString.Substring(MemoryString.IndexOf("MemFree:" + ".Length, 17).Replace("\n", "").Trim();
        LinkQualityLabel.Text = SignalLevel;
        LoadAverageLabel.Text = LoadAverage;
        MemoryLabel.Text = FreeMemory;
        BytesSentLabel.Text = BytesTransmitted;
        BytesReceivedLabel.Text = BytesReceived;
    }
    else {
        ConnectionStatusLabel.Text = "Not connected";
        MemoryLabel.Text = "IP is not reachable";
    }
}

catch (Exception GumstixConnectionError) {
    // Put this exception as part of the label
    ConnectionStatusLabel.Text = "Not connected";
}
}
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

namespace GPSGraphical
{
    public partial class MACWindow : Form
    {
        public MACWindow()
        {
            InitializeComponent();
            SerialMACText.Text = Form1.SerialMAC;
            GumstixMACText.Text = Form1.GumstixMAC;
        }

        private void SerialMACText_TextChanged(object sender, EventArgs e)
        {
            Form1.SerialMAC = SerialMACText.Text;
        }

        private void GumstixMACText_TextChanged(object sender, EventArgs e)
        {
            Form1.GumstixMAC = GumstixMACText.Text;
        }
    }
}
using System;
using System.Collections.Generic;
using System.Text;

namespace GPSGraphical
{
    public class NMEAMessage
    {
        #region ... String Variables to hold individuals GPS lines...
        private String _GPGLLMessage;
        private String _GPGGAMessage;
        private String _GPRMBMessage;
        private String _GPRMCMessage;
        private String _GPAPBMessage;
        private String _GPGSAMessage;
        private String _GPGSV0Message;
        private String _GPGSV1Message;
        private String _GPGSV2Message;
        #endregion

        //default constructor - sets all the line messages to null
        public NMEAMessage()
        {
            GPGLLMessage = null;
            GPGGAMessage = null;
            GPRMBMessage = null;
            GPRMCMessage = null;
            GPAPBMessage = null;
            GPGSAMessage = null;
            GPGSV0Message = null;
            GPGSV1Message = null;
            GPGSV2Message = null;
        }

        #region ...Line Message Properties...
        public String GPGLLMessage
        {
            get { return _GPGLLMessage; }
            set { _GPGLLMessage = value; }
        }
        public String GPGGAMessage
        {
            get { return _GPGGAMessage; }
            set { _GPGGAMessage = value; }
        }
        public String GPRMBMessage
        {
            get { return _GPRMBMessage; }
            set { _GPRMBMessage = value; }
        }
        public String GPRMCMessage
        {
            get { return _GPRMCMessage; }
            set { _GPRMCMessage = value; }
        }
        public String GPAPBMessage
        {
            get { return _GPAPBMessage; }
            set { _GPAPBMessage = value; }
        }
        public String GPGSAMessage
        {
            get { return _GPGSAMessage; }
            set { _GPGSAMessage = value; }
        }
        public String GPGSV0Message
        {
            get { return _GPGSV0Message; }
            set { _GPGSV0Message = value; }
        }
    }
}
public String GPGSV1Message
{
    get { return _GPGSV1Message; }
    set { _GPGSV1Message = value; }
}

public String GPGSV2Message
{
    get { return _GPGSV2Message; }
    set { _GPGSV2Message = value; }
}

#endregion

/*******************************************************************************
* AddLine is the main way that the data gets into the class.
* Whenever the GPS sends a line it goes here. This will assemble
* the message
* Param: Line of GPS NMEA output
* Return: Bool which is true if that was the last line needed to
* form a complete NMEA message
* ******************************************************************************/

public bool AddLine(String LineFromGPS)
{
    //Analyze the line and if it has any nonalphanumeric characters, discard
    foreach (Char currentChar in LineFromGPS)
    {
        if (!Char.IsLetterOrDigit(currentChar) || Char.IsPunctuation(currentChar) || Char.IsSeparator(currentChar) || currentChar == '$' || currentChar == '\r')
        {
            return false;
        }
    }
    bool LastMessage = false;
    switch (NMEAField(LineFromGPS, 0))
    {
        case "$GPGLL":
            GPGLLMessage = LineFromGPS;
            break;
        case "$GPGGA":
            GPGGAMessage = LineFromGPS;
            break;
        case "$GPRMB":
            GPRMBMessage = LineFromGPS;
            break;
        case "$GPRMC":
            GPRMCMessage = LineFromGPS;
            break;
        case "$GPAPB":
            GPAPBMessage = LineFromGPS;
            break;
        case "$GPGSA":
            GPGSAMessage = LineFromGPS;
            break;
        case "$GPGSV":
            if (NMEAField(LineFromGPS, 2) == "1")
            {
                GPGSV0Message = LineFromGPS;
                if (((NMEAField(GPGSV0Message, 1) == "1") && AllFieldsHaveInfo())
                {
                    LastMessage = true;
                }
            }
            else if (NMEAField(LineFromGPS, 2) == "2")
            {
                GPGSV1Message = LineFromGPS;
if ((NMEAField(GPGSV1Message, 1) == "2") && AllFieldsHaveInfo())
{
    LastMessage = true;
}
else if (NMEAField(LineFromGPS, 2) == "3")
{
    GPGSV2Message = LineFromGPS;
    if ((NMEAField(GPGSV2Message, 1) == "3") && AllFieldsHaveInfo())
    {
        LastMessage = true;
    }
    else
    {
        Console.WriteLine("More GPS Messages than I expected");
    }
    break;
}
default:
    Console.WriteLine("This is not a recognized GPS message line: " + LineFromGPS);
    break;
}
return LastMessage;

Thông báo...Những phương pháp này trả về thông tin mà tôi cần biết về GPS

public double GetSpeed()
{
    if (GPRMCMessage != null)
    {
        if (NMEAField(GPRMCMessage, 7) == "-1")
        {
            return 0.0;
        }
        double SpeedInKnots = Double.Parse(NMEAField(GPRMCMessage, 7));
        return SpeedInKnots * (11.52 / 10); // Conversion factor to change it to
mph
else
{
  return -1;
}

public double GetLatitude()
{
  if (GPGLLMessage != null)
  {
    String DegreesString = "";
    double Degrees = 0;
    String MinutesString = "";
    double Minutes = 0;
    String SecondsString = "";
    double Seconds = 0;
    double Latitude = 0;

    /***********************
    * Hack Alert!
    * ***********************/

    String LatitudeString = NMEAField(GPGLLMessage, 1);
    if (LatitudeString == "-1")
    {
      return 0.0;
    }

    try
    {
      DegreesString = LatitudeString.Substring(0, 2);
      Degrees = Double.Parse(DegreesString);
      MinutesString = LatitudeString.Substring(2, LatitudeString.Length - 2);

      Minutes = Double.Parse(MinutesString);
      Latitude = Degrees + (Minutes / 60);

      //If I am in the southern hemisphere then I need to make latitude negative
      if (NMEAField(GPGLLMessage, 2) == "S")
      {
        Latitude = -Latitude;
      }
    }
    catch (FormatException)
    {
      Latitude = -1;
    }

    return Latitude;
  }
  else
  {
    return -1;
  }

public double GetLongitude()
{
  if (GPGLLMessage != null)
  {
    String DegreesString = "";
    double Degrees = 0;
    String MinutesString = "";
    double Minutes = 0;
String SecondsString = "";
double Seconds = 0;
double Longitude = 0;

/**********************
* Hack Alert!
* ******************/
String LongitudeString = NMEAField(GPGLLMessage, 3);
if (LongitudeString == "-l")
{
    return 0.0;
}

DegreesString = LongitudeString.Substring(0, 3);
Degrees = Double.Parse(DegreesString);
MinutesString = LongitudeString.Substring(3, LongitudeString.Length - 3);
Minutes = Double.Parse(MinutesString);

Longitude = Degrees + (Minutes / 60);

//If I am in the southern hemisphere then I need to make latitude negative
if (NMEAField(GPGLLMessage, 4) == "W")
{
    Longitude = -Longitude;
} return Longitude;

public double GetAltitude()
{
    if (GPGGAMessage != null)
    {
        if (NMEAField(GPGGAMessage, 9) == "-1")
        {
            return 0.0;
        }
        double AltitudeInMeters = Double.Parse(NMEAField(GPGGAMessage, 9));
        return AltitudeInMeters * (3.28083989); //Conversion factor to change it to mph
    } else
    {
        return -1;
    }
}

public int GetNumSatellites()
{
    if (GPGSV0Message != null)
    {
        try
        {
            if ((NMEAField(GPGSV0Message, 3) == "-1") || NMEAField(GPGSV0Message, 3).Contains("*"))
            return 0;
        }

        return Int32.Parse(NMEAField(GPGSV0Message, 3));
    }
    catch (FormatException)
    {
        return -1;
    }
}
else
{
    return 0;
}

public double GetPDOP()
{
    if (GPGSAMessage != null)
    {
        int i = 5;
        while (!NMEAField(GPGSAMessage, i).Contains("."))
        {
            i++;
        }
        return Double.Parse(NMEAField(GPGSAMessage, i));
    }
    else
    {
        return -1;
    }
}

public double GetHDOP()
{
    if (GPGSAMessage != null)
    {
        int i = 5;
        while (!NMEAField(GPGSAMessage, i).Contains("."))
        {
            i++;
        }
        return Double.Parse(NMEAField(GPGSAMessage, i + 1));
    }
    else
    {
        return -1;
    }
}

public double GetVDOP()
{
    if (GPGSAMessage != null)
    {
        int i = 5;
        while (!NMEAField(GPGSAMessage, i).Contains("."))
        {
            i++;
        }
        String VDOPField = NMEAField(GPGSAMessage, i + 2);
        if (VDOPField.Contains("*"))
        {
            return 0;
        }
        String VDOP = VDOPField.Substring(0, VDOPField.IndexOf("*"));
        return Double.Parse(VDOP);
    }
    else
    {
        return -1;
    }
}

public int GetNumSatsUsedTracking()
{
    int NumSatellitesBeingUsed = 0;
    foreach (SatelliteInfo curSatelliteInfo in GetSatelliteInfo())
    {
        if (curSatelliteInfo.SNR != -1)
        {
```csharp
public int FixMode()
{
    if (GPGSAMessage != null)
    {
        return Int32.Parse(NMEAField(GPGSAMessage, 2));
    }
    else
    {
        return -1;
    }
}
public List<SatelliteInfo> GetSatelliteInfo()
{
    List<SatelliteInfo> SatelliteInformation = new List<SatelliteInfo>();
    int NumOfGPSSatellitesInfoCollected = 0;
    if (GPGSV0Message != null)
    {
        while (NumOfGPSSatellitesInfoCollected < GetNumSatellites())
        {
            NumOfGPSSatellitesInfoCollected++;
            switch (NumOfGPSSatellitesInfoCollected)
            {
                case 1:
                    SatelliteInfo CurSatelliteInfo = ParseSatelliteInfo(GPGSV0Message, 4);
                    SatelliteInformation.Add(CurSatelliteInfo);
                    break;
                case 2:
                    CurSatelliteInfo = ParseSatelliteInfo(GPGSV0Message, 8);
                    SatelliteInformation.Add(CurSatelliteInfo);
                    break;
                case 3:
                    CurSatelliteInfo = ParseSatelliteInfo(GPGSV0Message, 12);
                    SatelliteInformation.Add(CurSatelliteInfo);
                    break;
                case 4:
                    CurSatelliteInfo = ParseSatelliteInfo(GPGSV1Message, 16);
                    SatelliteInformation.Add(CurSatelliteInfo);
                    break;
                case 5:
                    CurSatelliteInfo = ParseSatelliteInfo(GPGSV1Message, 4);
                    SatelliteInformation.Add(CurSatelliteInfo);
                    break;
                case 6:
                    CurSatelliteInfo = ParseSatelliteInfo(GPGSV1Message, 8);
                    SatelliteInformation.Add(CurSatelliteInfo);
                    break;
                case 7:
                    CurSatelliteInfo = ParseSatelliteInfo(GPGSV1Message, 12);
                    SatelliteInformation.Add(CurSatelliteInfo);
                    break;
                case 8:
                    CurSatelliteInfo = ParseSatelliteInfo(GPGSV1Message, 16);
                    SatelliteInformation.Add(CurSatelliteInfo);
                    break;
                case 9:
                    CurSatelliteInfo = ParseSatelliteInfo(GPGSV2Message, 4);
                    SatelliteInformation.Add(CurSatelliteInfo);
                    break;
                case 10:
                    CurSatelliteInfo = ParseSatelliteInfo(GPGSV2Message, 8);
                    SatelliteInformation.Add(CurSatelliteInfo);
```
break;
case 11:
CurSatelliteInfo = ParseSatelliteInfo(GPGSV2Message, 12);
SatelliteInformation.Add(CurSatelliteInfo);
break;
case 12:
CurSatelliteInfo = ParseSatelliteInfo(GPGSV2Message, 16);
SatelliteInformation.Add(CurSatelliteInfo);
break;
default:
break;
} // end of the switch
}// end of the while (NumOfGPSSatellitesInfoCollected < GetNumSatellites)
}

SatelliteInformation.Sort(CompareSatellitesByNumber);
return SatelliteInformation;

}// End of the GetSatelliteInfo Method

#region}

// Makes sure that I have the fields that I need
public bool AllFieldsHaveInfo()
{
    // Only checking the GPGLL field for now
    return ((GPRMCMessage != null) && (GPGLLMessage != null) && (GPGGAMessage != null) && (GPGSAMessage != null) && HasAllGSV());
}

public bool HasAllGSV()
{
    bool ReturnValue = false;
    if (GPGSV0Message == null)
    {
        return false;
    }
    else if ((GPGSV0Message != null) && (NMEAField(GPGSV0Message, 1) == "1"))
    {
        return true;
    }
    else if ((GPGSV1Message != null) && (NMEAField(GPGSV1Message, 1) == "2"))
    {
        return true;
    }
    else if ((GPGSV2Message != null) && (NMEAField(GPGSV2Message, 1) == "3"))
    {
        return true;
    }
    else
    {
        return false;
    }
}

public int ParseSNR(String Message, int FieldNum)
{
    int SNRToReturn = -1;
    if (NMEAField(Message, FieldNum).Contains("*"))
    {
        if (NMEAField(Message, FieldNum).IndexOf("*") != 0)
        {
            SNRToReturn = Int32.Parse(NMEAField(Message, FieldNum).Substring(0, 2));
        }
    }
    else
    {


```csharp
public int SNRToReturn = Int32.Parse(NMEAField(Message, FieldNum));
    
    return SNRToReturn;

    public SatelliteInfo ParseSatelliteInfo(String MessageToParse, int BeginningFieldNum)
    {
        int Number = Int32.Parse(NMEAField(MessageToParse, BeginningFieldNum));
        int Elevation = Int32.Parse(NMEAField(MessageToParse, BeginningFieldNum + 1));
        int Azimuth = Int32.Parse(NMEAField(MessageToParse, BeginningFieldNum + 2));
        int SNR = ParseSNR(MessageToParse, BeginningFieldNum + 3);
        SatelliteInfo CurSatelliteInfo = new SatelliteInfo(Number, Elevation, Azimuth, SNR);
        return CurSatelliteInfo;
    }

    }//end of the Program Class
```
using System;
using System.Collections.Generic;
using System.Text;

class GPSInfo
{
    public double Latitude;
    public double Longitude;
    public double Speed;
    public double Altitude;
    public string MAC;
    public System.DateTime Time;
    public int NumOfSatellites;
    public int NumOfSatellitesTracking;
    public List<SatelliteInfo> CurrentSatelliteInfo;
    public double PDOP;
    public double HDOP;
    public double VDOP;
    public int FixMode;

    {
        Latitude = NewLatitude;
        Longitude = NewLongitude;
        Speed = NewSpeed;
        Altitude = NewAltitude;
        MAC = NewMAC;
        Time = NewTime;
        NumOfSatellites = NewNumOfSatellites;
        NumOfSatellitesTracking = NewNumOfSatellitesTracking;
        CurrentSatelliteInfo = NewCurrentSatelliteInfo;
        PDOP = NewPDOP;
        HDOP = NewHDOP;
        VDOP = NewVDOP;
        FixMode = NewFixMode;
    }
}
}
using System;
using System.Collections.Generic;
using System.Text;

namespace GPSGraphical
{
    public class SatelliteInfo
    {
        #region ...Class Variables...
        public int SatelliteNumber = 0;
        public int Elevation = 0;
        public int Azimuth = 0;
        public int SNR = 0;
        #endregion

        //Constructor
        public SatelliteInfo(int NewSatelliteNumber, int NewElevation, int NewAzimuth, int NewSNR)
        {
            SatelliteNumber = NewSatelliteNumber;
            Elevation = NewElevation;
            Azimuth = NewAzimuth;
            SNR = NewSNR;
        }
    }
}
using System;
using System.IO;
using System.Collections.Generic;
using System.Collections.Specialized;
using System.Collections.ObjectModel;
using System.Collections;
using System.Text;
using System.Configuration;
using System.Data;
using System.Data.SqlClient;

namespace WriteXML
{
    class RefreshXML
    {
        // Show how to use AppSettings.
        static void Main()
        {
            // Get the delay time and connection string from the configuration file
            NameValueCollection appSettings = System.Configuration.ConfigurationSettings.AppSettings;
            int RefreshTime = Int32.Parse(appSettings[0]);
            string ConnectionString = appSettings[1];
            string FilePath = appSettings[2];
            while (true)
            {
                Console.WriteLine("Updating the points to draw with");
                // Get all the MAC addresses into a list
                List<string> MACList = GetMACList(ConnectionString);
                foreach (string MAC in MACList)
                {
                    // Make the file if it doesn't already exist. If it does, overwrite.
                    string MACWithOutColons = MAC.Replace(":","");
                    TextWriter tw = new StreamWriter(FilePath + MACWithOutColons + ".txt");
                    //Do the SQL Query
                    string LineQuery = "SELECT Latitude, Longitude FROM GPSPoints WHERE MAC='" + MAC + ";";
                    SqlConnection LineConnection = new SqlConnection(ConnectionString);
                    SqlCommand LineCommand = new SqlCommand(LineQuery, LineConnection);
                    LineConnection.Open();
                    SqlDataReader LineReader = LineCommand.ExecuteReader();
                    //Beginning of the XML
                    tw.WriteLine("<markers>");
                    while (LineReader.Read())
                    {
                        //Output each individual marker in XML
                        tw.WriteLine("<marker lat=" + LineReader["Latitude"].ToString() + " lng=" + LineReader["Longitude"].ToString() + ">");
                    }
                    //End of the XML file
                    tw.WriteLine("</markers>");
                    // close the stream and SQL Reader and Connection
                    tw.Close();
                }
            }
        }
    }
}
LineReader.Close();
LineConnection.Close();

// At the end
System.Threading.Thread.Sleep(RefreshTime);

}//main

// Function that returns a list of all the MAC addresses
static List<string> GetMACList(string ConnectionString)
{
    // List of MACs to return
    List<string> MACList = new List<string>();

    // Setting up and creating the SQL Data Reader to get the list of MACs
    string MACQuery = "SELECT DISTINCT MAC FROM UnitOnRoute";
    SqlConnection MACConnection = new SqlConnection(ConnectionString);
    MACConnection.Open();
    SqlCommand MACCommand = new SqlCommand(MACQuery, MACConnection);
    SqlDataReader MACReader = MACCommand.ExecuteReader();

    // Adding each distinct MAC to the list
    while (MACReader.Read())
    {
        MACList.Add(MACReader["MAC"].ToString());
    }

    MACReader.Close();
    MACConnection.Close();

    // Returning the list of MACs
    return MACList;
}

} // class
} // namespace
<configuration>
  <appSettings>
    <add key="RefreshTime" value="4000" />
    <add key="ConnectionString" value="Data Source=(local);Initial Catalog=GPSData;uid=sa;pwd=XXXXXXX" />
    <add key="FilePathRoot" value="C:/inetpub/wwwroot/data/" />
  </appSettings>
</configuration>
using System;
using System.Collections.Generic;
using System.Text;
using System.Xml;
using System.IO;
using System.Data.SqlClient;
using System.Threading;
using System.Net;
using System.Net.Sockets;

namespace SQLImport
{
    class SQLImport
    {
        // Replace these with the IP and the port that you want to run this server app on
        const string IPTOLOGIN = "155.98.81.77";
        const int PORTTOLISTENON = 6000;

        static void Main(string[] args)
        {
            while (true)
            {
                // Information to open up the xml file
                // String FileName = "junk.xml";
                // FileStream fs = new FileStream(FileName, FileMode.Open);

                try
                {
                    IPAddress IPToListenOn = IPAddress.Parse(IPTOLOGIN);
                    /* Initializes the Listener and starts it listening*/
                    TcpListener WelcomeSocket = new TcpListener(IPToListenOn, PORTTOLISTENON);
                    WelcomeSocket.Start();

                    while (true)
                    {
                        // Wait for Web Navigator to connect to it
                        Console.WriteLine("Waiting for a GPS Tracker unit to connect.....");
                        Socket s = WelcomeSocket.AcceptSocket();
                        Console.WriteLine("Connection accepted from " + s.RemoteEndPoint);

