Repurposing a Roomba: Evaluating and training behavior in a simple agent

Donald Samuel Abbott-McCune
Virginia Commonwealth University

Follow this and additional works at: https://scholarscompass.vcu.edu/etd

Part of the Computer Sciences Commons

© The Author

Downloaded from
https://scholarscompass.vcu.edu/etd/1077

This Thesis is brought to you for free and open access by the Graduate School at VCU Scholars Compass. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of VCU Scholars Compass. For more information, please contact libcompass@vcu.edu.
© Donald Samuel Abbott-McCune, 2007

All Rights Reserved
Repurposing a Roomba: Evaluating and training behavior in a simple agent

A thesis submitted in partial fulfillment of the requirements for the degree Master of Science at Virginia Commonwealth University

By

Donald Samuel Abbott-McCune

Director: Dr. David Primeaux
Associate Professor, Department of Computer Science

Virginia Commonwealth University
Richmond, Virginia
May 2007
Acknowledgements

I would like to thank Dr. David Primeaux for his mentorship, advising, and for introducing me to new horizons of education. I would also express much thanks to the professors that guided me and taught me valuable lessons in Computer Science and to Deanna Pace for helping with scheduling and general counsel. Much appreciation goes to fellow graduate students whose bond and idea sharing helped create an ideal learning environment.

To my wife Valerie and my daughter Rebecca, I cannot express the gratitude I have for your support while many hours were exhaustively used to research. To Ralph and Janie, my in-laws who treat me like their own son, your support during this time has been immeasurable.

I finally would like to express appreciation to all the Soldiers in the U.S. Army who have pushed me and recommended for teaching at West Point Military Academy. I only hope to stand on the shoulders of the giants that have come before and serve admirably.
# Table of Contents

List of Figures ................................................................................................................................. vii

Chapter 1: Introduction ......................................................................................................................... 10

Roomba, what is it: ................................................................................................................................. 10

The SCI how it works: ............................................................................................................................... 11

Chapter 2: Research Hardware ......................................................................................................... 15

Interface to the Roomba: ......................................................................................................................... 15

Research Platform: ................................................................................................................................. 17

Chapter 3: Research Software ....................................................................................................... 18

Programming Platform: ......................................................................................................................... 18

SCI Modes: ........................................................................................................................................... 21

Roomba Drive Control: ......................................................................................................................... 23

Roomba sensor data: ............................................................................................................................... 25

Virtual Wall: .......................................................................................................................................... 27

Bump Sensors: ..................................................................................................................................... 28

Chapter 4: The Roomba task: ........................................................................................................ 30

Experiment 1: ..................................................................................................................................... 30

Experiment 1 enhanced ......................................................................................................................... 33

Results ................................................................................................................................................. 34
List of References

VITA
List of Figures

Figure 1: iRobot Roomba Scheduler.................................................................................................................. 10
Figure 2: SCI pin outs........................................................................................................................................... 12
Figure 3: Standard 9 pin serial port.................................................................................................................... 13
Figure 4: Serial Data Packet .............................................................................................................................. 13
Figure 5: Todbot original Serial Tether ................................................................................................................ 15
Figure 6: Robo Dynamics Rootooth adaptor ...................................................................................................... 16
Figure 7: Bluetooth interface showing the serial ports ...................................................................................... 17
Figure 8: Version 2.2 Roomba Control Program: Musical tab.............................................................................. 18
Figure 9: Roomba Control Program Version 2.2: Serial Port Setup Tab ............................................................. 19
Figure 10: Control Program Version 2.2: Drive Control Tab............................................................................... 19
Figure 11: Control Program Version 2.2: Free Willy Tab................................................................................... 20
Figure 12: Roomba-Term Start Page (A) Packet Page (B).................................................................................. 21
Figure 13: OPCODE for the Roomba .................................................................................................................. 22
Figure 14: SCI Bumps and Wheel Drops Byte Breakout .................................................................................... 27
Figure 15: Virtual Wall beam representation (A) Roomba sensor (B)................................................................. 28
Figure 16: Location of Roomba Bump Sensors................................................................................................... 28
Figure 17: Expected results (A) and actual results (B) ....................................................................................... 31
Figure 18: Previous code showing packets sent to and received from the Roomba.............................. 32
Figure 19: Actual Driving of Bud using Algorithm on Medium Pile................................................................. 33
Figure 20: Roomba Tire....................................................................................................................................... 36
Figure 21: Representation of Drive Belt on the Roomba Wheel........................................................................ 36
Figure 22: Roomba making contact with wall Valid (A) Roomba over Veering Invalid (B)...................... 39
Figure 23: Bud traversing 50 Foot Hallway ................................................................................. 44
Figure 24: Track B the L-shape with a virtual wall as the goal........................................................ 45
Figure 25: Bud on Hole 11 maneuvering around the incline............................................................ 47
Figure 26: Bud on hole 2 manuvering around the u-turn................................................................. 47
Figure 27: Bud on Hole 2 path........................................................................................................ 49
Abstract

Repurposing a Roomba: Evaluating and training behavior in a simple agent

By Donald Samuel Abbott-McCune

A thesis submitted in partial fulfillment of the requirements for the degree Master of Science at Virginia Commonwealth University

Virginia Commonwealth University, 2007

Director: Dr. David Primeaux, Associate Professor, Department of Computer Science