                        // Make a thread for the new connection and start it
                        GPSTrackerUnit GPSUnitSimulator = new GPSTrackerUnit(s);
                        Thread DirectorThread = new Thread(new ThreadStart(GPSUnitSimulator.ProcessUnitConnection));
                        DirectorThread.Start();
                    }

                    // Stop the TCP Listener - it should never get down to here
                    WelcomeSocket.Stop();
                }
                catch (Exception e)
                {
                    Console.WriteLine("Error..... " + e.StackTrace);
                }
            }
        }
    }
}

public class GPSTrackerUnit
{
    Socket SocketWithTracker = null;
    // constructor
public GPSTrackerUnit(Socket s)
{
    SocketWithTracker = s;
}

// This function is what processes every connection from a tracking unit to put it into the SQL Database
public void ProcessUnitConnection()
{
    while (true)
    {
        // Read the file from the network into a string
        List<XMLMessage> XMLMessagesReceived = ReceiveXMLMessages(1, true);

        // Write it to a file and then open it like before
        TextWriter tw = new StreamWriter("fromUnit.txt");

        // Write a line of text to the file
        tw.WriteLine(XMLMessagesReceived[0].ToString());

        // close the stream
        tw.Close();

        // Open the file
        FileStream fs = new FileStream("fromUnit.txt", FileMode.Open);
        XmlTextReader SQLImportReader = new XmlTextReader(fs);

        // Information needed to insert the entry into the table
        String ConnectionString = "Data Source=(local);Initial Catalog=GPSData;uid=sa;pwd=mmw&rlw565";
        SqlConnection DatabaseConnection = new SqlConnection(ConnectionString);
        DatabaseConnection.Open();

        // Read the root node
        SQLImportReader.Read();

        // Going through all the rest of the XML nodes
        while (SQLImportReader.Read() && !(SQLImportReader.Name.Equals("ROOT") && SQLImportReader.NodeType.ToString().Equals("EndElement")))
        {
            if (SQLImportReader.Name == "Point")
            {
                // Variables for the SQL Import
                double Latitude = 0;
                double Longitude = 0;
                double speed = 0;
                double altitude = 0;
                String MAC = "";

                while (SQLImportReader.MoveToNextAttribute())
                {
                    Console.WriteLine(SQLImportReader.Name + " " + SQLImportReader.Value);
                    switch (SQLImportReader.Name)
                    {
                        case "Latitude":
                            Latitude = Double.Parse(SQLImportReader.Value.ToString());
                            break;
                        case "Longitude":
                            Longitude = Double.Parse(SQLImportReader.Value.ToString());
                            break;
                        case "Speed":
                            if (SQLImportReader.Value.ToString() != "")
                            {
                                speed = Double.Parse(SQLImportReader.Value.ToString());
                            }
                            break;
                        case "Altitude":
                            altitude = Double.Parse(SQLImportReader.Value.ToString());
                            break;
                        default:
                            break;
                    }
                }
            }
        }
    }
}
case "Altitude":
    if (SQLImportReader.Value.ToString() != "")
        altitude = Double.Parse(SQLImportReader.Value.ToString());
    else
        altitude = 0;
    break;
    
    case "MAC":
        MAC = SQLImportReader.Value.ToString();
        break;
    default:
    Console.WriteLine("Error in Switch");
    break;
}
//End of the ProcessUnit Connection Method

/*-------------------------------------------------------------------------------
// The function will receive an arbitrary number of soap messages from web navigator
// Parameters: numToReceive is the number of soap messages that you want to receive
// shouldPrint enables or disables debug printing
// Returns a List<string> which contains the soap messages read from Web Navigator
-------------------------------------------------------------------------------*/
List<XMLMessage> ReceiveXMLMessages(int numToReceive, bool shouldPrint)
{
    List<XMLMessage> XMLMessages = new List<XMLMessage>(); //List of all the soap messages you received
    int NumXMLMessagesReceived = 0; //Number soap messages received
    String XMLMessage = ""; //Current soap message
    //Loop getting Soap Messages until received number you want
    while (NumXMLMessagesReceived < numToReceive)
//Receive byte by byte until the string ends in </c4Soap>
byte[] RecvBuffer = new byte[1];
int bytes = 1; // SocketWithTracker.Receive(RecvBuffer, RecvBuffer.Length, 0);

while (bytes > 0 && !XMLMessage.Contains("</ROOT>"))
{
    bytes = SocketWithTracker.Receive(RecvBuffer, RecvBuffer.Length, 0);

    //Make sure that it is < to begin with
    if (XMLMessage == "")
    {
        if ((char)RecvBuffer[0] == '<')
        {
            XMLMessage += (char)RecvBuffer[0];
        }
    }
    else
    {
        XMLMessage += (char)RecvBuffer[0];
    }
}

//Optional print out of the unaltered soap message received
if (XMLMessage != null && !XMLMessage.Equals(""))
{
    if (shouldPrint)
    {
        Console.WriteLine('
' + XMLMessage);
    }

    //Create the Soap Message here
    XMLMessage CurrentMessage = new XMLMessage(XMLMessage);

    //Add the soap message to the list and then clear it
    XMLMessages.Add(CurrentMessage);
    NumXMLMessagesReceived++;
    //XMLMessage = "";
}

//Prepare for next loop
Thread.Sleep(50);
XMLMessage = "";

//Return the list
return XMLMessages;

}//end of ReceiveXMLMessage Class

}//End of the GPSTrackerUnit Connection

public class XMLMessage
{
    String XMLString = null;

    public XMLMessage(String XMLString)
    {
        this.XMLString = XMLString;
    }

    public override string ToString()
    {
        return XMLString;
    }
}

}//end of the XMLMessage Class
} // end of the namespace
<configuration>
  <appSettings>
    <add key="RefreshTime" value="4000" />
    <add key="ConnectionString" value="Data Source=(local);Initial Catalog=GPSData;uid=sa;pwd=XXXXXXXXX" />
    <add key="FilePathRoot" value="c:/inetpub/wwwroot/data/" />
  </appSettings>
</configuration>
using System;
using System.Collections.Generic;
using System.Text;
using System.Net.Sockets;
using System.IO;
using System.Data.SqlClient;

namespace RouteCreator
{
    class Program
    {
        public static NMEAMessage CurrentNMEAMessage = new NMEAMessage();

        static void Main(string[] args)
        {
            //MAC
            Console.WriteLine("MAC? ");
            String MAC = Console.ReadLine();

            //Route Name
            Console.WriteLine("Route Name? ");
            String RouteName = Console.ReadLine();

            //Route Number
            Console.WriteLine("Route Number? ");
            String RouteNumber = Console.ReadLine();

            //Driver
            Console.WriteLine("Driver? ");
            String Driver = Console.ReadLine();

            //On Time
            Console.WriteLine("On Time? ");
            int OnTime = Int32.Parse(Console.ReadLine());

            //Last Stop
            Console.WriteLine("Last Stop? ");
            String LastStop = Console.ReadLine();

            //Next Stop
            Console.WriteLine("Next Stop? ");
            String NextStop = Console.ReadLine();

            //IP of Gumstix
            Console.WriteLine("IP of Gumstix: ");

            //Sampling Interval in ms
            Console.WriteLine("Sampling Interval (in ms): ");

            //Open a connection to the database
            String ConnectionString = "Data Source=(local);Initial Catalog=GPSData;uid=sa;pwd=XXXXXXX";

            SqlConnection DatabaseConnection = new SqlConnection(ConnectionString);
            DatabaseConnection.Open();

            //Create the table for the route
            String CreateTableString = "CREATE TABLE [dbo].[" + MAC + "]([Latitude] [float] NULL, [Longitude] [float] NULL, [Speed] [float] NULL, [Altitude] [float] NULL, [MAC] [varchar](50) COLLATE SQL_Latin1_General_CI_AS NULL) ON [PRIMARY]";

            SqlCommand CreateTableCommand = new SqlCommand(CreateTableString, DatabaseConnection);
        }
    }
}
CreateTableCommand.ExecuteNonQuery();

//Put it on the list of routes to simulate
String AddToSimulateListString = "INSERT INTO RoutesToSimulate (MAC, routeName, routeNumber, Driver, OnTime, LastStop, NextStop) VALUES (" + MAC + ", " + RouteName + ", " + RouteNumber + ", " + Driver + ", " + OnTime + ", " + LastStop + ", " + NextStop + ");
SqlCommand AddToSimulateListCommand = new SqlCommand(AddToSimulateListString, DatabaseConnection);
AddToSimulateListCommand.ExecuteNonQuery();

Serial GPSSerial = new Serial(MAC, DatabaseConnection);

while (true)
{
    System.Threading.Thread.Sleep(50);
}

//Close the database connection
DatabaseConnection.Close();
using System;
using System.Collections.Generic;
using System.Text;
using System.Data.SqlClient;

namespace RouteCreator
{
    class Serial
    {
        System.IO.Ports.SerialPort comPort; //System.IO.Ports.SerialPort comPort = null;
        NMEAMessage CurrentNMEAMessage = new NMEAMessage();
        String MAC = "";
        SqlConnection DatabaseConnection = null;
        public Serial (String NewMAC, SqlConnection NewDatabaseConnection)
        {
            comPort = new System.IO.Ports.SerialPort("COM3");
            MAC = NewMAC;
            DatabaseConnection = NewDatabaseConnection;
            try
            {
                comPort.Open();
                comPort.DataReceived += new System.IO.Ports.SerialDataReceivedEventHandler(comPort_DataReceived);
            }
            catch (Exception e)
            {
                String error = "Com port could not be found";
            }
        }
        void comPort_DataReceived(object sender, System.IO.Ports.SerialDataReceivedEventArgs e)
        {
            GPSInfo CurrentRoutePoint = ParseSerialGPS(CurrentNMEAMessage);
            //If there has been enough information received from the GPS update the window
            if (CurrentRoutePoint != null)
            {
                //Insert the point into the table
                String CommandString2 = "INSERT INTO " + MAC + " (Latitude, Longitude, speed, altitude, MAC) VALUES (" + CurrentRoutePoint.Latitude + "," + CurrentRoutePoint.Longitude + "," + CurrentRoutePoint.Speed + "," + CurrentRoutePoint.Altitude + "," + CurrentRoutePoint.MAC + ");";
                SqlCommand InsertReadingCommand = new SqlCommand(CommandString2, DatabaseConnection);
                int NumOfLinesModified = InsertReadingCommand.ExecuteNonQuery();
                //I think this is where I need to do all the work
            }
        }
        public GPSInfo ParseSerialGPS(NMEAMessage NewNMEAMessage)
        {
            //TODO: I don't think that I should be making this.
            NMEAMessage myNMEAMessage = NewNMEAMessage;
            String line = comPort.ReadLine();
            if (line == null)
                return null;
            else
            {
                //I think this is where I need to do all the work
            }
        }
    }
}
//Message is now complete and I need to do my processing on it
if (myNMEAMessage.AddLine(line))
{
    #region ...GPS Variables
    double Latitude = 0;
    double Longitude = 0;
    double Speed = 0;
    double Altitude = 0;

    int NumOfSatellites = 0;
    List<SatelliteInfo> CurSatelliteCoverage = null;
    double PDOP = -1;
    double HDOP = -1;
    double VDOP = -1;
    int NumSatsUsedTracking = 0;
    int FixMode = 0;
    #endregion

    //Get the variables like latitude, longitude, speed, altitude, number of satellites, etc.
    Latitude = myNMEAMessage.GetLatitude();
    Longitude = myNMEAMessage.GetLongitude();
    Speed = myNMEAMessage.GetSpeed();
    Altitude = myNMEAMessage.GetAltitude();
    NumOfSatellites = myNMEAMessage.GetNumSatellites();
    NumSatsUsedTracking = myNMEAMessage.GetNumSatsUsedTracking();
    CurSatelliteCoverage = myNMEAMessage.GetSatelliteInfo();
    PDOP = myNMEAMessage.GetPDOP();
    HDOP = myNMEAMessage.GetHDOP();
    VDOP = myNMEAMessage.GetVDOP();
    FixMode = myNMEAMessage.FixMode();
    
    GPSInfo CurrentGPSInfo = new GPSInfo(Latitude, Longitude, Speed, Altitude, "00:00:00:00:00:00", System.DateTime.Now, NumOfSatellites, NumSatsUsedTracking, CurSatelliteCoverage, PDOP, HDOP, VDOP, FixMode);
    return CurrentGPSInfo;
}

myNMEAMessage = new NMEAMessage();

return null;
using System;
using System.Collections.Generic;
using System.Text;
using System.Data.SqlClient;

namespace RouteSimulator
{
    class Program
    {
        public static List<List<GPSReading>> DataBank = new List<List<GPSReading>>();

        static void Main(string[] args)
        {
            // Initialize the SQL table with these routes so the website will display them
            InsertRoutesToSimulate();

            // Fill in the datastructure with information stored in the SQL tables
            FillDataBank();

            // Make index variables into each of the routes current point and initialize
            int[] PointIndex = new int[DataBank.Count];
            for (int i = 0; i < PointIndex.Length; i++)
            {
                PointIndex[i] = 0;
            }

            // Open the database
            String ConnectionString = "Data Source=(local);Initial Catalog=GPSData;uid=sa;pwd=XXXXXXXXXX";
            SqlConnection DatabaseConnection = new SqlConnection(ConnectionString);
            DatabaseConnection.Open();

            // Continually simulate the routes
            while (true)
            {
                int RouteIndex = 0;
                foreach (List<GPSReading> CurrentRoute in DataBank)
                {
                    // Put a reading into the GPSPoints database
                    GPSReading ReadingToInsert = CurrentRoute[PointIndex[RouteIndex]];

                    // Insert a point
                    String CommandString2 = "INSERT INTO GPSPoints (Latitude, Longitude, Speed, Altitude, MAC) VALUES (" + ReadingToInsert.Latitude + ", " + ReadingToInsert.Longitude + ", " + ReadingToInsert.Speed + ", " + ReadingToInsert.Altitude + ", " + ReadingToInsert.MAC + ");";
                    SqlCommand InsertReadingCommand = new SqlCommand(CommandString2, DatabaseConnection);
                    int NumOfLinesModified = InsertReadingCommand.ExecuteNonQuery();
                    Console.WriteLine("Adding another point");

                    // If route has been cycled through then delete its history from the database
                    if (PointIndex[RouteIndex] >= CurrentRoute.Count)
                    {
                        String DeleteString = "DELETE FROM GPSPoints WHERE MAC = " + ReadingToInsert.MAC + ");";
                        SqlCommand DeleteRouteCommand = new SqlCommand(DeleteString, DatabaseConnection);
                        int NumOfLinesDeleted = DeleteRouteCommand.ExecuteNonQuery();
                        PointIndex[RouteIndex] = 0;
                    }
                }
                RouteIndex++;
            }
        }
    }
}
System.Threading.Thread.Sleep(2000);

DatabaseConnection.Close();

/****************************************************************************
* FillDataBank will fill the databank structure the all the GPS reading objects
* from each of the routes that we are trying to simulate.
* ************************************************************************/
public static void FillDataBank()
{
    // Get the list of all the MACs
    List<String> MACList = RouteMACsToSimulate();

    // Open the database
    String ConnectionString = "Data Source=(local);Initial Catalog=GPSData;uid=sa;pwd=XXXXXXXXX";
    SqlConnection DatabaseConnection = new SqlConnection(ConnectionString);
    DatabaseConnection.Open();

    foreach (String CurrentMAC in MACList)
    {
        DataBank.Add(new List<GPSReading>());
        String RouteString = "SELECT Latitude, Longitude, speed, altitude FROM [" + CurrentMAC + "]";
        SqlCommand ObtainRouteCommand = new SqlCommand(RouteString, DatabaseConnection);
        SqlDataReader RouteReader = ObtainRouteCommand.ExecuteReader();
        while (RouteReader.Read())
        {
            // Make a GPSReading object
            GPSReading RouteGPSReading = new GPSReading(Double.Parse(RouteReader["Latitude"].ToString()), Double.Parse(RouteReader["Longitude"].ToString()), Double.Parse(RouteReader["speed"].ToString()), Double.Parse(RouteReader["altitude"].ToString()), CurrentMAC);
            DataBank[MACList.IndexOf(CurrentMAC)].Add(RouteGPSReading);
        }
        RouteReader.Close();
    }

    // Close the database
    DatabaseConnection.Close();
}

/****************************************************************************
* InsertRoutesToSimulate will obtain from the database what routes to simulate
* and will put the appropriate entries in the UnitOnRoute Database
* ************************************************************************/
public static void InsertRoutesToSimulate()
{
    // Open the database
    String ConnectionString = "Data Source=(local);Initial Catalog=GPSData;uid=sa;pwd=XXXXXXXXX";
    SqlConnection DatabaseConnection2 = new SqlConnection(ConnectionString);
    DatabaseConnection2.Open();

    // Actual Database modifications
    String MACString = "SELECT MAC, routeName, routeNumber, Driver, OnTime, LastStop, NextStop FROM RoutesToSimulate";
    SqlCommand ObtainMACCommand = new SqlCommand(MACString, DatabaseConnection2);
    SqlDataReader MACReader = ObtainMACCommand.ExecuteReader();

    List<RouteInfo> RouteInfoList = new List<RouteInfo>();
//Get the data to put into the table
while (MACReader.Read())
{
    //Make the RouteInfo object
    RouteInfo CurrentRouteInfo = new RouteInfo(MACReader["MAC"].ToString(),
    MACReader["routeName"].ToString(), MACReader["routeNumber"].ToString(), MACReader["Driver"].ToString(),
    Int32.Parse(MACReader["OnTime"].ToString()), MACReader["LastStop"].ToString(),
    MACReader["NextStop"].ToString());
    RouteInfoList.Add(CurrentRouteInfo);
}
MACReader.Close();
foreach (RouteInfo CurrentRoute in RouteInfoList)
{
    if (!ContainsMAC(CurrentRoute.MAC))
    {
        String CommandString2 = "INSERT INTO UnitOnRoute (MAC, routeName, routeNumber, Driver, OnTime, LastStop, NextStop) VALUES ('' + CurrentRoute.MAC + '','' + CurrentRoute.RouteName + ','' + CurrentRoute.RouteNumber + ','' + CurrentRoute.Driver + ','' + CurrentRoute.OnTime + ','' + CurrentRoute.LastStop + ','' + CurrentRoute.NextStop + '');
        SqlCommand InsertReadingCommand = new SqlCommand(CommandString2,
        DatabaseConnection2);
        int NumOfLinesModified = InsertReadingCommand.ExecuteNonQuery();
    }
}
//Close the database
DatabaseConnection2.Close();
return;
}

/***************************************************************************/
* RouteMACsToSimulate will return a list of the MACs of all the routes that* 
* need to be simulated* */
public static List<String> RouteMACsToSimulate()
{
    //Open the database
    String ConnectionString = "Data Source=localhost;Initial Catalog=GPSData;uid=sa;pwd=XXXXXXXXX";
    SqlConnection DatabaseConnection = new SqlConnection(ConnectionString);
    DatabaseConnection.Open();
    //Actual Database modifications
    List<String> MACList = new List<String>();
    String MACString = "SELECT MAC FROM RoutesToSimulate";
    SqlCommand ObtainMACSCommand = new SqlCommand(MACString, DatabaseConnection);
    SqlDataReader MACReader = ObtainMACSCommand.ExecuteReader();
    while (MACReader.Read())
    {
        //Put the MACS into a list
        MACList.Add(MACReader["MAC"].ToString());
    }
    MACReader.Close();
    //Close the database
    DatabaseConnection.Close();
    return MACList;
}

/***************************************************************************/
* Contains MAC will check the UnitOnRoute database to see* 
* if there is already a route registered under that MAC. If
* there is it will go ahead and put its point in the route that
* is already registered
************************************************************/

```csharp
static bool ContainsMAC(string MAC)
{
    // Open the database
    string connectionString = "Data Source=(local);Initial Catalog=GPSData;uid=sa;pwd=XXXXXXXXXX";
    SqlConnection databaseConnection = new SqlConnection(connectionString);
    databaseConnection.Open();

    // Actual Database modifications
    List<string> macList = new List<string>();
    string macString = "SELECT MAC FROM UnitOnRoute";
    SqlCommand obtainMacCommand = new SqlCommand(macString, databaseConnection);
    SqlDataReader macReader2 = obtainMacCommand.ExecuteReader();
    while (macReader2.Read())
    {
        // Put the MACs into a list
        macList.Add(macReader2["MAC"].ToString());
    }
    macReader2.Close();

    // Close the database
    databaseConnection.Close();

    if (macList.Contains(MAC))
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

// Class representing the data received each time the GPS makes a position reading

```csharp
public class GPSReading
{
    public double Latitude;
    public double Longitude;
    public double Speed;
    public double Altitude;
    public string MAC;
    public GPSReading(double NewLatitude, double NewLongitude, double NewSpeed, double NewAltitude, string NewMAC)
    {
        Latitude = NewLatitude;
        Longitude = NewLongitude;
        Speed = NewSpeed;
        Altitude = NewAltitude;
        MAC = NewMAC;
    }
}
```

// Class representing the information needed about a route

```csharp
public class RouteInfo
{
    public string MAC;
    public string RouteName;
    public string RouteNumber;
    public string Driver;
    public int OnTime;
    public string LastStop;
    public string NextStop;
    public RouteInfo(string NewMac, string NewRouteName, string NewRouteNumber, string
```
NewDriver, int NewOnTime, String NewLastStop, String NewNextStop)
{
    MAC = NewMac;
    RouteName = NewRouteName;
    RouteNumber = NewRouteNumber;
    Driver = NewDriver;
    OnTime = NewOnTime;
    LastStop = NewLastStop;
    NextStop = NewNextStop;
}
}
using System;
using System.Collections.Generic;
using System.Text;
using System.Net;
using System.Net.Sockets;
using System.IO;

namespace MapPointSerial
{
    class Program
    {
        static void Main(string[] args)
        {
            // Create the COM Port to send the data out of which will be looped back to any program on COM
            System.IO.Ports.SerialPort comPort7 = new System.IO.Ports.SerialPort("COM7");
            comPort7.Open();
            comPort7.BaudRate = 4800;

            // Read in the IP of the Gumstix
            Console.Write("Please input the IP of the gumstix: ");
            String GumstixIP = Console.ReadLine();

            // Open up the Network interface to get the data from
            String IPOfGumstix = GumstixIP;
            TcpClient tcpclnt = new TcpClient();
            int value = tcpclnt.SendTimeout;
            int val2 = tcpclnt.ReceiveTimeout;
            StreamWriter SW = null;
            StreamReader SR = null;

            try
            {
                tcpclnt.Connect(IPOfGumstix, 2947);
                SW = new StreamWriter(tcpclnt.GetStream());
                SR = new StreamReader(tcpclnt.GetStream());
                SW.WriteLine("r");
                SW.Flush();
            }
            catch (Exception e)
            {
                Console.WriteLine("Could not connect to: " + IPOfGumstix);
                Console.WriteLine("Please hit any key to acknowledge error and end program");
                Console.ReadLine();
                return;
            }

            while (true)
            {
                try
                {
                    String NMEALine = SR.ReadLine();
                    if (!NMEALine.Contains("GP2DA"))
                    {
                        comPort7.Write(NMEALine + \r\n);
                        Console.WriteLine("From gumstix to comport" + NMEALine);
                    }
                }
                catch (Exception e)
                {
                    Console.WriteLine("There was an exception");
                    Console.WriteLine(e.Message);
                }
            }
        }
    }
}
<@ Master Language="C#" AutoEventWireup="true" CodeFile="Admin.master.cs" Inherits="Admin_Admin" />

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="content-type" content="text/html; charset=utf-8" />
<style type="text/css">

  A {text-decoration:none}

  -->
</style>
<title>UShuttle Admin</title>
</head>

<body style="background-color: #ffeec" onload="onLoad(40.771, -111.845)">
<form id="form1" runat="server">
<table style="width: 100%; height: 100%;">
<tr style="width: 100%; height: 10px;">
<td style="width: 100%; height: 10px;">
<table style="width: 100%; background-color: #ffdd88;" border="0">
cellpadding="0" cellspacing="0">
<tr>
<td style="width: 50%; height: 1px; font-weight: bold;">
  &nbsp; &nbsp; | &nbsp; &nbsp;
  &nbsp; &nbsp; | &nbsp; &nbsp;
  &nbsp; &nbsp; | &nbsp; &nbsp;
  &nbsp; &nbsp; | &nbsp; &nbsp;
  <asp:HyperLink ID="SendEmail" runat="server" ForeColor="#C00000" Font-Bold="True" Font-Names="Arial" NavigateUrl="/Admin/SendEmail.aspx">Send Email</asp:HyperLink>
  &nbsp; &nbsp; | &nbsp; &nbsp;
  <asp:HyperLink ID="InputRoute" runat="server" ForeColor="#C00000" Font-Bold="True" Font-Names="Arial" NavigateUrl="/Admin/InputRoute.aspx">Input Route</asp:HyperLink>
  &nbsp; &nbsp; | &nbsp; &nbsp;
</td>
</tr>
</table>
</form>
</body>
</html>
<tr>
<td style="width: 100%; height: 100%">
<asp:contentplaceholder id="MainBodyPlaceHolder3" runat="server">
</asp:contentplaceholder>
</td>
</tr>
<tr>
<td style="width: 100%; height: 15px; background-color: #ffdd88;">
<table style="width: 100%">
<tr>
<td style="width: 50%; text-align: left; height: 19px;">
<td style="width: 50%; text-align: right; height: 19px;">
<asp:HyperLink ID="HyperLink1" runat="server" BorderStyle="None" ForeColor="#C00000" NavigateUrl="help.aspx" Font-Bold="True" Font-Names="Arial" Font-Size="Smaller" Width="274px">Help</asp:HyperLink></td>
</tr>
</table>
</td>
</tr>
</table>
</form>
</body>
</html>
This is the help page. It is still under construction
From here you will hopefully be able to upload the data file from the Route Recorder to add a route.
From this page you will hopefully be able to schedule when drivers are working and what routes they are on so you don’t have to change it every time.
From here you will hopefully be able to send an email to all the users that have registered to notify them of software updates or route changes.
```xml
<%@ Page Language="C#" MasterPageFile="/Admin/Admin.master" AutoEventWireup="true" %>
<asp:Content ContentID="Content1" ContentPlaceHolderID="MainBodyPlaceHolder3" Runat="Server">
  <asp:GridView ID="GridView1" runat="server" AutoGenerateColumns="False"
               CellPadding="4"
               DataKeyNames="UserName" ForeColor="#333333" GridLines="None"
               OnSelectedIndexChanged="GridView1_SelectedIndexChanged">
    <FooterStyle BackColor="#507CD1" Font-Bold="True" ForeColor="White" />
    <Columns>
      <asp:BoundField DataField="UserName" HeaderText="User Name" />
      <asp:BoundField DataField="Email" HeaderText="Email" />
      <asp:CommandField ShowSelectButton="True" />
    </Columns>
    <RowStyle BackColor="#EFF3FB" />
    <EditRowStyle BackColor="#2461BF" />
    <SelectedRowStyle BackColor="#D1DF1" Font-Bold="True" ForeColor="#333333" />
    <PagerStyle BackColor="#2461BF" ForeColor="White" HorizontalAlign="Center" />
    <HeaderStyle BackColor="#507CD1" Font-Bold="True" ForeColor="White" />
    <AlternatingRowStyle BackColor="White" />
  </asp:GridView>
<br />
  <asp:DetailsView ID="DetailsView1" runat="server" CellPadding="4" ForeColor="#333333"
                 GridLines="None" Height="50px" Width="125px" >
    <FooterStyle BackColor="#507CD1" Font-Bold="True" ForeColor="White" />
    <CommandRowStyle BackColor="#D1DF1" Font-Bold="True" />
    <EditRowStyle BackColor="#2461BF" />
    <RowStyle BackColor="#EFF3FB" />
    <PagerStyle BackColor="#2461BF" ForeColor="White" HorizontalAlign="Center" />
    <FieldHeaderStyle BackColor="#DEEBF5" Font-Bold="True" />
    <HeaderStyle BackColor="#507CD1" Font-Bold="True" ForeColor="White" />
    <AlternatingRowStyle BackColor="White" />
  </asp:DetailsView>
</asp:Content>
```
<?xml version="1.0"?>
<!--
  Note: As an alternative to hand editing this file you can use the
  web admin tool to configure settings for your application. Use
  the Website->Asp.Net Configuration option in Visual Studio.
  A full list of settings and comments can be found in
  machine.config.comments usually located in
  \Windows\Microsoft.Net\Framework\v2.x\Config
-->
<configuration xmlns="http://schemas.microsoft.com/.NetConfiguration/v2.0">
  <appSettings/>
  <connectionStrings/>
  <system.web>
    <!-- Letting only administrators in -->
    <authorization>
      <allow roles="admin" />
      <deny users="*" />  
    </authorization>
  </system.web>
</configuration>
This is the alarm page. It is still under construction.
//TODO: Global Data - document why I need each of these
var map;
var gmarkers = [];
var htmls = [];
var to_htmls = [];
var from_htmls = [];
var i=0;

function onLoad(lastLat, lastLng) {
    if (GBrowserIsCompatible()) {
        //Blank icon
        var icon = new GIcon();
        icon.image = "images/blank.png";
        icon.shadow = "images/blank.png";
        icon.iconSize = new GSize(1, 1);
        icon.shadowSize = new GSize(1, 1);
        icon.iconAnchor = new GPoint(1, 1);
        icon.infoWindowAnchor = new GPoint(1, 1);

        //Icon that has the shuttle picture
        var icon2 = new GIcon();
        icon2.image = "images/shuttle.png";
        icon2.shadow = "images/blank.png";
        icon2.iconSize = new GSize(40, 20);
        icon2.shadowSize = new GSize(1, 1);
        icon2.iconAnchor = new GPoint(1, 1);
        icon2.infoWindowAnchor = new GPoint(1, 1);

        //Making the map and setting its properties
        map = new GMap2(document.getElementById("map"));
        map.setCenter(new GLatLng(lastLat, lastLng), 14);
        map.addControl(new GSmallMapControl());
        map.setMapType(G_HYBRID_TYPE);
        map.addOverlay(new GMarker(map.getCenter(), icon2));

        //Listener that when you move the map it displays the new center coordinates
        GEvent.addListener(map, 'move', function() {
            map.clearOverlays();
            var center = map.getCenter();
            var latLngStr = '(' + center.lat() + ', ' + center.lng() + ')';
            document.getElementById("message").innerHTML = latLngStr;
            //This will put a marker at the center point
            var point = new GLatLng(center.lat(), center.lng());
            map.addOverlay(new GMarker(point));
        });
    }
}
<script>
</script>

</head>
<body onLoad="onLoad(40.7618209, -111.835069)"
<form id="form2" runat="server">
<table>
<tr>
<td style="height: 375px; width: 700px;" align="center" valign="middle">
<div id="map" style="width: 640px; height: 375px"></div>
</td>
<td id="message"></td>
</tr>
</table>
</form>
</body>
</html>

<!DOCTYPE html PUBLIC "-/W3C/DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
<title>Create a new user</title>
</head>
<body>
<form id="form1" runat="server">
<div>
    <asp:CreateUserWizard ID="CreateUserWizard1" runat="server" BackColor="#FFFBDD" BorderColor="#FFFBAD" BorderStyle="Solid" BorderWidth="1px" FinishDestinationPageUrl="/Default.aspx"
        Font-Names="Verdana" Font-Size="0.8em" MembershipProvider="" MySqlMembershipProvider="" FinishCompleteButtonImageURL="/~/images/Ulogo.gif"
        DisplayName="" DisplayCancelButton="True" DisplaySideBar="" ActiveStepIndex="1" LoginCreatedUser="False">
        <WizardSteps>
            <asp: CreateUserWizardStep runat="server">
                <table border="0" style="font-size: 100%; font-family: Verdana; background-color: #fffbd6">
                    <tr align="center" colspan="2" style="font-weight: bold;">
                        <td align="center" colspan="2" style="font-size: 100%; font-family: Verdana; background-color: #990000">Your New Account</td>
                    </tr>
                    <tr align="right">
                        <asp:RequiredFieldValidator ID="UserNameRequired" runat="server" ControlToValidate="UserName" ErrorMessage="User Name is required." ValidationGroup="CreateUserWizard1" ToolTip="" />
                    </tr>
                    <tr align="right">
                        <asp:RequiredFieldValidator ID="PasswordRequired" runat="server" ControlToValidate="Password" ErrorMessage="Password is required." ValidationGroup="CreateUserWizard1" ToolTip="" />
                    </tr>
                    <tr align="right">
                        <asp:RequiredFieldValidator ID="ConfirmPasswordRequired" runat="server" ControlToValidate="" ErrorMessage="" ToolTip="" />
                    </tr>
                    <tr align="right">
                        <asp:RequiredFieldValidator ID="UserName" runat="server" ControlToValidate="UserName"ErrorMessage="User Name is required." ValidationGroup="CreateUserWizard1" ToolTip="" />
                    </tr>
                    <tr align="right">
                        <asp:RequiredFieldValidator ID="Password" runat="server" ControlToValidate="Password" ErrorMessage="Password is required." ValidationGroup="CreateUserWizard1" ToolTip="" />
                    </tr>
                    <tr align="right">
                        <asp:RequiredFieldValidator ID="ConfirmPassword" runat="server" ControlToValidate="" ErrorMessage="" ToolTip="" />
                    </tr>
                </table>
            </asp:CreateUserWizardStep>
        </WizardSteps>
    </asp:CreateUserWizard>
</div>
</form>
</body>
</html>
<asp:TextBox ID="ConfirmPassword" runat="server" TextMode="Password"></asp:TextBox>
<asp:RequiredFieldValidator ID="" ControlToValidate="ConfirmPassword"
ErrorMessage="Confirm Password is required." ToolTip="Confirm Password is required." ValidationGroup="CreateUserWizard1"></asp:RequiredFieldValidator>

<asp:TextBox ID="Email" runat="server" AssociatedControlID="Email">E-mail:</asp:TextBox>
<asp:RequiredFieldValidator ID="EmailRequired" runat="server" ControlToValidate="Email" ErrorMessage="E-mail is required." ToolTip="E-mail is required." ValidationGroup="CreateUserWizard1"></asp:RequiredFieldValidator>

<asp:TextBox ID="Question" runat="server" AssociatedControlID="Question">Security Question:</asp:TextBox>
<asp:RequiredFieldValidator ID="QuestionRequired" runat="server" ControlToValidate="Question" ErrorMessage="Security question is required." ToolTip="Security question is required." ValidationGroup="CreateUserWizard1"></asp:RequiredFieldValidator>

<asp:TextBox ID="Answer" runat="server" AssociatedControlID="Answer">Security Answer:</asp:TextBox>


<asp:Literal ID="ErrorMessage" runat="server" EnableViewState="False"></asp:Literal>
<table border="0" cellspacing="5" style="width: 100%; height: 100%">
  
  <tr align="right">
    <td align="right" colspan="0">
      <asp:Button ID="StepNextButton" runat="server" BackColor="#CC9966" BorderStyle="Solid" BorderWidth="1px" CommandName="MoveNext" Font-Names="Verdana" ForeColor="#990000" Text="Create User" ValidationGroup="CreateUserWizard1" />
    </td>
  </tr>
</table>
<asp: CreateUserWizard ID="CreateUserWizard1" runat="server" ValidationGroup="CreateUserWizard1" CommandName="Cancel" Font-Faces="Verdana" ForeColor="#990000" Text="Cancel" ValidationGroup="CreateUserWizard1" />

</FinishNavigationTemplate>

<StepNavigationTemplate>
  <asp:Button ID="StepPreviousButton" runat="server" BackColor="White" BorderColor="#CC9966" BorderStyle="Solid" BorderWidth="1px" CausesValidation="False" CommandName="MovePrevious" Font-Faces="Verdana" ForeColor="#990000" Text="Previous" />
  <asp:Button ID="StepNextButton" runat="server" BackColor="White" BorderColor="#CC9966" BorderStyle="Solid" BorderWidth="1px" CommandName="MoveNext" Font-Faces="Verdana" ForeColor="#990000" Text="Next" />
</StepNavigationTemplate>

<StartNavigationTemplate>
  <asp:Button ID="StartNextButton" runat="server" BackColor="White" BorderColor="#CC9966" BorderStyle="Solid" BorderWidth="1px" CommandName="MoveNext" Font-Faces="Verdana" ForeColor="#990000" Text="Next" />
  <asp:Button ID="CancelButton" runat="server" BackColor="White" BorderColor="#CC9966" BorderStyle="Solid" BorderWidth="1px" CausesValidation="False" CommandName="Cancel" Font-Faces="Verdana" ForeColor="#990000" Text="Cancel" ValidationGroup="CreateUserWizard1" />
</StartNavigationTemplate>

</asp:CreateUserWizard>

</div>

</form>

</body>

</html>
public partial class Practice_Master : System.Web.UI.Page
{
    #region Global Variables
    Routes ShuttleRoutes;
    #endregion

    //***************************************************************************************
    // Preforms the initialization necessary for the page and retrieves the info
    // from the database
    //***************************************************************************************

    protected void Page_Load(object sender, EventArgs e)
    {
        //Gets all the latest shuttle route information from the database
        ShuttleRoutes = new Routes();
    }

    //***************************************************************************************
    * Generates the three JavaScript functions
    * ToggleHidden will toggle if the route numbers are displayed or not on the right hand side?
    * ToggleHiddenInfo will toggle if current route info is shown on the left hand side
    * ChangeArrow will toggle an image showing an arrow pointing up or down
    //***************************************************************************************

    protected string GenerateJavaScript()
    {
        string JavaScript = "";
        JavaScript += \n\nfunction ToggleHidden(r)"
        JavaScript += \n{";
        JavaScript += \nif(document.getElementById(r).style.display == '''");
        JavaScript += \n";
        JavaScript += \ndocument.getElementById(r).style.display = 'none';";
        JavaScript += \n";
        JavaScript += \nelse"
        JavaScript += \n{";
        JavaScript += \ndocument.getElementById(r).style.display = '';
        JavaScript += \n";
        JavaScript += \nfunction ToggleHiddenInfo(RouteNumber)"
        JavaScript += \n{"
        JavaScript += \nif(document.getElementById("Route" + RouteNumber).style.display == '''");
        JavaScript += \n";
        JavaScript += \ndocument.getElementById("Route" + RouteNumber).style.display = 'none';";
        JavaScript += \n";
        JavaScript += \nelse"
        JavaScript += \n{";
        JavaScript += \n";
protected IEnumerator ToggleRouteFunctions(RouteNumber)
{
    // This function will toggle an arrow when it is given the image name
    var ToggleRoute = function ToggleArrow(img_name)
    {
        var dnarrow = document[img_name].src.match('dnarrow');
        if (dnarrow)
        {
            document[img_name].src = './images/uparrow9x5.png';
        }
        else
        {
            document[img_name].src = './images/dnarrow9x5.png';
        }
    }

    // This function will toggle an arrow when it is given the image name
    var SetCenterMAC = function SetCenterMAC(MAC)
    {
        var centerMAC = MAC;
        ChangeCenter();
    }

    return ToggleRoute;
}

// This function will generate the HTML needed to display the right hand column
// where you can select routes
// ******************************************************************************
protected string GenerateRoutesToShow()
{
    // TODO: Try and execute JavaScript here in C# that will show all the routes when I
    click on the color of the route
    // Output the start of the table that allows you to select routes
    RoutesToShow = "<table style="width: 100%\" id="Table1\"">
    // Drop down list to choose who to follow
    RoutesToShow += "\n<select align="center" id="FollowBox" onChange="SetCenterMAC(this.value)">
    " + ShuttleRoutes.RouteNames();
    foreach (string RouteName in ShuttleRoutes.RouteNumbers(RouteName))
    {
        RoutesToShow += "\n<option value="\"" + ShuttleRoutes.GetSingleInfo
    
}
(RouteName, RouteNumber, "MAC").Replace(":\", "]\") + "]" + RouteName + "] - " + RouteNumber + "]</option>");
}
RoutesToShow += "]</select>
;
RoutesToShow += "]<script type="text/javascript">
;
RoutesToShow += "]function ToggleState(RouteDisplayChange){
;
RoutesToShow += "]\t if(RouteDisplayChange);
;
RoutesToShow += "]\t\nreturn false"
;
RoutesToShow += "]\nalert("Equal\")
;
RoutesToShow += "]\t"
;
RoutesToShow += "]\nelse"
;
RoutesToShow += "]return true; RouteDisplayChange = true;
;
RoutesToShow += "]\nalert("Not Equal\")
;
RoutesToShow += "]\t"
;
RoutesToShow += "]"
;

foreach (string RouteName in ShuttleRoutes.RouteNames())
{
    foreach (string RouteNumber in ShuttleRoutes.RouteNumbers(RouteName))
    {
        RoutesToShow += "]\nvar Display" + ShuttleRoutes.GetSingleInfo(RouteName, RouteNumber, "]" + RouteName + "] - " + RouteNumber + "]<input id=""Checkbox" + CheckboxNumber + "" type="checkbox" onChange="if (""Display" + ShuttleRoutes.GetSingleInfo(RouteName, RouteNumber, "MAC").Replace(":\", "]") + "]" + "\n\nDisplay"
;
foreach (string RouteName in ShuttleRoutes.RouteNumbers())
{
    foreach (string RouteNumber in ShuttleRoutes.RouteNumbers(RouteName))
    {
        RoutesToShow += "]\n\nDisplay"
;
    }
}

//Append the code for the end of the table
RoutesToShow += "]</table>
;

return RoutesToShow;

/*----------------------------------------------------------------------------------------*/
/* GenerateCurrentRouteData() will form the HTML needed to show the user the current */
/* route data and update it on the left hand column of the display                        */
/*----------------------------------------------------------------------------------------*/
protected string GenerateCurrentRouteData()
{
    string RouteInfo = ""
;
    //String to return
    NameValueCollection appSettings = System.Configuration.ConfigurationSettings.*/
    AppSettings; //Access to the configuration file

    //If these settings are displayed by default
    Boolean LatitudeDefault = appSettings[0] == "yes" ? true : false;
```c
Boolean LongitudeDefault = appSettings[1] == "yes" ? true : false;
Boolean AltitudeDefault = appSettings[2] == "yes" ? true : false;
Boolean LastStopDefault = appSettings[6] == "yes" ? true : false;
Boolean NextStopDefault = appSettings[7] == "yes" ? true : false;

//If these settings are displayed when user clicks show all
Boolean LatitudeAllow = appSettings[8] == "yes" ? true : false;
Boolean LongitudeAllow = appSettings[9] == "yes" ? true : false;
Boolean SpeedAllow = appSettings[10] == "yes" ? true : false;
Boolean DriverAllow = appSettings[12] == "yes" ? true : false;
Boolean LastStopAllow = appSettings[14] == "yes" ? true : false;
Boolean NextStopAllow = appSettings[15] == "yes" ? true : false;

//Start the table
RouteInfo += "\n<table style="width: 100%;"">

//Loop through each of the route colors
foreach(string RouteName in ShuttleRoutes.RouteNames())
{
    //Make the row that displays each color with the down arrow. Both are links
    RouteInfo += "\n<tr align="center" style="width: 100%;">
        RouteInfo += "\n<td style="color:#C00000;font-family:Arial;font-size:Smaller;"">
            RouteInfo += "\n        RouteInfo += "\n<a href="javascript:void(0)" onClick="ToggleHidden('Hidden')">
                RouteInfo += "\n        RouteInfo += "\n<a href="javascript:void(0)" onClick="ChangeArrow('leftdownarrow')">
                        RouteInfo += "\n</a"
                    RouteInfo += "\n</a"
                RouteInfo += "\n</td">
        RouteInfo += "\n</tr>

    //This table row will contain the hidden table
    RouteInfo += "\n<tr id="Hidden" style="display: none;">";
            RouteInfo += "\n<td align="center" style="color:#C00000;font-family:Arial;font-size:Smaller;font-weight:bold;">
                RouteInfo += "\n</td">
        RouteInfo += "\n</tr>

    //Loops through the Route Numbers under the Route Name
    foreach(String RouteNumber in ShuttleRoutes.RouteNumbers(RouteName))
    {
        //Making the table
        RouteInfo += "\n<tr><td">
            RouteInfo += "\n<table style="font-size:smallest; background-color: #ffcc33;">

        //Route number header
        RouteInfo += "\n<tr align="center" width="100%" colspan="2">
                RouteInfo += "\n<tr align="center">
                    RouteInfo += "\n</td></tr>

        //Put in the default information
        RouteInfo += GenerateDefaultInfo("Latitude", ShuttleRoutes.GetSingleInfo(RouteNumber, "Latitude"));
            RouteInfo += GenerateDefaultInfo("Longitude", ShuttleRoutes.GetSingleInfo(RouteNumber, "Longitude"));
                RouteInfo += GenerateDefaultInfo("Speed", ShuttleRoutes.GetSingleInfo(RouteNumber, "Speed"));
```
(RouteName, RouteNumber, "Altitude"));
RouteInfo += GenerateDefaultInfo("On Time?", ShuttleRoutes.GetSingleInfo
(RouteName, RouteNumber, "OnTime"));
(RouteName, RouteNumber, "LastStop");
(RouteName, RouteNumber, "NextStop");
//Option to show all the data
//RouteInfo += "\n<tr><td align="center" width="100\%" colspan="2">";
//RouteInfo += "<a href=javascript:void(0) onClick=> Show All </a>";
//RouteInfo += "\n</td></tr>";

//End the table
RouteInfo += "\n</table>";
RouteInfo += "\n</td></tr>";
RouteInfo += "\n</table>";
RouteInfo += "\n</td>";
RouteInfo += "\n</tr>";

//Ends the table
RouteInfo += "\n</table>";
return RouteInfo;
}

/***************************************************************************/
* Generates the a new formatted table row with title InfoTitle and value value
***************************************************************************/
private string GenerateDefaultInfo(string InfoTitle, string value)
{
    string DefaultInfo = "";

    //Output the explanation to the data
    DefaultInfo += "\n<tr><td width="65\%" align="left" style="font-size:X-small\" >";
    DefaultInfo += InfoTitle;
    DefaultInfo += "\n</td>";

    //Output the data
    DefaultInfo += "\n<td width="35\%" style="font-size:X-small\" >";
    if (value == null)
    {
        DefaultInfo += "Not Obtained";
    }
    else
    {
        DefaultInfo += value;
    }
    DefaultInfo += "\n</td></tr>";

    return DefaultInfo;
}
<%@ Import Namespace="System.Data.SqlClient" %>
<%@ Import Namespace="System.Data" %>

<asp:Content ID="Content1" ContentPlaceHolderID="MainBodyPlaceHolder" Runat="Server">
<script type="text/javascript">
    <% Response.Write(GenerateJavaScript()); %>
</script>
<table style="width: 100%; height: 300px" border="0" cellspacing="0" cellpadding="0">
    <tr>
        <!-- First Column that displays the current route information -->
        <td style="width: 250px; height: 300px; background-color: #ffdd88;">
            <% Response.Write(GenerateCurrentRouteData()); %>
        </td>
        <!-- Column in the middle that displays the map -->
        <td style="height: 375px; width: 700px; align="center" valign="middle">
            <div id="map" style="width: 640px; height: 375px">
            </div>
        </td>
        <!-- Column on the right hand side where you can select the routes that you want to see -->
        <td style="width: 100px; height: 300px; background-color: #ffdd88;">
            <% Response.Write(GenerateRoutesToShow()); %>
        </td>
    </tr>
</table>
</asp:Content>
This is the default page. I want to put an editable list of the drivers and MACs and routes here.
Welcome to the help for the GPS Tracker Project

How can this shuttle tracking system help me?

This shuttle system will track the shuttle system at the University of Utah. It will allow you to know where shuttles are and how much time the shuttle has until it reaches you.

Who designed this shuttle system?

The shuttle tracking system is the senior project two computer engineering students who both rode the shuttle and knew how frustrating it can be?

I don't have a laptop to track the shuttle. How can I know where it is?

You can track the shuttle from any device that supports a standard internet browser such as Internet capable PDAs and cell phones. We are also busy working on getting email alerts and SMS message alerts to provide you with better shuttle information.

Why should I create an account?

An account will save settings specific to you. For example, say that you always want to see the green shuttle and no other shuttles. You can login and configure that display. Your preferences will automatically be saved so that the next time you login you will see the same display.
using System;
using System.Data;
using System.Configuration;
using System.Collections;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;
using System.Data.SqlClient;
using System.Data.Sql;

public partial class LoggedIn_Alarms : System.Web.UI.Page
{
    RadioButtonList RouteList = new RadioButtonList(); //Control to select the route ✔
    CheckBoxList DaysOfWeekList = new CheckBoxList(); //Control to select the days ✔
    DropDownList MinutesList = new DropDownList();
    DropDownList HoursList = new DropDownList();
    DropDownList AMPMList = new DropDownList();

    protected void Page_Load(object sender, EventArgs e)
    {
        //Gets all the information about the route
        Routes ShuttleRoutes = new Routes();

        //Add a header row to the table
        TableRow HeaderRow = new TableRow();
        Table1.Controls.Add(HeaderRow);
        HeaderRow.Width = Unit.Percentage(100);

        TableCell TitleCell = new TableCell();
        TitleCell.ColumnSpan = 6;
        TitleCell.Text = "Please set an alarm";
        TitleCell.Font.Bold = true;
        TitleCell.HorizontalAlign = HorizontalAlign.Center;
        TitleCell.Width = Unit.Percentage(100);
        HeaderRow.Controls.Add(TitleCell);

        //Add titles for each column
        TableRow TitleRow = new TableRow();
        Table1.Controls.Add(TitleRow);

        TableCell RouteCell = new TableCell();
        RouteCell.Text = "Route";
        RouteCell.Font.Bold = true;
        RouteCell.HorizontalAlign = HorizontalAlign.Center;
        TitleRow.Controls.Add(RouteCell);

        //Days of Week Title
        TableCell DaysOfWeekTitleCell = new TableCell();
        DaysOfWeekTitleCell.Text = "Days of Week";
        DaysOfWeekTitleCell.Font.Bold = true;
        DaysOfWeekTitleCell.HorizontalAlign = HorizontalAlign.Center;
        TitleRow.Controls.Add(DaysOfWeekTitleCell);

        //Hour Title
        TableCell HoursTitleCell = new TableCell();
        HoursTitleCell.Text = "Hour";
        HoursTitleCell.Font.Bold = true;
        HoursTitleCell.HorizontalAlign = HorizontalAlign.Center;
        TitleRow.Controls.Add(HoursTitleCell);

        //Minutes Title
TableCell MinutesTitleCell = new TableCell();
MinutesTitleCell.Text = "Minute";
MinutesTitleCell.Font.Bold = true;
MinutesTitleCell.HorizontalAlign = HorizontalAlign.Center;
TitleRow.Controls.Add(MinutesTitleCell);

// AM/PM Title
TableCell AMPMTTitleCell = new TableCell();
AMPMTTitleCell.Text = "Minute";
AMPMTTitleCell.Font.Bold = true;
MinutesTitleCell.HorizontalAlign = HorizontalAlign.Center;
TitleRow.Controls.Add(MinutesTitleCell);

// Add a row to the table
TableRow Row1 = new TableRow();
Table1.Controls.Add(Row1);
Row1.Width = Unit.Percentage(100);

// Add a cell with the route information in it
TableCell RouteSelectionCell = new TableCell();
foreach (string RouteName in ShuttleRoutes.RouteNames())
{
    foreach (string RouteNumber in ShuttleRoutes.RouteNumbers(RouteName))
    {
        RouteList.Items.Add(RouteName + " - " + RouteNumber);
    }
}
// Put the cell into the table row
Row1.Controls.Add(RouteSelectionCell);

// Add a cell with the days of the week in it
TableCell DaysOfWeekCell = new TableCell();
DaysOfWeekList.Items.Add("Monday");
DaysOfWeekList.Items.Add("Tuesday");
DaysOfWeekList.Items.Add("Wednesday");
DaysOfWeekList.Items.Add("Thursday");
DaysOfWeekList.Items.Add("Friday");
// Don't need these for the university shuttle system
// DaysOfWeekList.Items.Add("Saturday");
// DaysOfWeekList.Items.Add("Sunday");

DaysOfWeekCell.Width = Unit.Pixel(100);
// Put the cell into the table row
Row1.Controls.Add(DaysOfWeekCell);

// Add a cell with the days of the week in it
TableCell HoursCell = new TableCell();
for (int hour = 1; hour <= 12; hour++)
{
    HoursList.Items.Add(hour.ToString());
}
// Put the cell into the table row
Row1.Controls.Add(HoursCell);

// Add a cell with the days of the week in it
TableCell MinutesCell = new TableCell();
for (int minute = 0; minute <= 59; minute++)
{

MinutesList.Items.Add(minute.ToString());
}
//Put the cell into the table row
Row1.Controls.Add(MinutesCell);

//Add a cell with the days of the week in it
TableCell AMPMCell = new TableCell();
AMPMCell.Controls.Add(AMPMList);
AMPMList.Items.Add("AM");
AMPMList.Items.Add("PM");
AMPMList.AutoPostBack = true;
//Put the cell into the table row
Row1.Controls.Add(AMPMCell);

//I need to create the alarm here
//Add a cell with the days of the week in it
TableCell CreateAlarmButtonCell = new TableCell();
//DropDownList AMPMList = new DropDownList();
Button CreateButton = new Button();
CreateAlarmButtonCell.Controls.Add(CreateButton);
CreateButton.Visible = true;
CreateButton.Text = "Create Alarm";
CreateButton.Click += new EventHandler(CreateButton_Click);
//Put the cell into the table row
Row1.Controls.Add(CreateAlarmButtonCell);

//Fill the Grid View Control with the users alarms
//Open the database
string ConnectionString = "Data Source=(local);Initial Catalog=gpsData;uid=sa;pwd=XXXXXX;";
SqlConnection DatabaseConnection = new SqlConnection(ConnectionString);
DatabaseConnection.Open();

string UserAlertsQuery = "SELECT Type, DaysOfWeek, StartTime, EndTime FROM Alerts WHERE UserId = '" + Membership.GetUser().UserName + "';"
SqlCommand RetrieveUserAlertsCommand = new SqlCommand(UserAlertsQuery, DatabaseConnection);
SqlDataAdapter adapter = new SqlDataAdapter(RetrieveUserAlertsCommand);

//Fill the data set
DataSet pubs = new DataSet();
adapter.Fill(pubs, "Alerts");

//Perform the binding
GridView1.DataSource = pubs;
GridView1.DataBind();
}
void CreateButton_Click(object sender, EventArgs e)
{
UserMessage.InnerText = "Alarm is created: ";
UserMessage.InnerText += RouteList.SelectedItem;
String DaysOfWeek = "";
for(int CurrentDayIndex = 0; CurrentDayIndex<DaysOfWeekList.Items.Count; CurrentDayIndex++)
{
    if(DaysOfWeekList.Items[CurrentDayIndex].Selected == true)
    {
        DaysOfWeek += DaysOfWeekList.Items[CurrentDayIndex].Value;
    }
}
UserMessage.InnerText += DaysOfWeek;
UserMessage.InnerText += HoursList.SelectedItem;
UserMessage.InnerText += MinutesList.SelectedItem;
UserMessage.InnerRadiusText += AMPMList.SelectedItem;

// Test to see if username is null
if (Membership.GetUser().UserName == null)
{
    UserMessage.InnerText += "User name is null";
}
else
{
    UserMessage.InnerText += "User name is: " + Membership.GetUser().UserName;
}

//***************************************************************************
// Add the alert to the database
//***************************************************************************

// Open the database
string ConnectionString = "Data Source=(local);Initial Catalog=gpsData;uid=sa;pwd=XXXXXXX;"
SqlConnection DatabaseConnection = new SqlConnection(ConnectionString);
DatabaseConnection.Open();

// Get the number of alerts that have been set for the current user
String AlertCountQuery = "SELECT COUNT(*) AS AlertCount FROM Alerts WHERE UserId =" + Membership.GetUser().UserName + ""
SqlCommand GetAlertCountCommand = new SqlCommand(AlertCountQuery, DatabaseConnection);
SqlDataReader AlertCountReader = GetAlertCountCommand.ExecuteReader();
// If the Gumstix has information, display it
AlertCountReader.Read();
int AlertCount = Int32.Parse(AlertCountReader["AlertCount"].ToString());
AlertCountReader.Close();

// Make an autogenerated name for the alarm
String AlarmName = Membership.GetUser().UserName + AlertCount.ToString();

// Schedule the alarm
s.OnTrigger += new EventScheduler.Invoke(TriggerAlarm);
EventScheduler.Scheduler.AddSchedule(s);

// Put the alarm in
String CommandString = "INSERT INTO Alerts (UserId, Name, Type, DaysOfWeek, StartTime, EndTime, Created) VALUES (" + Membership.GetUser().UserName + ", " + AlarmName + ", " + TestAlarm + ", " + DaysOfWeek + ", " + DateTime.Now + ", " + DateTime.Now + ");";
SqlCommand InsertCommand = new SqlCommand(CommandString, DatabaseConnection);
SqlDataReader ReaderThatInserts = InsertCommand.ExecuteReader();
ReaderThatInserts.Close();

// Close the database
DatabaseConnection.Close();

}

void TriggerAlarm(String AlarmName)
{
    UserMessage.InnerRadiusText += "The alarm has gone off at: " + DateTime.Now;
    MembershipUser myMembershipUser = Membership.GetUser();
    // Page.
}
<%@ Page Language="C#" MasterPageFile="/LoggedIn/LoggedIn.master" AutoEventWireup="true" %>
<asp:Content ID="Content1" ContentPlaceHolderID="MainBodyPlaceHolder2" Runat="Server">
    <div>
        &nbsp;
    </div>
    <asp:Table ID="Table1" runat="server" align="center">
        <div id="UserMessage" runat="server">
            <asp:GridView ID="GridView1" runat="server" AllowPaging="True" BorderStyle="None" align="center">
                <RowStyle BorderColor="White" ForeColor="Maroon" />
                <HeaderStyle BackColor="#FFC080" />
                <AlternatingRowStyle BackColor="#FFE0C0" />
            </asp:GridView>
        </div>
    </asp:Table>
</asp:Content>
using System;
using System.Data;
using System.Data.SqlClient;
using System.Configuration;
using System.Collections;
using System;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;
using System.Collections.Generic;
using System.Collections.Generic;
using System.Collections.Generic;

public partial class LoggedIn_Master : System.Web.UI.Page
{
    #region Global Variables
    private string connectionString;
    List<List<List<string>>> DataStructure;
    List<string> RouteNames;
    #endregion

    protected void Page_Load(object sender, EventArgs e)
    {
        Get3dData();
    }

    private void Get3dData()
    {
        //I want to pull the data here and put it in a 3D array
        string connectionString = WebConfigurationManager.ConnectionStrings["GPSDataConnectionString"].ConnectionString;
        string RoutesQuery = "SELECT * FROM dbo.RouteStatus";
        SqlConnection RoutesConnection = new SqlConnection(connectionString);
        SqlCommand RoutesCommand = new SqlCommand(RoutesQuery, RoutesConnection);
        SqlDataAdapter RoutesAdapter = new SqlDataAdapter(RoutesCommand);
        DataSet RoutesData = new DataSet();
        RoutesConnection.Open();
        RoutesAdapter.Fill(RoutesData, "GPSData");

        //Holds each unique route
        RouteNames = new List<string>();
        DataStructure = new List<List<List<string>>>();

        //Go through each row
        foreach (DataRow row in RoutesData.Tables["GPSData"].Rows)
        {
            string RouteName = row["routeName"].ToString();
            string RouteNumber = row["routeNumber"].ToString();

            //TODO: I need to make an indexer for this
            List<string> CurRouteInfo = new List<string>();
            CurRouteInfo.Add(RouteName);
            CurRouteInfo.Add(RouteNumber);
            CurRouteInfo.Add(row["Latitude"].ToString());
            CurRouteInfo.Add(row["Longitude"].ToString());
            CurRouteInfo.Add(row["speed"].ToString());
            CurRouteInfo.Add(row["altitude"].ToString());
            CurRouteInfo.Add(row["MAC"].ToString());
            CurRouteInfo.Add(row["OnTime"].ToString());
            CurRouteInfo.Add(row["LastStop"].ToString());
            CurRouteInfo.Add(row["NextStop"].ToString());

            //if it is a new route name add a list for columns for it and put the first
        }
    }
}
entry in the list
if (!RouteNames.Contains(RouteName))
{
    //Add it to the list of route names already in use
    RouteNames.Add(RouteName);

    //Create a new list for columns for route numbers on the 3d data structure
    List<List<string>> curRouteNumber = new List<List<string>>() { };

    //Put the first route number's info in
    curRouteNumber.Add(CurRouteInfo);

    //Put it into the 3D data structure
    DataStructure.Add(curRouteNumber);
}
else
{
    //Put it in the right place
    DataStructure[RouteNames.IndexOf(RouteName)].Add(CurRouteInfo);
}

protected string GenerateJavaScript()
{
    string JavaScript = "";
    JavaScript += "\n\nfunction ToggleHidden(r)"
    JavaScript += "\n{";
    JavaScript += "\nif(document.getElementById(r).style.display == '')"
    JavaScript += "\n{";
    JavaScript += "\ndocument.getElementById(r).style.display = 'none';";
    JavaScript += "\n";
    JavaScript += "\nelse"
    JavaScript += "\n{";
    JavaScript += "\ndocument.getElementById(r).style.display = '';";
    JavaScript += "\n";
    JavaScript += "\n}";
    JavaScript += "\nfunction ToggleHiddenInfo(RouteNumber)"
    JavaScript += "\n{";
    JavaScript += "\nif(document.getElementById("Route" + RouteNumber).style.display == '')"
    JavaScript += "\n{";
    JavaScript += "\ndocument.getElementById("Route" + RouteNumber).style.display = 'none';";
    JavaScript += "\n";
    JavaScript += "\nelse"
    JavaScript += "\n{";
    JavaScript += "\ndocument.getElementById("Route" + RouteNumber).style.display = '';";
    JavaScript += "\n";
    JavaScript += "\n}";
    JavaScript += "\n";
    JavaScript += "\n";

    //This function will toggle an arrow when it is given the image name
    JavaScript += "\n\nfunction ChangeArrow(img_name)"
    JavaScript += "\n{";
    JavaScript += "\nif(document[img_name].src.match('dnarrow') == 'dnarrow')"
    JavaScript += "\n{";
    JavaScript += "\ndocument[img_name].src = '..\images\uparrow9x5.png';";
    JavaScript += "\n";
    JavaScript += "\nelse"
    JavaScript += "\n{";
    JavaScript += "\ndocument[img_name].src = '..\images\dnarrow9x5.png';";
    JavaScript += "\n";
    JavaScript += "\n}";
    JavaScript += "\nreturn JavaScript;";
protected string GenerateRoutesToShow()
{
    // TODO: Try and execute JavaScript here in C# that will show all the routes when I click on the color of the route
    // Output the start of the table that allows you to select routes
    string RoutesToShow = "\r\n<tr style="width: 100%" id="Table1">\r\n";
    
    foreach (string RouteName in RouteNames)
    {
        // Outputs the main rows with links that show everything and the arrow to show individual routes
        RoutesToShow += "\r\n<table style="width: 100%" id="Table1">
\r\n<thead><tr><th>RouteName: " + RouteName + ", Size: " + RoutesToShow + 
\r\n<tbody><tr>
\r\n" + RoutesToShow + 
\r\n"</tr></tbody></table>";

        // Make the hidden tables that show the route numbers
        RoutesToShow += "\r\n<table id="Hidden" + RouteName + " RouteNumbers style="display:none;"\r\n" + RoutesToShow + 
\r\n<thead><tr><th>RouteName: " + RouteName + ", Size: " + RoutesToShow + 
\r\n<tbody><tr>
\r\n" + RoutesToShow + 
\r\n"</tr></tbody></table>";

        RoutesToShow += "\r\n" + RoutesToShow + 
\r\n"</tr></tbody></table>";
    }

    // Append the code for the end of the table
    RoutesToShow += "\r\n</tr></tbody></table>";

    return RoutesToShow;
}

protected string GenerateCurrentRouteData()
{
    // TODO: Put in the Configuration files a way to specify the data I want
    // Get the delay time and connection string from the configuration file
    string RouteInfo = "";

    NameValueCollection appSettings = System.Configuration.ConfigurationSettings.AppSettings;

    Boolean LatitudeDefault = appSettings[0] == "yes" ? true : false;
    Boolean LongitudeDefault = appSettings[1] == "yes" ? true : false;
    Boolean SpeedDefault = appSettings[2] == "yes" ? true : false;
    Boolean AltitudeDefault = appSettings[3] == "yes" ? true : false;
    Boolean LastStopDefault = appSettings[6] == "yes" ? true : false;
    Boolean NextStopDefault = appSettings[7] == "yes" ? true : false;
    Boolean LatitudeAllow = appSettings[8] == "yes" ? true : false;
    Boolean LongitudeAllow = appSettings[9] == "yes" ? true : false;
Boolean SpeedAllow = appSettings[10] == "yes" ? true : false;
Boolean DriverAllow = appSettings[12] == "yes" ? true : false;
Boolean LastStopAllow = appSettings[14] == "yes" ? true : false;
Boolean NextStopAllow = appSettings[15] == "yes" ? true : false;

// Output the route name with the arrow to the right hand side
// When I click on blue I want to see each route with it's associated info
// When I click on the down arrow I want to see a list of routes and be able to select which data is shown
// At the bottom of the field I want to see
// I want each route info table to have a slightly darker background to set it apart

// Start the table
RouteInfo += "\n<table>";

// Loop through each of the route colors
foreach (string RouteName in RouteNames)
{
    // Make the row that displays each color with the down arrow. Both are links
    RouteInfo += "\n<tr align="center">\n";
    RouteInfo += "\n<td style="color:#C00000;font-family:Arial;font-size:Smaller;" font-weight:bold;">\n";
    RouteInfo += "\n<a href="javascript:void(0)" onClick="ToggleHidden('Hidden') + RouteName + 'InfoRoutes');\"">" + RouteName + "</a>;\n";
    RouteInfo += "\n<td style="color:#C00000;font-family:Arial; font-size:Smaller; font-weight:bold;">\n";
    RouteInfo += "\n\n</td>;\n";
    RouteInfo += "\n</tr>";

    // This table row will contain the hidden table
    RouteInfo += "\n<tr id="Hidden" + RouteName + 'InfoRoutes style="display: none;"">\n";
    RouteInfo += "\n<td align="center" style="color:#C00000;font-family:Arial; font-size:Smaller; font-weight:bold;">\n";
    RouteInfo += "\n</td>;\n";
    RouteInfo += "\n</tr>";

    // Loops through the Route Numbers under the Route Name
    for (int i = 0; i < DataStructure[RouteNames.IndexOf(RouteName)].Count; i++)
    {
        RouteInfo += "\n<tr><td>\n";

        // Making the table
        RouteInfo += "\n<table style="font-size:smallest; background-color: #ffcc33;">\n";

        // Route number header
        RouteInfo += "\n<tr align="center" width="100%" colspan="2">\n";
        RouteInfo += "\n<td align="center" width="100%" colspan="2" >\n";
        RouteInfo += "\n\n</td>\n";
        RouteInfo += "\n</tr>";

        // Put in the default information
        RouteInfo += GenerateDefaultInfo(LatitudeDefault, "Latitude", 2, RouteName, i);
        RouteInfo += GenerateDefaultInfo(LongitudeDefault, "Longitude", 3, RouteName, i);
        RouteInfo += GenerateDefaultInfo(SpeedDefault, "Speed", 4, RouteName, i);
        RouteInfo += GenerateDefaultInfo(AltitudeDefault, "Altitude", 5, RouteName, i);
        RouteInfo += GenerateDefaultInfo(DriverDefault, "Driver", 6, RouteName, i);
    }

    RouteInfo += "\n</table>";
}
RouteInfo += GenerateDefaultInfo(AltitudeDefault, "Altitude", 7, RouteName, i);
RouteInfo += GenerateDefaultInfo(LastStopDefault, "Last Stop", 8, RouteName, i);
RouteInfo += GenerateDefaultInfo(NextStopDefault, "Next Stop", 9, RouteName, i);

// Option to show all the data
RouteInfo += "\n<tr><td align="center" width="100%" colspan="2">";
RouteInfo += "<a href=javascript:void(0) onClick= Show All /></a>";
RouteInfo += "\n</td></tr>";

// End the table
RouteInfo += "\n</table>";
RouteInfo += "\n</td></tr>";
}
return RouteInfo;

private string GenerateDefaultInfo(bool IsPrintDefault, string InfoTitle, int InfoNumber, string RouteName, int RouteNumber)
{
    string DefaultInfo = "";
    if (IsPrintDefault)
    {
        // Output the explanation to the data
        DefaultInfo += "\n<tr><td width="65%" align="left" style="font-size:X-small">";
        DefaultInfo += InfoTitle;
        DefaultInfo += "\n</td>";

        // Output the data
        DefaultInfo += "\n<td width="35%" style="font-size:X-small">";
        if (DataStructure[RouteNames.IndexOf(RouteName)][RouteNumber][InfoNumber] == "")
        {
            DefaultInfo += "Not Obtained";
        }
        else
        {
            DefaultInfo += DataStructure[RouteNames.IndexOf(RouteName)][RouteNumber][InfoNumber];
        }
    }
    DefaultInfo += "\n</td></tr>";
    return DefaultInfo;
}
<%@ Page Language="C#" MasterPageFile="~/loggedIn/LoggedIn.master" AutoEventWireup="true" %>
<codefile>Default.aspx.cs</codefile> Inherits="LoggedIn_Master" %>
<% Import Namespace="System.Data.SqlClient" %>
<% Import Namespace="System.Data" %>

<asp:Content ID="Content1" ContentPlaceHolderID="MainBodyPlaceHolder2" Runat="Server">
  <script type="text/javascript">
    <% Response.Write(GenerateJavaScript()); %>  
  </script>
  <table style="width: 100%; height: 300px" border="0" cellpadding="0" cellspacing="0">
    <tr>
      <!-- First Column that displays the current route information -->
      <td style="width: 250px; height: 300px; background-color: #ffdd88;">
        <% Response.Write(GenerateCurrentRouteData()); %>  
      </td>
    </tr>
    <!-- Column in the middle that displays the map -->
    <td style="height: 300px; width: 700px; align="center" valign="middle">
      <div id="map" style="width: 640px; height: 300px"></div>
    </td>
    <!-- Column on the right hand side where you can select the routes that you want to see -->
    <td style="width: 100px; height: 300px; background-color: #ffdd88;">
      <% Response.Write(GenerateRoutesToShow()); %>  
    </td>
  </tr>
</table>
</asp:Content>
This is the help page. It is still under construction
public partial class MasterPage : System.Web.UI.MasterPage
{
    protected void Page_Load(object sender, EventArgs e)
    {
    }

    protected void LogoutButton_Click(object sender, EventArgs e)
    {
        FormsAuthentication.SignOut();
        Response.Redirect("../Default.aspx");
    }

    public string GetMACs()
    {
        // Format that I want to output it in
        // var city = new Array("New York", "London", "New Delhi");
        string connectionString = WebConfigurationManager.ConnectionStrings["GPSDataConnectionString"].ConnectionString;
        string RoutesQuery = "SELECT DISTINCT MAC FROM dbo.UnitOnRoute";
        SqlConnection RoutesConnection = new SqlConnection(connectionString);
        SqlCommand RoutesCommand = new SqlCommand(RoutesQuery, RoutesConnection);
        SqlDataAdapter RoutesAdapter = new SqlDataAdapter(RoutesCommand);
        DataSet RoutesData = new DataSet();
        RoutesConnection.Open();
        RoutesAdapter.Fill(RoutesData, "GPSData");
        string MACs = null;

        // Go through each row
        int RowsExamined = 0;
        foreach (DataRow row in RoutesData.Tables["GPSData"].Rows)
        {
            MACs += "DisplayLine(\"" + row["MAC"].ToString() + ")\n"
        }
        MACs = MACs.Replace(":", ",");
        return MACs;
    }
}
function onLoad(lastLat, lastLng) {
  if (GBrowserIsCompatible()) {
    //Blank icon
    var icon = new GIcon();
    icon.image = "./images/blank.png"
    icon.shadow = "./images/blank.png"
    icon.iconSize = new GSize(1, 1);
    icon.shadowSize = new GSize(1, 1);
    icon.iconAnchor = new GPoint(1, 1);
    icon.infoWindowAnchor = new GPoint(1, 1);

    //Icon that has the shuttle picture
    var icon2 = new GIcon();
    icon2.image = "./images/shuttle.png"
    icon2.shadow = "./images/shuttle.png"
    icon2.iconSize = new GSize(20, 10);
    icon2.shadowSize = new GSize(1, 1);
    icon2.iconAnchor = new GPoint(1, 1);
    icon2.infoWindowAnchor = new GPoint(1, 1);

    //Making the map and setting its properties
    map = new GMap2(document.getElementById("map"));
    map.setCenter(new GLatLng(lastLat, lastLng), 15);
    map.addControl(new GSmallMapControl());
    map.setMapType(G_HYBRID_TYPE);
    busMarker = new GMarker(map.getCenter(), icon2);
    map.addOverlay(busMarker);
  }
}
function ChangeCenter(){
   //centerMAC is the MAC without colons of the bus that we want to center on

   GDownloadUrl("../data/" + centerMAC + ".txt", function(data, responseCode)
   {
      var xml = GXml.parse(data);
      var markers = xml.documentElement.getElementsByTagName("marker");
      map.panTo(new GLatLng(parseFloat(markers[markers.length-1].getAttribute("lat")),
                  parseFloat(markers[markers.length-1].getAttribute("lng"))));
   });
}

function DisplayAllLines()
{
   //map.clearOverlays();
   <%Response.Write(GetMACs());%>
}

function DisplayLine(MACAddress){
   //TODO: Put a setting in the web.config file for the file path here
   //TODO: Rename the files as XML
   GDownloadUrl("../data/" + MACAddress + ".txt", function(data, responseCode)
   {
      var xml = GXml.parse(data);
      var markers = xml.documentElement.getElementsByTagName("marker");

      var points = [];
      for (var i = 0; i < markers.length; i++) {
         points.push(new GLatLng(parseFloat(markers[i].getAttribute("lat")),
                            parseFloat(markers[i].getAttribute("lng"))));
         centerLat = markers[i].getAttribute("lat");
         centerLng = markers[i].getAttribute("lng");
      }
      map.addOverlay(new GPolyline(points));
   });
}

///]]>
</script>
</head>
<body style="background-color: #ffeecc" onload="onLoad(40.771, -111.845)">
<form id="form1" runat="server">
   <table style="width: 100%; height: 100%"><tr style="width: 100%; height: 10px">
      <td style="width: 10%; height: 10px;"><table style="width: 100%; background-color: #ffdd88;" border="0">
         <tr cellpadding="0" cellspacing="0">
            <td style="width: 10%; />
            <td style="height: 1px; font-family: Verdana; font-weight:খ" />&gt;
         </tr>
      </table>
   </td>
</table>
<form>
</form>
</body>
</html>
using System;
using System.Data;
using System.Data.SqlClient;
using System.Configuration;
using System.Collections;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;
using System.Collections.Generic;
using System.Collections.Specialized;

public partial class PDA : System.Web.UI.Page
{
    #region Global Variables
    private string connectionString;
    List<List<string>> DataStructure;
    List<string> RouteNames;
    #endregion

    protected void Page_Load(object sender, EventArgs e)
    {
        Get3dData();
    }

    private void Get3dData()
    {
        // I want to pull the data here and put it in a 3D array
        connectionString = WebConfigurationManager.ConnectionStrings["GPSDataConnectionString"].ConnectionString;
        string RoutesQuery = "SELECT * FROM dbo.RouteStatus";
        SqlConnection RoutesConnection = new SqlConnection(connectionString);
        SqlCommand RoutesCommand = new SqlCommand(RoutesQuery, RoutesConnection);
        SqlDataAdapter RoutesAdapter = new SqlDataAdapter(RoutesCommand);
        DataSet RoutesData = new DataSet();
        RoutesConnection.Open();
        RoutesAdapter.Fill(RoutesData, "GPSData");

        // Holds each unique route
        RouteNames = new List<string>();
        DataStructure = new List<List<string>>();

        // Go through each row
        foreach (DataRow row in RoutesData.Tables["GPSData"].Rows)
        {
            string RouteName = row["routeName"].ToString();
            string RouteNumber = row["routeNumber"].ToString();

            // TODO: I need to make an indexer for this
            List<string> CurRouteInfo = new List<string>();
            CurRouteInfo.Add(RouteName);
            CurRouteInfo.Add(RouteNumber);
            CurRouteInfo.Add(row["Latitude"].ToString());
            CurRouteInfo.Add(row["Longitude"].ToString());
            CurRouteInfo.Add(row["speed"].ToString());
            CurRouteInfo.Add(row["altitude"].ToString());
            CurRouteInfo.Add(row["Driver"].ToString());
            CurRouteInfo.Add(row["MAC"].ToString());
            CurRouteInfo.Add(row["OnTime"].ToString());
            CurRouteInfo.Add(row["LastStop"].ToString());
            CurRouteInfo.Add(row["NextStop"].ToString());
        }
    }
}
protected string GenerateJavaScript()
{
    string JavaScript = "";
    JavaScript += "\n
function ToggleHidden(r)"
    JavaScript += "\n"
    JavaScript += "\n";
    JavaScript += "\nif(document.getElementById(r).style.display == '')";
    JavaScript += "\n"
    JavaScript += "\n";
    JavaScript += "\ndocument.getElementById(r).style.display = "none";
    JavaScript += "\n"
    JavaScript += "\n";
    JavaScript += "\nelse"
    JavaScript += "\n"
    JavaScript += "\n";
    JavaScript += "\ndocument.getElementById(r).style.display = ''";
    JavaScript += "\n"
    JavaScript += "\n";
    JavaScript += "\nfunction ToggleHiddenInfo(RouteNumber)"
    JavaScript += "\n"
    JavaScript += "\nif(document.getElementById(\"Route\" + RouteNumber).style.display == '')";
    JavaScript += "\n"
    JavaScript += "\n"
    JavaScript += "\ndocument.getElementById(\"Route\" + RouteNumber).style.display = "none";
    JavaScript += "\n"
    JavaScript += "\n";
    JavaScript += "\nelse"
    JavaScript += "\n"
    JavaScript += "\n";
    JavaScript += "\ndocument.getElementById(\"Route\" + RouteNumber).style.display = ''";
    JavaScript += "\n"
    JavaScript += "\n";
    JavaScript += "\n//This function will toggle an arrow when it is given the image name"
    JavaScript += "\n\nfunction ChangeArrow(img_name)"
    JavaScript += "\n"
    JavaScript += "\nif(img_name.src.match('dnarrow') == 'dnarrow')";
    JavaScript += "\n"
    JavaScript += "\n";
    JavaScript += "\ndocument[img_name].src = '../images/uparrow9x5.png'";
    JavaScript += "\n"
    JavaScript += "\n";
    JavaScript += "\nelse"
    JavaScript += "\n"
    JavaScript += "\n";
    JavaScript += "\ndocument[img_name].src = '../images/dnarrow9x5.png'";
    JavaScript += "\n"
    JavaScript += "\n";
    JavaScript += "\n";
    JavaScript += "\n";
protected string GenerateRoutesToShow()
{
   // TODO: Try and execute JavaScript here in C# that will show all the routes when I click on the color of the route
   // Output the start of the table that allows you to select routes
   string RoutesToShow = "<table style="width: 100\%" id="Table1">");

   foreach (string RouteName in RouteNames)
   {
      // Outputs the main rows with links that show everything and the arrow to show individual routes
      RoutesToShow += "\n<tr align="center">
   
   RoutesToShow += "<a href="javascript:void(0)" onClick="ShowAllRoutes(' + RouteName + '");">"
   RoutesToShow += "<" + RouteName + "'/>");
   RoutesToShow += "<a href="javascript:void(0)" onclick="ToggleHidden('Hidden " + RouteName + "Routes');">"
   RoutesToShow += "<" + RouteName + "'/>");
   RoutesToShow += "<a href="javascript:void(0)" onclick="DisplayLine(' + RouteName + "]' +
   RoutesToShow += "<" + 
   RoutesToShow += "</a>&nbsp;|&nbsp;");

   // Append the code for the end of the table
   RoutesToShow += "</table>");

   return RoutesToShow;
}

protected string GenerateCurrentRouteData()
{
   // TODO: Put in the Configuration files a way to specify the data I want
   // Get the delay time and connection string from the configuration file
   string RouteInfo = "";

   NameValueCollection appSettings = System.Configuration.ConfigurationSettings. "";
   AppSettings;
   Boolean LatitudeDefault = appSettings[0] == "yes" ? true : false;
   Boolean LongitudeDefault = appSettings[1] == "yes" ? true : false;
   Boolean SpeedDefault = appSettings[2] == "yes" ? true : false;
   Boolean AltitudeDefault = appSettings[3] == "yes" ? true : false;
   Boolean LastStopDefault = appSettings[6] == "yes" ? true : false;
   Boolean NextStopDefault = appSettings[7] == "yes" ? true : false;
   Boolean LatitudeAllow = appSettings[8] == "yes" ? true : false;
Boolean LongitudeAllow = appSettings[9] == "yes" ? true : false;
Boolean SpeedAllow = appSettings[10] == "yes" ? true : false;
Boolean DriverAllow = appSettings[12] == "yes" ? true : false;
Boolean LastStopAllow = appSettings[14] == "yes" ? true : false;
Boolean NextStopAllow = appSettings[15] == "yes" ? true : false;

//Output the route name with the arrow to the right hand side
//When I click on blue I want to see each route with it's associated info
//When I click on the downarrow I want to see a list of routes and be able to select which data is shown
//At the bottom of the field I want to see
//I want each route info table to have a slightly darker background to set it apart and maybe a border

//Start the table
RouteInfo += "\n<table>";

//Loop through each of the route colors
foreach (string RouteName in RouteNames)
{
  //Make the row that displays each color with the down arrow. Both are links
  RouteInfo += "\n<tr align="center">
  RouteInfo += "<td style="color:#C00000;font-family:Arial;font-size:Smaller; font-weight:bold;">";
  RouteInfo += "<a href="javascript:void(0)" onclick="ToggleHidden('Hidden" + RouteName + "InfoRoutes');">" + RouteName + "</a>"
  RouteInfo += "<img name="leftdownarrow" + RouteName + ");"></td>
  RouteInfo += "</tr>
  RouteInfo += "\n</table>";

  //This table row will contain the hidden table
  RouteInfo += "\n<tr id="Hidden" + RouteName + ">"InfoRoutes style="display: none;">"
  RouteInfo += "<td align="center" style="color:#C00000;font-family:Arial; font-size:Smaller;font-weight:bold;">"
  RouteInfo += "<img src="/images/dnarrow9x5.png style="border-style:None;border-width:0px;"/">"
  RouteInfo += "</td>
  RouteInfo += "</tr>
}

//Loops through the Route Numbers under the Route Name
for (int i = 0; i < DataStructure[RouteNames.IndexOf(RouteName)].Count; i++)
{
  RouteInfo += "\n<tr><td>";

  //Making the table
  RouteInfo += "\n<table style="font-size:smallest; background-color: #ffe333;">"

  //Number header
  RouteInfo += "\n<tr align="center" width="100%" colspan="2">"
  RouteInfo += "<td align="center" nowrap="true" style="border-style:None;border-width:0px;"/"
  RouteInfo += "</table>
  RouteInfo += "\n</tr>");

  //Put in the default information
  RouteInfo += GenerateDefaultInfo(LatitudeDefault, "Latitude", 2, RouteName, i);
  RouteInfo += GenerateDefaultInfo(LongitudeDefault, "Longitude", 3, RouteName, i);
  RouteInfo += GenerateDefaultInfo(SpeedDefault, "Speed", 4, RouteName, i);
  RouteInfo += GenerateDefaultInfo(AltitudeDefault, "Altitude", 5, RouteName, i);
  RouteInfo += GenerateDefaultInfo(DriverDefault, "Driver", 6, RouteName, i)"
RouteInfo += GenerateDefaultInfo(AltitudeDefault, "Altitude", 7, RouteName, i);
RouteInfo += GenerateDefaultInfo(LastStopDefault, "Last Stop", 8, RouteName, i);
RouteInfo += GenerateDefaultInfo(NextStopDefault, "Next Stop", 9, RouteName, i);

// Option to show all the data
RouteInfo += "\n<tr><td align="center" width="100%" colspan="2">";
RouteInfo += "<a href=javascript: void(0) onClick=> Show All </a>";
RouteInfo += "\n</td></tr>";
// End the table
RouteInfo += "\n</table>";
RouteInfo += "\n</td></tr>";

// Ends the table
RouteInfo += "\n</table>";
return RouteInfo;

private string GenerateDefaultInfo(bool IsPrintDefault, string InfoTitle, int InfoNumber, string RouteName, int RouteNumber)
{
    string DefaultInfo = "";
    if (IsPrintDefault)
    {
        // Output the explanation to the data
        DefaultInfo += "\n<tr><td width="65%" align="left" style="font-size:X-small">";
        DefaultInfo += InfoTitle;
        DefaultInfo += "\n</td>";

        // Output the data
        DefaultInfo += "\n</td width="35%" style="font-size:X-small">";
        if (DataStructure[RouteNames.IndexOf(RouteName)][RouteNumber][InfoNumber] == "")
        {
            DefaultInfo += "Not Obtained";
        }
        else
        {
            DefaultInfo += DataStructure[RouteNames.IndexOf(RouteName)][RouteNumber][InfoNumber];
        }
        DefaultInfo += "\n</td>";
    }
    return DefaultInfo;
}
<@ Page Language="C#" MasterPageFile="/loggedIn/Login/LoggedIn.master" AutoEventWireup="true" CodeFile="PDA.aspx.cs" Inherits="PDA" Title="Untitled Page" %>
<% Import Namespace="System.Data.SqlClient" %>
<% Import Namespace="System.Data" %>
<-- First Column that displays the current route information -->
<table style="width: 475px; height: 150px" border="0" cellpadding="0" cellspacing="0">
<tr>
<td style="width: 100px; height: 150px; background-color: #ffdd88;">
<% Response.Write(GenerateCurrentRouteData()); %>
</td>
</tr>
</table>
<-- Column in the middle that displays the map -->
<div id="map" style="width: 260px; height: 150px"></div>
<-- Column on the right hand side where you can select the routes that you want to see -->
<table style="width: 75px; height: 150px; background-color: #ffdd88;">
<tr>
<td style="width: 75px; height: 150px; background-color: #ffdd88;">
<% Response.Write(GenerateRoutesToShow()); %>
</td>
</tr>
</table>
</asp:Content>
using System;
using System.Data;
using System.Configuration;
using System.Collections;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;
using System.Data.SqlClient;

public partial class MasterPage : System.Web.UI.MasterPage
{
    Routes IconCopy;

    protected void Page_Load(object sender, EventArgs e)
    {
        IconCopy = new Routes();
    }

    /**
     * DrawEveryLine will return JavaScript code to draw every line from the database
     * It uses the DisplayLine JavaScript function to do that.
     * How it determines what lines to generate JavaScript DisplayLine code for - it takes
     * unique MACs from the UnitOnRoute table.
     * */
    public string DrawEveryLine()
    {
        string MACs = null; //String that will have the JavaScript that I want to
        return
            //Access all the MACs from the UnitOnRoute table in the database
            string connectionString = WebConfigurationManager.ConnectionStrings[
                "GPSDataConnectionString"].ConnectionString;
            string RoutesQuery = "SELECT DISTINCT MAC FROM dbo.UnitOnRoute";
            SqlConnection RoutesConnection = new SqlConnection(connectionString);
            SqlCommand RoutesCommand = new SqlCommand(RoutesQuery, RoutesConnection);
            SqlDataAdapter RoutesAdapter = new SqlDataAdapter(RoutesCommand);
            DataSet RoutesData = new DataSet();
            RoutesConnection.Open();
            RoutesAdapter.Fill(RoutesData, "GPSData");
            RoutesConnection.Close();
            //Go through each row and put in the text to display a line for that MAC
            foreach (DataRow row in RoutesData.Tables["GPSData"].Rows)
            {
                //MACs += "\nif(Display) + row["MAC"].ToString() + ")\n";
                MACs += "\nDisplayLine(\"" + row["MAC"].ToString() + \\"\")\n";
                //MACs += "\nalert('Works' + row["MAC"].ToString() + \\"");
                //MACs += "\n"");
            }
            MACs = MACs.Replace(";", ";");

            return MACs;
    }

    public string MakeShuttleIconVariables()
    {
        string ShuttleVariableString = "";
        foreach (string CurrentMAC in IconCopy.GetAllMACs())
        {
            ShuttleVariableString += "var ShuttleIcon" + CurrentMAC.Replace(";", ";") + ";\n"
        }

        return ShuttleVariableString;
    }
}
public String DefineShuttleIcons()
{
    String ShuttleVariableSpecs = "";
    foreach (String CurrentMAC in IconCopy.GetAllMACs())
    {
        ShuttleVariableSpecs += "ShuttleIcon" + CurrentMAC.Replace(":", "") + " = new GIcon()
        = "images/shuttle.png";\n"
        ShuttleVariableSpecs += "ShuttleIcon" + CurrentMAC.Replace(":", "") + ".image 
        iconSize = new GSize(20, 10);\n"
        ShuttleVariableSpecs += "ShuttleIcon" + CurrentMAC.Replace(":", "") + ".
        shadowSize = new GSize(1, 1);\n"
        ShuttleVariableSpecs += "ShuttleIcon" + CurrentMAC.Replace(":", "") + ".
        iconAnchor = new GPoint(1, 1);\n"
        ShuttleVariableSpecs += "ShuttleIcon" + CurrentMAC.Replace(":", "") + ".
        infoWindowAnchor = new GPoint(1, 1);\n"
    }
    return ShuttleVariableSpecs;
}

public String DisplayCurrentShuttleIcon()
{
    String ShuttleIconDisplay = "";
    foreach (String CurrentMAC in IconCopy.GetAllMACs())
    {
        ShuttleIconDisplay += "if(MACAddress == \"" + CurrentMAC.Replace(":", "") + "\"
        "\")\n"
        ShuttleIconDisplay += "\"
        ShuttleIconDisplay += "Marker" + CurrentMAC.Replace(":", "") + " new GMarker 
        (new GLatLng(centerLat, centerLng), ShuttleIcon" + CurrentMAC.Replace(":", "") + ");\n"
        //ShuttleIconDisplay += "alert("adding a marker\")";
        ShuttleIconDisplay += "map.addOverlay(X" + CurrentMAC.Replace(":", "") + "
        "");\n"
        ShuttleIconDisplay += "\"
        ShuttleIconDisplay += "\n";
        ShuttleIconDisplay += "\n";
        ShuttleIconDisplay += "\n"
    }
    return ShuttleIconDisplay;
}

public String AddHistory()
{
    String LinesToShow = "";
    /*foreach (String CurrentMAC in IconCopy.GetAllMACs())
    {
        LinesToShow += "nif(Display) + CurrentMAC.Replace(":", ") + 
        LinesToShow += "nmap.addOverlay(new GPolyline(points));\n        LinesToShow += "\n"
    }*/
    return LinesToShow;
}
<% @Master Language="C#" AutoEventWireup="true" CodeFile="LoggedOut.master.cs" Inherits="MasterPage" %>

<!DOCTYPE html PUBLIC "+//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="content-type" content="text/html; charset=utf-8"/>

<style type="text/css">
!---
A {text-decoration:none}
-->
</style>

<title>Welcome to UShuttle</title>

<script src="http://maps.google.com/maps?file=api&amp;v=2&amp;key=ABQIAAAA43Rl2mBpb6EmiyMHLLvHoxT2yXp_ZAY8_ufC3CFXhHIE1NvwkxT3204dTCYR1XJ_qw_YQfJjTqubvg">
</script>

<script type="text/javascript">
var map; //the map object

//centerMAC cannot have colons in it - All MACs should except the file name
var centerMAC; //used for the ChangeCenter function - it is the MAC that we want to center on

if (GBrowserIsCompatible()) {

  <% Response.Write(MakeShuttleIconVariables()); %> 

  /**************************************************************************
  // onLoad is what I call when the page is loaded.
  // It makes the map and puts the icon there, starts the refresh and displays
  // all the tracks and centers it on center MAC
  ***************************************************************************/
  function onLoad(lastLat, lastLng) {
    if (GBrowserIsCompatible()) {

      <% Response.Write(DefineShuttleIcons()); %> 

      //Creating the icon that is the shuttle picture
      //<!--
      //var ShuttleIcon = new GIcon();
      //ShuttleIcon.image = "images/shuttle.png";
      //ShuttleIcon.iconSize = new GSize(20, 10);
      //ShuttleIcon.shadowSize = new GSize(1, 1);
      //ShuttleIcon.iconAnchor = new GPoint(1, 1);
      //ShuttleIcon.infoWindowAnchor = new GPoint(1, 1); -->

      //Making the map and setting its properties
      map = new GMap2(document.getElementById("map"));
      map.setCenter(new GLatLng(lastLat, lastLng), 14);
      map.addControl(new GSmallMapControl());
      map.setMapType(G_HYBRID_TYPE);

      //Putting the icon on the map
      //busMarker = new GMarker(map.getCenter(), ShuttleIcon);
      //map.addOverlay(busMarker);

      //Event Listener that whenever the map is moved it will put the shuttle at the center
      //TODO: I should delete this
      GEvent.addListener(map, "move", function() {
        busMarker.setPosition(map.getCenter());
      });

      //Refreshing the route every 6 seconds
      var RouteDisplayTimer = setInterval("DisplayAllLines(); ChangeCenter();", 

  } 

</script>

</head>
// Make the map display the lines currently on startup
DisplayAllLines();
ChangeCenter();
}

function ChangeCenter()
{
    // centerMAC is the MAC without colons of the bus that we want to center on
    if(centerMAC == "Overview" || centerMAC == null)
    {
        map.setCenter(new GLatLng(40.7618209, -111.835069));
    }
    else
    {
        GDownloadUrl("data/" + centerMAC + ".txt", function(data, responseCode)
        {
            // TODO: Rename the files as XML
            var xml = GXml.parse(data);
            var markers = xml.documentElement.getElementsByTagName("marker");
            if(markers.length >= 1)
            {
                // TODO: Put a setting in the web.config file for the file path here
                // alert("In changing center and changing to: " + centerMAC);
                // alert(parseFloat(markers[markers.length-1].getAttribute("lat")));
                // alert(parseFloat(markers[markers.length-1].getAttribute("lng")));
                map.setCenter(new GLatLng(parseFloat(markers[markers.length-1].getAttribute("lat")), parseFloat(markers[markers.length-1].getAttribute("lng"))));
            }
        });
    }
}

// Changes the map so the center is at the last point of the shuttle with the center Mac
// which is designated above, it does not move the shuttle icon

function DisplayAllLines()
{
    map.clearOverlays();
    <Response.Write(DrawEveryLine());%>
}

function DisplayLine(MACAddress)
{
    // This function for the given MAC will draw the history line for that unit
    GDownloadUrl("data/" + MACAddress + ".txt", function(data, responseCode)
    {
        //alert("Displaying Line" + MACAddress);
        var xml = GXml.parse(data);
        var markers = xml.documentElement.getElementsByTagName("marker");
```
var points = [];
for (var i = 0; i < markers.length; i++) {
    points.push(new GLatLng(parseFloat(markers[i].getAttribute("lat")),
                               parseFloat(markers[i].getAttribute("lng"))));
    centerLat = markers[i].getAttribute("lat");
    centerLng = markers[i].getAttribute("lng");
    //alert(centerLat);
    //alert(centerLng);
}

//Need a programmatic if here
<% Response.Write(DisplayCurrentShuttleIcon()); %> %

<% Response.Write(AddHistory()); %> %
//map.addOverlay(new GPolyline(points));

//alert("Added overlay");
}
</script>
</head>
<body style="background-color: #ffeecf" onload="onLoad(40.7618209, -111.835069)">
<form id="form1" runat="server">
<table style="width: 100%; height: 100%">
   <!-- Main top row with the links -->
   <tr style="width: 100%; height: 10px">
      <td style="width: 10%; height: 10px;">
      </td><tr style="width: 10%; height: 10px;">
         <table style="width: 100%; background-color: #ffdd88; border="0">  
            cellpadding="0" cellspacing="0">
            <tr>
               <td style="width: 6%; height: 1px;">
               </td>
               <td style="width: 7%; height: 1px;">
                  <asp:HyperLink ID="DefaultToWideScreenLink" runat="server"ForeColor="#C00000" Font-Bold="True" Font-Names="Arial" Font-Size="Smaller" NavigateUrl="WideScreen.aspx">Wide Screen</asp:HyperLink>
               </td>
               <td style="width: 4%; height: 1px;">
                  <asp:HyperLink ID="DefaultToPDALink" runat="server"ForeColor="#C00000" Font-Bold="True" Font-Names="Arial" Font-Size="Smaller" NavigateUrl="FDA.aspx">PDA</asp:HyperLink>
               </td>
            </tr>
            <tr style="width: 59%; height: 1px; align="right">
               <asp:Login ID="LogintoUSshuttle" runat="server"BackColor="#FFDD88" BorderColor="#FFDFAD" BorderPadding="4" BorderStyle="None" BorderWidth="0px" Font-Names="Verdana" Font-Size="0.8em"ForeColor="#333333" Height="1px" TextLayout="true" Width="100%" DestinationPageUrl="LoggedIn/Default.aspx">"MySQLMembershipProvider" FailureAction="RedirectToLoginPage" TitleTextStyle BackColor="#990000" Font-Bold="true" Font-Size="0.9em" ForeColor="White" />
               <InstructionTextStyle Font-Italic="true" ForeColor="#Black" />
               <TextBoxStyle Font-Size="0.8em" />
            </tr>
         </table>
      </td>
   </tr>
</table>
```

<LoginButtonStyle BackColor="#fff" BorderColor="#f00"><br />
CC9966" BorderStyle="Solid" BorderWidth="1px"
Font-Names="Verdana" Font-Size="0.8em"
ForeColor="#990000" />
</LoginButtonStyle>
</LayoutTemplate>
</asp:Login>
</asp:LayoutTemplate>
</table>

"-- The rest of the body that I change every page -->
<asp:contentplaceholder id="MainBodyPlaceholder" runat="server">
</asp:contentplaceholder>
</asp:LayoutTemplate>

"-- The bottom row with the copyright and the ability to create new users etc

<asp:HyperLink ID="HyperLink2" runat="server">
NavigateUrl="CreateNewUser.aspx" Font-Bold="True"
Font-Names="Arial" Font-Size="Smaller" Width="274px">Create New User</asp:HyperLink>
</asp:LayoutTemplate>
</table>

<asp:HyperLink ID="HyperLink1" runat="server">
NavigateUrl="Help.aspx" Font-Bold="True" Font-
Help</asp:HyperLink>
</td>
</tr>
</table>
</form>
</body>
</html>
<?xml version="1.0"?>

<configuration>

<!-- Settings for the whole application -->
<appSettings>
    <add key="LatitudeDefault" value="yes"/>
    <add key="LongitudeDefault" value="yes"/>
    <add key="SpeedDefault" value="yes"/>
    <add key="AltitudeDefault" value="no"/>
    <add key="DriverDefault" value="yes"/>
    <add key="OnTimeDefault" value="no"/>
    <add key="LastStopDefault" value="yes"/>
    <add key="NextStopDefault" value="no"/>
    <add key="LatitudeAllow" value="yes"/>
    <add key="LongitudeAllow" value="yes"/>
    <add key="SpeedAllow" value="yes"/>
    <add key="AltitudeAllow" value="yes"/>
    <add key="DriverAllow" value="yes"/>
    <add key="OnTimeAllow" value="yes"/>
    <add key="LastStopAllow" value="yes"/>
    <add key="NextStopAllow" value="no"/>
</appSettings>

<!-- SQL Connection Strings -->
<connectionStrings>
    <add name="TestConnectionString" connectionString="Data Source=(local);Initial Catalog=WsTest;User ID=sa;Password=mmw&amp;rlw565"/>
    <add name="MyLocalSqlServer" connectionString="Data Source=MyLocalSQLServer;Initial Catalog=aspnetdb;data source= (local);User ID=sa;Password=mmw&amp;rlw565"/>
    <add name="GPSDataConnectionString" connectionString="Data Source=(local);Initial Catalog=gpsData;uid=sa;pwd=mmw&amp;rlw565"/>
</connectionStrings>

<!-- Website Configuration Settings -->
<system.web>
    <!-- Turn this off for deployment -->
    <compilation debug="true"/>
    <!-- Login authentication code -->
    <authentication mode="Forms">
        <forms
            name="UShuttle"
            timeout="15"
            loginUrl="/Default.aspx"
        />
    </authentication>
    <!-- Membership Roles Provider Code -->
    <membership defaultProvider="MySqlMembershipProvider">
        <providers>
            <!-- Removes all existing connection strings -->
            <clear/>
            <add name="MySqlMembershipProvider" connectionStringName="MyLocalSqlServer" applicationName="SrProjWebSite" requiresQuestionAndAnswer="false" minRequiredPasswordLength="6" passwordAttemptWindow="10" maxInvalidPasswordAttempts="3" />
        </providers>
    </membership>
</system.web>
</configuration>
<web.config>

<requiresUniqueEmail>true</requiresUniqueEmail>
<type>System.Web.Security.SqlMembershipProvider, System.Web, Version=2.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a</type>

</providers>
</roleManager>

<roleManager enabled="true" defaultProvider="CustomizedRoleProvider">
<providers>
<add connectionStringName="MyLocalSQLServer" name="CustomizedRoleProvider" type="System.Web.Security.SqlRoleProvider" />
</providers>
</roleManager>

<!-- The <customErrors> section enables configuration of what to do if/when an unhandled error occurs during the execution of a request. Specifically, it enables developers to configure html error pages to be displayed in place of a error stack trace. -->

<customErrors mode="RemoteOnly" defaultRedirect="/GenericErrorPage.htm">
<error statusCode="403" redirect="/NoAccess.htm" />
<error statusCode="404" redirect="/FileNotFound.htm" />
</customErrors>

</system.web>
</configuration>
using System;
using System.Data;
using System.Data.SqlClient;
using System.Configuration;
using System.Collections;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;
using System.Collections.Generic;

public partial class WideScreen : System.Web.UI.Page
{
    #region Global Variables
    private string connectionString;
    List<List<string>> DataStructure; // Holds each unique route
    List<string> RouteNames;
    #endregion

    protected void Page_Load(object sender, EventArgs e)
    {
        Get3dData();
    }
    private void Get3dData()
    {
        // I want to pull the data here and put it in a 3D array
        connectionString = WebConfigurationManager.ConnectionStrings["GPSDataConnectionString"].ConnectionString;
        string RoutesQuery = "SELECT * FROM dbo.RouteStatus";
        SqlConnection RoutesConnection = new SqlConnection(connectionString);
        SqlCommand RoutesCommand = new SqlCommand(RoutesQuery, RoutesConnection);
        SqlDataAdapter RoutesAdapter = new SqlDataAdapter(RoutesCommand);
        DataSet RoutesData = new DataSet();
        RoutesConnection.Open();
        RoutesAdapter.Fill(RoutesData, "GPSData");
        // Holds each unique route
        RouteNames = new List<string> {""};
        DataStructure = new List<List<string>> {""};
        // Go through each row
        foreach (DataRow row in RoutesData.Tables["GPSData"].Rows)
        {
            string RouteName = row["routeName"].ToString();
            string RouteNumber = row["routeNumber"].ToString();
            // TODO: I need to make an indexer for this
            List<string> CurRouteInfo = new List<string> {""};
            CurRouteInfo.Add(RouteName);
            CurRouteInfo.Add(RouteNumber);
            CurRouteInfo.Add(row["Latitude"].ToString());
            CurRouteInfo.Add(row["Longitude"].ToString());
            CurRouteInfo.Add(row["speed"].ToString());
            CurRouteInfo.Add(row["altitude"].ToString());
            CurRouteInfo.Add(row["Driver"].ToString());
            CurRouteInfo.Add(row["MAC"].ToString());
            CurRouteInfo.Add(row["OnTime"].ToString());
            CurRouteInfo.Add(row["LastStop"].ToString());
            CurRouteInfo.Add(row["NextStop"].ToString());
        }
    }
}
protected string GenerateJavaScript()
{
    string JavaScript = "";
    JavaScript += "\n\nfunction ToggleHidden(r)"
    JavaScript += "\n{
    JavaScript += "\nif(document.getElementById(r).style.display == 'none'){
    JavaScript += "\n    JavaScript += "\nelse{
    JavaScript += "\nDataStructure[RouteNames.IndexOf(RouteName)].Add(CurRouteInfo);
    JavaScript += "\n}"
    JavaScript += "\nfunction ChangeArrow(img_name)"
    JavaScript += "\n{
    JavaScript += "\n\nif(document.getElementById(" + RouteNumber + ").style.display == 'none'){
    JavaScript += "\n    JavaScript += "\nelse{
    JavaScript += "\n    JavaScript += "\n//This function will toggle an arrow when it is given the image name
    JavaScript += "\n\nfunction ChangeArrow(img_name)"
    JavaScript += "\n{
    JavaScript += "\n\nif(document.getElementById(" + img_name + ").src.match('dnarrow') == "dnarrow"){
    JavaScript += "\n    JavaScript += "\nelse{
    JavaScript += "\n"}
protected string GenerateRoutesToShow()
{
    //TODO: Try and execute JavaScript here in C# that will show all the routes when I click on the color of the route
    //Output the start of the table that allows you to select routes
    string RoutesToShow = "<table style="width: 100%" id="Table1">");
    
    foreach (string RouteName in RouteNames)
    {
        //Outputs the main rows with links that show everything and the arrow to show individual routes
        RoutesToShow += "<tr align="center">
            <td onClick="ShowAllRoutes('" + RouteName + ")";
            RoutesToShow += "<a href="javascript: void(0)" onClick="ShowAllRoutes('" + RouteName + ")";
            RoutesToShow += "<a href="javascript: void(0)" onClick="DisplayLine('" + RouteName + ")";

        //Make the hidden tables that show the route numbers
        RoutesToShow += "\n<tr id="Hidden" + RouteName + ">" + RouteName + ">";
        RoutesToShow += "\n<font face="arial" size="smaller" weight="bold" >";
        for (int i = 0; i < DataStructure[RouteNames.IndexOf(RouteName)].Count; i++)
        {
            RoutesToShow += "<a href="javascript: void(0)" onClick="DisplayLine('" + RouteName + ">";
            DataStructure[RouteNames.IndexOf(RouteName)][i][7].Replace("","" + "\n<");
            RoutesToShow += DataStructure[RouteNames.IndexOf(RouteName)][i][1];
            if (i != DataStructure[RouteNames.IndexOf(RouteName)].Count - 1)
            {
                RoutesToShow += "</a>&nbsp;|&nbsp;|
            }
        }
        RoutesToShow += "</td>");
        RoutesToShow += "</tr>");
    }
}

//Append the code for the end of the table
RoutesToShow += "</table>");
return RoutesToShow;
}

protected string GenerateCurrentRouteData()
{
    //TODO: Put in the Configuration files a way to specify the data I want
    //Get the delay time and connection string from the configuration file
    string RouteInfo = ";
    
    NameValueCollection appSettings = System.Configuration.ConfigurationSettings. 
    
    AppSettings;
    Boolean LatitudeDefault = appSettings[0] == "yes" ? true : false;
    Boolean LongitudeDefault = appSettings[1] == "yes" ? true : false;
    Boolean SpeedDefault = appSettings[2] == "yes" ? true : false;
Boolean LongitudeAllow = appSettings[9] == "yes" ? true : false;
Boolean SpeedAllow = appSettings[10] == "yes" ? true : false;
Boolean DriverAllow = appSettings[12] == "yes" ? true : false;
Boolean LastStopAllow = appSettings[14] == "yes" ? true : false;
Boolean NextStopAllow = appSettings[15] == "yes" ? true : false;

// Output the route name with the arrow to the right hand side
// When I click on blue I want to see each route with it's associated info
// When I click on the downarrow I want to see a list of routes and be able to select which data is shown
// At the bottom of the field I want to see
// I want each route info table to have a slightly darker background to set it apart and maybe a border

// Start the table
RouteInfo += "\n<table>";

// Loop through each of the route colors
foreach (string RouteName in RouteNames)
{
    // Make the row that displays each color with the down arrow. Both are links
    RouteInfo += "\n<tr align="center">\n";
    RouteInfo += "\n<td style="color:#000000;font-family:Arial;font-size:Smaller;weight:bold;">\";
    RouteInfo += "\n<a href="javascript:void(0)" onClick="ToggleHidden('Hidden')\" + RouteName + "InfoRoutes');\""> + RouteName + "/</a>";
    RouteInfo += "\n</td>\n";
    RouteInfo += "\n</tr>\n";

    // This table row will contain the hidden table
    RouteInfo += "\n<tr id=Hidden" + RouteName + " InfoRoutes style="display: none;">\n";
    RouteInfo += "\n<td align="center" style="color:#000000;font-family:Arial;font-size:Smaller;font-weight:bold;">\";
    RouteInfo += "\n</td>\n";
    RouteInfo += "\n</tr>\n";

    // Loops through the Route Numbers under the Route Name
    for (int i = 0; i < DataStructure[RouteNames.IndexOf(RouteName)].Count; i++)
    {
        RouteInfo += "\n<tr><td>\";

        // Making the table
        RouteInfo += "\n<table style="font-size:smallest; background-color: #fffcc3;"\n";

        // Number route header
        RouteInfo += "\n<tr><td align="center" width="100%" colspan="2">\";
        RouteInfo += "\nNumber " + DataStructure[RouteNames.IndexOf(RouteName)][i] + ";
        RouteInfo += "\n</td><tr>\n";

        // Put in the default information
        RouteInfo += GenerateDefaultInfo(LatitudeDefault, "Latitude", 2, RouteName, i);
        RouteInfo += GenerateDefaultInfo(LongitudeDefault, "Longitude", 3, RouteName, i);
        RouteInfo += GenerateDefaultInfo(SpeedDefault, "Speed", 4, RouteName, i);
        RouteInfo += GenerateDefaultInfo(AltitudeDefault, "Altitude", 5, RouteName, i);
        RouteInfo += GenerateDefaultInfo(DriverDefault, "Driver", 6, RouteName, i);
    }

    // Close the table
    RouteInfo += "</table>\n";
}

// Close the table row
RouteInfo += "</tr>\n";

// End the table
RouteInfo += "</table>\n";}
RouteInfo += GenerateDefaultInfo(AltitudeDefault, "Altitude", 7, RouteName, i);
RouteInfo += GenerateDefaultInfo(LastStopDefault, "Last Stop", 8, RouteName, i);
RouteInfo += GenerateDefaultInfo(NextStopDefault, "Next Stop", 9, RouteName, i);

//Option to show all the data
RouteInfo += "<a href=javascript: void(0) onClick= Show All </a>";

//End the table
RouteInfo += "</table>";
ReturnInfo += "</td></tr>

//Ends the table
RouteInfo += "</table>";

return RouteInfo;

private string GenerateDefaultInfo(bool IsPrintDefault, string InfoTitle, int InfoNumber, string RouteName, int RouteNumber)
{
    string DefaultInfo = "";
    if (IsPrintDefault)
    {
        //Output the explanation to the data
        DefaultInfo += "<tr><td width="65%" align="left" style="font-size: X-small">"
        DefaultInfo += InfoTitle;
        DefaultInfo += "</td>

        //Output the data
        DefaultInfo += "<td width="35%" style="font-size:X-small">";
        if (DataStructure[RouteNames.IndexOf(RouteName)][RouteNumber][InfoNumber] == "")
        {
            DefaultInfo += "Not Obtained";
        }
        else
        {
            DefaultInfo += DataStructure[RouteNames.IndexOf(RouteName)][RouteNumber][InfoNumber];
        }
        DefaultInfo += "</td>"
    }
    return DefaultInfo;
}
<%@ Import Namespace="System.Data.SqlClient" %>
<%@ Import Namespace="System.Data" %>
<asp:Content ID="Content1" ContentPlaceHolderID="MainBodyPlaceHolder" Runat="Server">  
    <script type="text/javascript">
        <% Response.Write(GenerateJavaScript()); %>
    </script>

    <table style="width: 100%; height: 500px" border="0" cellpadding="0" cellspacing="0">
        <tr>
            <td style="width: 12%; height: 100%; background-color: #ffdd88;">
                <% Response.Write(GenerateCurrentRouteData()); %> 
            </td> <!-- First Column that displays the current route information -->
            <td style="width: 76%; height: 100%;" align="center" valign="middle">
                <div id="map" style="width: 100%; height: 100%"></div> 
            </td> <!-- Column in the middle that displays the map -->
            <td style="width: 12%; height: 100%; background-color: #ffdd88;"> 
                <% Response.Write(GenerateRoutesToShow()); %> 
            </td> <!-- Column on the right hand side where you can select the routes that you want to see -->
        
        </tr>
        </table>
</asp:Content>
using System;
using System.Data;
using System.Data.SqlClient;
using System.Configuration;
using System.Collections;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;
using System.Collections.Generic;
using System.Collections.Specialized;

public partial class PDA : System.Web.UI.Page
{
    #region Global Variables
    private string connectionString;
    List<List<string>> DataStructure;
    List<string> RouteNames;
    #endregion

    protected void Page_Load(object sender, EventArgs e)
    {
        Get3dData();
    }

    private void Get3dData()
    {
        //I want to pull the data here and put it in a 3D array
        //(connectionString = WebConfigurationManager.ConnectionStrings["GPSDataConnectionString"]).ConnectionString;
        string RoutesQuery = "SELECT * FROM dbo.RouteStatus";
        SqlConnection RoutesConnection = new SqlConnection(connectionString);
        SqlCommand RoutesCommand = new SqlCommand(RoutesQuery, RoutesConnection);
        SqlDataAdapter RoutesAdapter = new SqlDataAdapter(RoutesCommand);
        DataSet RoutesData = new DataSet();
        RoutesConnection.Open();
        RoutesAdapter.Fill(RoutesData, "GPSData");

        //Holds each unique route
        RouteNames = new List<string>();
        DataStructure = new List<List<List<string>>>();

        //Go through each row
        foreach (DataRow row in RoutesData.Tables["GPSData"].Rows)
        {
            string RouteName = row["routeName"].ToString();
            string RouteNumber = row["routeNumber"].ToString();

            //TODO: I need to make an indexer for this
            List<string> CurRouteInfo = new List<string>();
            CurRouteInfo.Add(RouteName);
            CurRouteInfo.Add(RouteNumber);
            CurRouteInfo.Add(row["Latitude"].ToString());
            CurRouteInfo.Add(row["Longitude"].ToString());
            CurRouteInfo.Add(row["speed"].ToString());
            CurRouteInfo.Add(row["altitude"].ToString());
            CurRouteInfo.Add(row["Driver"].ToString());
            CurRouteInfo.Add(row["MAC"].ToString());
            CurRouteInfo.Add(row["OnTime"].ToString());
            CurRouteInfo.Add(row["LastStop"].ToString());
            CurRouteInfo.Add(row["NextStop"].ToString());
        }
    }
}
//if it is a new route name add a list for columns for it and put the first entry in the list
if (!RouteNames.Contains(RouteName))
{
    //Add it to the list of route names already in use
    RouteNames.Add(RouteName);

    //Create a new list for columns for route numbers on the 3d data structure List<List<string>> curRouteNumber = new List<List<string>>();

    //Put the first route number's info in curRouteNumber.Add(CurRouteInfo);

    //Put it into the 3D data structure DataStructure.Add(curRouteNumber);
}
else
{
    //Put it in the right place
    DataStructure[RouteNames.IndexOf(RouteName)].Add(CurRouteInfo);
}

protected string GenerateJavaScript()
{
    string JavaScript = "";
    JavaScript += "\n\nfunction ToggleHidden(r)"
    JavaScript += "\n"
    JavaScript += "\nif(document.getElementById(r).style.display == '')"n
    JavaScript += "\n"
    JavaScript += "\ndocument.getElementById(r).style.display = 'none';"n
    JavaScript += "\n"
    JavaScript += "\nelse"
    JavaScript += "\n"
    JavaScript += "\ndocument.getElementById(r).style.display = '';"n
    JavaScript += "\n"
    JavaScript += "\nelse"
    JavaScript += "\n"
    JavaScript += "\n"r"
    JavaScript += "\n"
    JavaScript += "\nfunction ToggleHiddenInfo(RouteNumber)"
    JavaScript += "\n"
    JavaScript += "\nif(document.getElementById("Route" + RouteNumber).style.display == '')"n
    JavaScript += "\n"
    JavaScript += "\ndocument.getElementById("Route" + RouteNumber).style.display = 'none';"n
    JavaScript += "\n"
    JavaScript += "\nelse"
    JavaScript += "\n"
    JavaScript += "\n"
    JavaScript += "\ndocument.getElementById("Route" + RouteNumber).style.display = '';"n
    JavaScript += "\n"
    JavaScript += "\nelse"
    JavaScript += "\n"
    JavaScript += "\n"
//This function will toggle an arrow when it is given the image name
    JavaScript += "\n\nfunction ChangeArrow(img_name)"
    JavaScript += "\n"
    JavaScript += "\nif(document[img_name].src.match('dnarrow') == 'dnarrow')"
    JavaScript += "\n"
    JavaScript += "\ndocument[img_name].src = './images/uparrow9x5.png';"n
    JavaScript += "\n"
    JavaScript += "\n\nelse"
    JavaScript += "\n"
    JavaScript += "\ndocument[img_name].src = './images/dnarrow9x5.png';"n
    JavaScript += "\n"
    JavaScript += "\nelse"
    JavaScript += "\n"
    JavaScript += "\n"
return JavaScript;
}

protected string GenerateRoutesToShow()
{
  // TODO: Try and execute JavaScript here in C# that will show all the routes when I click on the color of the route
  // Output the start of the table that allows you to select routes
  string RoutesToShow = "<table style="width: 100\%" id="Table1">";

  foreach (string RouteName in RouteNames)
  {
    // Outputs the main rows with links that show everything and the arrow to show individual routes
    RoutesToShow += "\n<tr align="center">\n  <td><font style="color: \#000000; font-family: Arial; font-size: Smaller; font-weight: bold;">"; 
    RoutesToShow += RouteName + "</a>" + RouteName + "</td>\n</tr>";

    RoutesToShow += "\n<tr id="Hidden" + RouteName + "Routes style="display:none;\" ">
  <td><font style="color: \#000000; font-family: Arial; font-size: Smaller; font-weight: bold;">"; 
    for (int i = 0; i < DataStructure.RouteNames.IndexOf(RouteName).Count; i++)
    {
      RoutesToShow += "\n<tr id="Hidden" + RouteName + "Routes style="display:none;\" ">
    <td><font style="color: \#000000; font-family: Arial; font-size: Smaller; font-weight: bold;">"; 

    RoutesToShow += DataStructure.RouteNames.IndexOf(RouteName)[i][7].Replace(":", ",") + "\n</tr>";

    RoutesToShow += 
    if (i != DataStructure.RouteNames.IndexOf(RouteName).Count - 1)
    {
      RoutesToShow += "</a>\n  <td><font style="color: \#000000; font-family: Arial; font-size: Smaller; font-weight: bold;">"; 
    }

    RoutesToShow += 
  </td>\n</tr>";

  // Append the code for the end of the table
  RoutesToShow += "</table>";

  return RoutesToShow;
}

protected string GenerateCurrentRouteData()
{
  // TODO: Put in the Configuration files a way to specify the data I want
  // Get the delay time and connection string from the configuration file

  string RouteInfo = "";

  AppSettings appSettings = SystemConfiguration.ConfigurationSettings.AppSettings;
  Boolean LatitudeDefault = appSettings[0] == "yes" ? true : false;
  Boolean LongitudeDefault = appSettings[1] == "yes" ? true : false;
  Boolean SpeedDefault = appSettings[2] == "yes" ? true : false;
  Boolean LastStopDefault = appSettings[6] == "yes" ? true : false;
  Boolean NextStopDefault = appSettings[7] == "yes" ? true : false;
  Boolean LatitudeAllow = appSettings[8] == "yes" ? true : false;
  Boolean LatitudeAllow = appSettings[8] == "yes" ? true : false;
Boolean LongitudeAllow = appSettings[9] == "yes"? true : false;
Boolean SpeedAllow = appSettings[10] == "yes"? true : false;
Boolean AltitudeAllow = appSettings[11] == "yes"? true : false;
Boolean DriverAllow = appSettings[12] == "yes"? true : false;
Boolean OnTimeAllow = appSettings[13] == "yes"? true : false;
Boolean LastStopAllow = appSettings[14] == "yes"? true : false;
Boolean NextStopAllow = appSettings[15] == "yes"? true : false;

//Output the route name with the arrow to the right hand side
//When I click on blue I want to see each route with it's associated info
//When I click on the downarrow I want to see a list of routes and be able to 
//select which data is shown
//At the bottom of the field I want to see
//I want each route info table to have a slightly darker background to set it apart
//and maybe a border

//Start the table
RouteInfo += "\n<table">

//Loop through each of the route colors
foreach (string RouteName in RouteNames)
{
    //Make the row that displays each color with the down arrow. Both are links
    RouteInfo += "\n<tr align="center">\n";
    RouteInfo += "\n<td style="color:#000000;font-family:Arial;font-size:Smaller; font-weight:bold;">";
    RouteInfo += "\n<a href="javascript:void(0)" onclick="ToggleHidden('Hidden') + RouteName + " InfoRoutes');">" + RouteName + "</a>;\n";
    RouteInfo += "\n</td>\n";
    RouteInfo += "\n</tr>\n";

    //This table row will contain the hidden table
    RouteInfo += "\n<tr id=Hidden" + RouteName + " InfoRoutes style="display: none;">\n";
    RouteInfo += "\n<td align="center" style="color:#000000;font-family:Arial; font-size:Smaller;font-weight:bold;">";
    RouteInfo += "\n</td>\n";
    RouteInfo += "\n</tr>\n";

    //Loops through the Route Numbers under the Route Name
    for (int i = 0; i < DataStructure[RouteNames.IndexOf(RouteName)].Count; i++)
    {
        RouteInfo += "\n<tr><td>\n";

        //Making the table
        RouteInfo += "\n<table style="font-size:smallest; background-color: #ffcc33;">";

        //Number row header
        RouteInfo += "\n<tr><td align="center" width="100%" colspan="2">\n";
        RouteInfo += "\nNumber " + DataStructure[RouteNames.IndexOf(RouteName)][i] \n";
        RouteInfo += "\n</td></tr>\n";

        //Put in the default information
        RouteInfo += GenerateDefaultInfo(LatitudeDefault, , "Latitude", 2, RouteName, i);
        RouteInfo += GenerateDefaultInfo(LongitudeDefault, , "Longitude", 3, RouteName, i);
        RouteInfo += GenerateDefaultInfo(SpeedDefault, , "Speed", 4, RouteName, i);
        RouteInfo += GenerateDefaultInfo(AltitudeDefault, , "Altitude", 5, RouteName, i);
        RouteInfo += GenerateDefaultInfo(DriverDefault, , "Driver", 6, RouteName, i);

        //Next row info
    }

    //Close table
    RouteInfo += "\n</table>\n";
}";

//Close table
RouteInfo += "\n</table>\n";
    RouteInfo += GenerateDefaultInfo(AltitudeDefault, "Altitude", 7, RouteName, i);
    RouteInfo += GenerateDefaultInfo(LastStopDefault, "Last Stop", 8, RouteName, i);
    RouteInfo += GenerateDefaultInfo(NextStopDefault, "Next Stop", 9, RouteName, i);

    // Option to show all the data
    RouteInfo += "\n<tr><td align="center" width="100%" colspan="2">\n    RouteInfo += "<a href=javascript:void(0) onClick=> Show All </a>";
    RouteInfo += "\n</td></tr>";

    // End the table
    RouteInfo += "\n</table>";
    RouteInfo += "\n</td></tr>";
}

    // Ends the table
    RouteInfo += "\n</table>";
    return RouteInfo;
}

private string GenerateDefaultInfo(bool IsPrintDefault, string InfoTitle, int InfoNumber, string RouteName, int RouteNumber)
{
    string DefaultInfo = "";
    if (IsPrintDefault)
    {
        // Output the explanation to the data
        DefaultInfo += "\n<tr><td width="65%" align="left" style="font-size:X-small">\n        DefaultInfo += InfoTitle;
        DefaultInfo += "\n</td>";

        // Output the data
        DefaultInfo += "\n<td width="35%" style="font-size:X-small">\n        if (DataStructure[RouteNames.IndexOf(RouteName)][RouteNumber][InfoNumber] == "")
        {
            DefaultInfo += "Not Obtained";
        }
        else
        {
            DefaultInfo += DataStructure[RouteNames.IndexOf(RouteName)][RouteNumber][InfoNumber];
        }
        DefaultInfo += "\n</td>";
    }
    return DefaultInfo;
}
```html
<%@ Page Language="C#" MasterPageFile="~/LoggedOut.master" AutoEventWireup="true" CodeFile="PDA.aspx.cs" Inherits="PDA" Title="Untitled Page" %>
<%@ Import Namespace="System.Data.SqlClient" %>
<%@ Import Namespace="System.Data" %>

<asp:Content ID="Content1" ContentPlaceHolderID="MainBodyPlaceHolder" Runat="Server">
    <script type="text/javascript">
        <% Response.Write(GenerateJavaScript()); %>
    </script>

    <table style="width: 475px; height: 150px; border=0; cellpadding=0; cellspacing=0">
        <tr>
            <!-- First Column that displays the current route information -->
            <td style="width: 100px; height: 150px; background-color: #ffdd88;">
                <% Response.Write(GenerateCurrentRouteData()); %>
            </td>
            <!-- Column in the middle that displays the map -->
            <td style="height: 150px; width: 300px; align=center; valign=middle">
                <div id="map" style="width: 260px; height: 150px"></div>
            </td>
            <!-- Column on the right hand side where you can select the routes that you want to see -->
            <td style="width: 75px; height: 150px; background-color: #ffdd88;">
                <% Response.Write(GenerateRoutesToShow()); %>
            </td>
        </tr>
    </table>
</asp:Content>
```
using System;
using System.Data;
using System.Configuration;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;

/// <summary>
/// Summary description for RouteInfo
/// This class represents the information for a single bus and its associated route
/// </summary>
public class RouteInfo
{
    #region ...InstanceVariables...
    private String _RouteName;
    private String _RouteNumber;
    private String _MAC;
    private double _Latitude;
    private double _Longitude;
    private double _Speed;
    private double _Altitude;
    private double _OnTime;
    private String _LastStop;
    private String _NextStop;
    #endregion

    #region ... Constructors ...
    //Default Constructor
    public RouteInfo()
    {
    }
    //Constructor that will populate all the data fields
    public RouteInfo(String NewRouteName, String NewRouteNumber, String NewMAC, Double NewLatitude, Double NewLongitude, Double NewSpeed, Double NewAltitude, Double NewOnTime, String NewLastStop, String NewNextStop)
    {
        RouteName = NewRouteName;
        RouteNumber = NewRouteNumber;
        MAC = NewMAC;
        Latitude = NewLatitude;
        Longitude = NewLongitude;
        Speed = NewSpeed;
        Altitude = NewAltitude;
        OnTime = NewOnTime;
        LastStop = NewLastStop;
        NextStop = NewNextStop;
    }
    #endregion

    #region ... Class Properties ...
    public String RouteName
    {
        get { return _RouteName; }
        set { _RouteName = value; }
    }
    public String RouteNumber
    {
        get { return _RouteNumber; }
        set { _RouteNumber = value; }
    }
    #endregion
public String MAC
{
    get { return _MAC; }
    set { _MAC = value; }
}

public double Latitude
{
    get { return _Latitude; }
    set { _Latitude = value; }
}

public double Longitude
{
    get { return _Longitude; }
    set { _Longitude = value; }
}

public double Speed
{
    get { return _Speed; }
    set { _Speed = value; }
}

public double Altitude
{
    get { return _Altitude; }
    set { _Altitude = value; }
}

public double OnTime
{
    get { return _OnTime; }
    set { _OnTime = value; }
}

public String LastStop
{
    get { return _LastStop; }
    set { _LastStop = value; }
}

public String NextStop
{
    get { return _NextStop; }
    set { _NextStop = value; }
}

#endregion
/// <summary>
/// Routes will hold a list of all the routes that are in the system at a time
/// It will pull the information from the database upon creation
/// It provides commonly methods to find a route etc.
/// </summary>

public class Routes
{
    #region ... Instance Variables ...

    private List<RouteInfo> AllRoutes = new List<RouteInfo>();
    #endregion

    public Routes()
    {
        //I want to pull the data here and put it in a 3D array
        String ConnectionString = WebConfigurationManager.ConnectionStrings[
               "GPSDataConnectionString"][0].ConnectionString;
        string RoutesQuery = "SELECT * FROM dbo.RouteStatus";
        SqlConnection RoutesConnection = new SqlConnection_ConnectionString();
        SqlCommand RoutesCommand = new SqlCommand_RoutesQuery_RoutesConnection();
        SqlDataAdapter RoutesAdapter = new SqlDataAdapter_RoutesCommand_RoutesData();
        DataSet RoutesData = new DataSet();
        RoutesConnection.Open();
        RoutesAdapter.Fill(RoutesData, "GPSData");

        //Go through each row and add the Route information to the list
        foreach (DataRow row in RoutesData.Tables["GPSData"].Rows)
        {
            RouteInfo RouteToAdd = new RouteInfo(row["routeName"].ToString(), row["routeNumber"].ToString(), row["MAC"].ToString(), Double.Parse(row["Latitude"]).ToString(), Double.Parse(row["Longitude"]).ToString(), Double.Parse(row["speed"]).ToString(), Double.Parse(row["altitude"]).ToString(), Double.Parse(row["OnTime"]).ToString(), row["LastStop"].ToString(), row["NextStop"].ToString());
            AllRoutes.Add(RouteToAdd);
        }
    }

    //RouteNames will give the names or all routes in the system

    public List<String> RouteNames()
    {
        //Create a List to store the names that have been found
        List<String> RouteNames = new List<String>();
        foreach (RouteInfo CurrentRouteInfo in AllRoutes)
        {
            if (RouteNames.Contains(CurrentRouteInfo.RouteName))
            {
                RouteNames.Add(CurrentRouteInfo.RouteName);
            }
        }
        return RouteNames;
    }
}
public List<String> RouteNumbers(String RouteName)
{
    // Create a list to store the names that have been found
    List<String> RouteNumbers = new List<String>();
    foreach (RouteInfo CurrentRouteInfo in AllRoutes)
    {
        if (CurrentRouteInfo.RouteName.Equals(RouteName))
        {
            RouteNumbers.Add(CurrentRouteInfo.RouteNumber);
        }
    }
    return RouteNumbers;
}

public List<String> MACsOnRoute(String RouteName)
{
    // Create a list to store the names that have been found
    List<String> MACs = new List<String>();
    foreach (RouteInfo CurrentRouteInfo in AllRoutes)
    {
        if (CurrentRouteInfo.RouteName.Equals(RouteName))
        {
            MACs.Add(CurrentRouteInfo.MAC);
        }
    }
    return MACs;
}

public String RouteNumberFromMAC(String MAC)
{
    foreach (RouteInfo CurrentRouteInfo in AllRoutes)
    {
        if (CurrentRouteInfo.MAC.Equals(MAC))
        {
            return CurrentRouteInfo.RouteNumber;
        }
    }
    return null;
}

public String GetSingleInfo(String RouteName, String RouteNumber, String InfoRequested)
{
    foreach (RouteInfo CurrentRouteInfo in AllRoutes)
    {
        if (CurrentRouteInfo.RouteName.Equals(RouteName) && CurrentRouteInfo.RouteNumber.Equals(RouteNumber))
        {
            switch (InfoRequested)
            
            
        }
    }
}
case "Latitude":
    return CurrentRouteInfo.Latitude.ToString();
    break;
case "Longitude":
    return CurrentRouteInfo.Longitude.ToString();
    break;
case "MAC":
    return CurrentRouteInfo.MAC;
    break;
case "Speed":
    return CurrentRouteInfo.Speed.ToString();
    break;
case "Altitude":
    return CurrentRouteInfo.Altitude.ToString();
    break;
case "OnTime":
    return CurrentRouteInfo.OnTime.ToString();
    break;
case "LastStop":
    return CurrentRouteInfo.LastStop;
    break;
case "NextStop":
    return CurrentRouteInfo.NextStop;
    break;
default:
    break;
}

return null;

//**************************************************************
// GetSingleInfo will get a specific piece of information from the
// shuttle with the specified MAC address - overloaded method
//*****************************************************************
public String GetSingleInfo(String MAC, String InfoRequested)
{
    foreach (RouteInfo CurrentRouteInfo in AllRoutes)
    {
        if (CurrentRouteInfo.MAC.Equals(MAC))
        {
            switch (InfoRequested)
            {
                case "Latitude":
                    return CurrentRouteInfo.Latitude.ToString();
                    break;
                case "Longitude":
                    return CurrentRouteInfo.Longitude.ToString();
                    break;
                case "MAC":
                    return CurrentRouteInfo.MAC;
                    break;
                case "Speed":
                    return CurrentRouteInfo.Speed.ToString();
                    break;
                case "Altitude":
                    return CurrentRouteInfo.Altitude.ToString();
                    break;
                case "OnTime":
                    return CurrentRouteInfo.OnTime.ToString();
                    break;
                case "LastStop":
                    return CurrentRouteInfo.LastStop;
                    break;
                case "NextStop":
                    return CurrentRouteInfo.NextStop;
                    break;
            }
        }
    }

    return null;
}
return CurrentRouteInfo.NextStop;
break;
default:
    break;
}
}  
return null;
}

/**************************************************************************
* GetAllMACs will return the MAC addresses of all the routes
* in a List<String> data type
* **************************************************************************/
public List<String> GetAllMACs()
{
    List<String> AllMACs = new List<String>();
    foreach (RouteInfo CurrentRoute in AllRoutes)
    {
        AllMACs.Add(CurrentRoute.MAC);
    }
    return AllMACs;
}
using System;
using System.Threading;

namespace EventScheduler
{
    // delegate for OnTrigger() event
    public delegate void Invoke(string scheduleName);

    // enumeration for schedule types
    public enum ScheduleType { ONETIME, INTERVAL, DAILY, WEEKLY, MONTHLY };

    // base class for all schedules
    abstract public class Schedule : IComparable
    {
        public event Invoke OnTrigger;
        protected string m_name; // name of the schedule
        protected ScheduleType m_type; // type of schedule
        protected bool m_active; // is schedule active ?

        protected DateTime m_startTime; // time the schedule starts
        protected DateTime m_stopTime; // ending time for the schedule
        protected DateTime m_nextTime; // time when this schedule is invoked next, used by scheduler

        // m_fromTime and m_toTime are used to defined a time range during the day
        // between which the schedule can run.
        // This is useful to define a range of working hours during which a schedule can run
        protected TimeSpan m_fromTime;
        protected TimeSpan m_toTime;

        // Array containing the 7 weekdays and their status
        // Using DayOfWeek enumeration for index of this array
        // By default Sunday and Saturday are non-working days
        bool[] m_workingWeekDays = new bool[]{false, true, true, true, true, true, true};

        // time interval in seconds used by schedules like IntervalSchedule
        long m_interval = 0;

        // Accessor for type of schedule
        public ScheduleType Type
        {
            get { return m_type; }
        }

        // Accessor for name of schedule
        // Name is set in constructor only and cannot be changed
        public string Name
        {
            get { return m_name; }
        }

        public bool Active
        {
            get { return m_active; }
            set { m_active = value; }
        }

        // check if no week days are active
        protected bool NoFreeWeekDay()
        {
            bool check = false;
            for (int index=0; index<7; check = check|m_workingWeekDays[index], index++);
            return check;
        }

        // Setting the status of a week day
public void SetWeekDay(DayOfWeek day, bool On)
{
    m_workingWeekDays[(int)day] = On;
    Active = true; // assuming

    // Make schedule inactive if all weekdays are inactive
    // If a schedule is not using the weekdays the array should not be touched
    if (NoFreeWeekDay())
        Active = false;
}

// Return if the week day is set active
public bool WeekDayActive(DayOfWeek day)
{
    return m_workingWeekDays[(int)day];
}

// Method which will return when the Schedule has to be invoked next
// This method is used by Scheduler for sorting Schedule objects in the list
public DateTime NextInvokeTime
{
    get { return m_nextTime; }
}

// Accessor for m_startTime
public DateTime StartTime
{
    get { return m_startTime; }
    set
    {
        // start time can only be in future
        if (value.CompareTo(DateTime.Now) <= 0)
            throw new SchedulerException("Start Time should be in future");

        m_startTime = value;
    }
}

// Accessor for m_interval in seconds
// I am using a lower limit of 30 seconds
public long Interval
{
    get { return m_interval; }
    set
    {
        if (value < 30)
            throw new SchedulerException("Interval cannot be less than 60 seconds");

        m_interval = value;
    }
}

// Constructor
public Schedule(string name, DateTime startTime, ScheduleType type)
{
    StartTime = startTime;
    m_nextTime = startTime;
    m_type = type;
    m_name = name;
}

// Sets the next time this Schedule is kicked off and kicks off events on
// a seperate thread, freeing the Scheduler to continue
public void TriggerEvents()
{
    CalculateNextInvokeTime(); // first set next invoke time to continue with rescheduling
    ThreadStart ts = new ThreadStart(KickOffEvents);
}
Thread t = new Thread(ts);
t.Start();
}

// Implementation of ThreadStart delegate.
// Used by Scheduler to kick off events on a separate thread
private void KickOffEvents()
{
    if (OnTrigger != null)
        OnTrigger(Name);
}

// To be implemented by specific schedule objects when to invoke the schedule next
internal abstract void CalculateNextInvokeTime();

// check to see if the Schedule can be invoked on the week day it is next scheduled
protected bool CanInvokeOnNextWeekDay()
{
    return m_workingWeekDays[(int)m_nextTime.DayOfWeek];
}

// Check to see if the next time calculated is within the time range
// given by m_fromTime and m_toTime
// The ranges can be during a day, for eg. 9 AM to 6 PM on same day
// or overlapping 2 different days like 10 PM to 5 AM (i.e over the night)
protected bool IsInvokeTimeInRange()
{
    if (m_fromTime < m_toTime) // eg. like 9 AM to 6 PM
        return (m_nextTime.TimeOfDay > m_fromTime && m_nextTime.TimeOfDay < m_toTime);
    else // eg. like 10 PM to 5 AM
        return (m_nextTime.TimeOfDay > m_toTime && m_nextTime.TimeOfDay < m_fromTime);
}

// IComparable interface implementation is used to sort the array of Schedules
// by the Scheduler
public int CompareTo(object obj)
{
    if (obj is Schedule)
    {
        return m_nextTime.CompareTo(((Schedule)obj).m_nextTime);
    }
    throw new Exception("Not a Schedule object");
}
using System;
using System.Threading;
using System.Collections;

namespace EventScheduler
{
    public enum SchedulerEventType { CREATED, DELETED, INVOKED, }

    public delegate void SchedulerEventDelegate(SchedulerEventType type, string scheduleName);

    // This is the main class which will maintain the list of Schedules
    // and also manage them, like rescheduling, deleting schedules etc.
    public sealed class Scheduler
    {
        // Event raised when for any event inside the scheduler
        public static event SchedulerEventDelegate OnSchedulerEvent;

        // next event which needs to be kicked off,
        // this is set when a new Schedule is added or after invoking a Schedule
        public static Schedule m_nextSchedule = null;
        public static ArrayList m_schedulesList = new ArrayList(); // list of schedules
        public static Timer m_timer = new Timer(new TimerCallback(DispatchEvents), null, Timeout.Infinite, Timeout.Infinite);

        // Get schedule at a particular index in the array list
        public static Schedule GetScheduleAt(int index) { if (index < 0 || index >= m_schedulesList.Count) return null; return (Schedule)m_schedulesList[index]; }

        // Number of schedules in the list
        public static int Count() { return m_schedulesList.Count; }

        // Indexer to access a Schedule object by name
        public static Schedule GetSchedule(string scheduleName) { for (int index = 0; index < m_schedulesList.Count; index++) if (((Schedule)m_schedulesList[index]).Name == scheduleName) return (Schedule)m_schedulesList[index]; return null; }

        // call back for the timer function
        static void DispatchEvents(object obj) // obj ignored { if (m_nextSchedule == null) return; m_nextSchedule.TriggerEvents(); // make this happen on a thread to let this thread continue if (m_nextSchedule.Type == ScheduleType.ONETIME) { RemoveSchedule(m_nextSchedule); // remove the schedule from the list } else { if (OnSchedulerEvent != null) OnSchedulerEvent(SchedulerEventType.INVOKED, m_nextSchedule.Name); }
    }
m_schedulesList.Sort();
    SetNextEventTime();
}

// method to set the time when the timer should wake up to invoke the next schedule
static void SetNextEventTime()
{
    if (m_schedulesList.Count == 0)
    {
        m_timer.Change(Timeout.Infinite, Timeout.Infinite); // this will put the timer to sleep
        return;
    }
    m_nextSchedule = (Schedule)m_schedulesList[0];
    TimeSpan ts = m_nextSchedule.NextInvokeTime.Subtract(DateTime.Now);
    if (ts < TimeSpan.Zero)
        ts = TimeSpan.Zero; // cannot be negative!
    m_timer.Change((int)ts.TotalMilliseconds, Timeout.Infinite); // invoke after the timespan
}

// add a new schedule
public static void AddSchedule(Schedule s)
{
    if (GetSchedule(s.Name) != null)
        throw new SchedulerException("Schedule with the same name already exists");
    m_schedulesList.Add(s);
    m_schedulesList.Sort();
    // adjust the next event time if schedule is added at the top of the list
    if (m_schedulesList[0] == s)
        SetNextEventTime();
    if (OnSchedulerEvent != null)
        OnSchedulerEvent(SchedulerEventType.CREATED, s.Name);
}

// remove a schedule object from the list
public static void RemoveSchedule(Schedule s)
{
    m_schedulesList.Remove(s);
    SetNextEventTime();
    if (OnSchedulerEvent != null)
        OnSchedulerEvent(SchedulerEventType.DELETED, s.Name);
}

// remove schedule by name
public static void RemoveSchedule(string name)
{
    RemoveSchedule(GetSchedule(name));
}
using System;

namespace EventScheduler
{
    /// <summary>
    /// Summary description for SchedulerException.
    /// </summary>
    public class SchedulerException : Exception
    {
        public SchedulerException(string msg) : base(msg)
        {
        }
    }
}
using System;

namespace EventScheduler
{
    // OneTimeSchedule is used to schedule an event to run only once
    // Used by specific tasks to check self status
    public class OneTimeSchedule : Schedule
    {
        public OneTimeSchedule(string name, DateTime startTime)
            : base(name, startTime, ScheduleType.ONETIME)
        {
        }

        internal override void CalculateNextInvokeTime()
        {
            // it does not matter, since this is a one time schedule
            m_nextTime = DateTime.MaxValue;
        }
    }

    // IntervalSchedule is used to schedule an event to be invoked at regular intervals
    // the interval is specified in seconds. Useful mainly in checking status of threads
    // and connections. Use an interval of 60 hours for an hourly schedule
    public class IntervalSchedule : Schedule
    {
        public IntervalSchedule(string name, DateTime startTime, int secs,
            TimeSpan fromTime, TimeSpan toTime) // time range for the day
            : base(name, startTime, ScheduleType.INTERVAL)
        {
            m_fromTime = fromTime;
            m_toTime = toTime;
            Interval = secs;
        }

        internal override void CalculateNextInvokeTime()
        {
            // add the interval of m_seconds
            m_nextTime = m_nextTime.AddSeconds(Interval);

            // if next invoke time is not within the time range, then set it to next start time
            if (!IsInvokeTimeInRange())
            {
                if (m_nextTime.TimeOfDay < m_fromTime)
                    m_nextTime.AddSeconds(m_fromTime.Seconds - m_nextTime.TimeOfDay.Seconds);
                else
                    m_nextTime.AddSeconds((24 * 3600) - m_nextTime.TimeOfDay.Seconds + m_fromTime.Seconds);
            }

            // check to see if the next invoke time is on a working day
            while (!CanInvokeOnNextWeekDay())
            {
                m_nextTime = m_nextTime.AddDays(1); // start checking on the next day
            }
        }
    }

    // Daily schedule is used set off to the event every day
    // Mainly useful in maintenance, recovery, logging and report generation
    // Restrictions can be imposed on the week days on which to run the schedule
    public class DailySchedule : Schedule
    {
        public DailySchedule(string name, DateTime startTime)
            : base(name, startTime, ScheduleType.DAILY)
        {
        }

        internal override void CalculateNextInvokeTime()
        {
        }
    }
}
// add a day, and check for any weekday restrictions and keep adding a day
m_nextTime = m_nextTime.AddDays(1);
while (!CanInvokeOnNextWeekDay())
    m_nextTime = m_nextTime.AddDays(1);
}

// Weekly schedules, useful generally in lazy maintenance jobs and
// restarting services and others major jobs
public class WeeklySchedule : Schedule
{
    public WeeklySchedule(string name, DateTime startTime) : base(name, startTime, ScheduleType.WEEKLY)
    {
    }
    // add a week (or 7 days) to the date
    internal override void CalculateNextInvokeTime()
    {
        m_nextTime = m_nextTime.AddDays(7);
    }
}

// Monthly schedule - used to kick off an event every month on the same day as
// and also at the same hour and minute as given in start time
public class MonthlySchedule : Schedule
{
    public MonthlySchedule(string name, DateTime startTime) : base(name, startTime, ScheduleType.MONTHLY)
    {
    }
    // add a month to the present time
    internal override void CalculateNextInvokeTime()
    {
        m_nextTime = m_nextTime.AddMonths(1);
    }
}
<?xml version="1.0"?>
<configuration>

<appSettings/>

<connectionStrings/>

<system.web>

<!-- Denies users that haven't been authenticated from accessing this page -->
<authorization>
<deny users="?"/>
<allow users="*"/>
</authorization>

</system.web>

</configuration>
JavaScript += "\nelse";
<script type="text/javascript">
    <% Response.Write(GenerateJavaScript()); %>
</script>

<table style="width: 100%; height: 500px" border="0" cellpadding="0" cellspacing="0">
    <tr>
        <!-- First Column that displays the current route information -->
        <td style="width: 12%; height: 100%; background-color: #ffdd88;">
            <% Response.Write(GenerateCurrentRouteData()); %>
        </td>
    </tr>
    <!-- Column in the middle that displays the map -->
    <td style="width: 76%; height: 100%;" align="center" valign="middle">
        <div id="map" style="width: 100%; height: 100%"></div>
    </td>
    <!-- Column on the right hand side where you can select the routes that you want to see -->
    <td style="width: 12%; height: 100%; background-color: #ffdd88;">
        <% Response.Write(GenerateRoutesToShow()); %>
    </td>
</tr>
</table>