Recent attempts to reprogram a Roomba to be used as a simple agent has led to interesting behavior. Observation has shown that the behavior of the Roomba is not only dependant on the percepts of the Roomba, but also relies heavily on the uncontrollable environmental conditions that the Roomba is placed in. Ultimately this makes the Roomba a great platform to test and teach aspects of artificial intelligence. This paper will show how most of the tested environmental conditions are mitigated by a learning agent that will adjust behavior dependant on the percepts that are received.
Chapter 1: Introduction

Roomba, what is it:

The Roomba (Figure 1) is a vacuum designed by iRobot to provide automated cleaning services for those who like new gadgets and simply do not want to vacuum themselves. The vacuum is available off the shelf for base model US $199.99 to US $329.99 for the model used in this research. The Roomba utilizes random walks and a combination of sensors to detect dirt and sensors to detect cliffs or walls.

![Figure 1: iRobot Roomba Scheduler](image)

The designers at iRobot integrated a Serial Command Interface (SCI) (1) to monitor the internal components of the vacuum cleaner. iRobot wanted to open the little robot up to developers and innovators to provide an inexpensive robot platform to learn on. This led them to
embedding software into the Roomba beginning October 2005, to allow interactions with the robot without erasing the original programming to allow vacuuming functionality. They also developed a device called the OSMO (2) for earlier model Roombas to give them access to the open interface. For this research the Roomba Scheduler was chosen as the test platform because it had the most sensor functionality and remote scheduling available at the time. Although since the inception of this research and the explosion of hackers and educator interest, iRobot has introduced the iRobot® Create™ an ungraded version which allows much more control over the robot.

“The iRobot Create Programmable Robot is a durable, reliable, fully assembled programmable robot with out-of-the-box operation. 10 built-in demos and 32 built-in sensors allow you to control, create and experiment with robotics. An open cargo bay and 25-pin expansion port allow you to add your own sensors, grippers, wireless connections, computers or other hardware.” (3)

The Create Roomba can be accessorized with a module which has a breakout board, a microcontroller, and additional serial inputs.

**The SCI how it works:**

The SCI is hidden beneath a snap-away cover right above the power plug. (Figure 1) The Roomba communicates with a computer via a serial connection. The pin outs for the SCI are shown in figure 2. The SCI connector is the primary communications interface between the Roomba and an external device such as the PC. Anything connected to the SCI will draw power from the Roomba’s battery and will shorten the charge capacity specified by the manufacturer.
Also great care should be taken when connecting two dissimilar devices to safeguard the voltages that can be transmitted between them.

Table: SCI pin outs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Vpwr</td>
<td>Roomba battery + (unregulated)</td>
</tr>
<tr>
<td>2</td>
<td>Vpwr</td>
<td>Roomba battery + (unregulated)</td>
</tr>
<tr>
<td>3</td>
<td>RXD</td>
<td>0 – 5V Serial input to Roomba</td>
</tr>
<tr>
<td>4</td>
<td>TXD</td>
<td>0 – 5V Serial output from Roomba</td>
</tr>
<tr>
<td>5</td>
<td>DD</td>
<td>Device Detect input (active low) – used to wake up Roomba from sleep</td>
</tr>
<tr>
<td>6</td>
<td>GND</td>
<td>Roomba battery ground</td>
</tr>
<tr>
<td>7</td>
<td>GND</td>
<td>Roomba battery ground</td>
</tr>
</tbody>
</table>

Figure 2: SCI pin outs

iRobot recommended using a MAX232 chip for the communication with the PC (1), as it uses RS232 voltage level and would protect the PC and Roomba from overvoltages. The RS232 is the standard 9-pin D connector on most desktop computers, although on newer laptop computers serial ports have mostly been eliminated in favor of USB port which can transfer data at a significantly faster rate. (4) The serial port as shown in figure 3 uses asynchronous communications one bit at a time.
Figure 3: Standard 9 pin serial port

The data packet is represented in figure 4, and shows the start and stop bit used to signal the information flow.

<table>
<thead>
<tr>
<th>PIN</th>
<th>PURPOSE</th>
<th>SIGNAL NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pin 1</td>
<td>Data Carrier Detect</td>
<td>DCD</td>
</tr>
<tr>
<td>Pin 2</td>
<td>Received Data</td>
<td>RxData</td>
</tr>
<tr>
<td>Pin 3</td>
<td>Transmitted Data</td>
<td>TxData</td>
</tr>
<tr>
<td>Pin 4</td>
<td>Data Terminal Ready</td>
<td>DTR</td>
</tr>
<tr>
<td>Pin 5</td>
<td>Signal Ground</td>
<td>Gnd</td>
</tr>
<tr>
<td>Pin 6</td>
<td>Data Set Ready</td>
<td>DSR</td>
</tr>
<tr>
<td>Pin 7</td>
<td>Request To Send</td>
<td>RTS</td>
</tr>
<tr>
<td>Pin 8</td>
<td>Clear To Send</td>
<td>CTS</td>
</tr>
<tr>
<td>Pin 9</td>
<td>Ring Indicator</td>
<td>RI</td>
</tr>
</tbody>
</table>

Figure 4: Serial Data Packet

While serial communication can use a variety of baud rates, the Roomba is limited to two different baud rates, 57600 which is the default and 19200 which is a compatibility mode for other microprocessors that cannot operate at the higher rate. The baud rate refers to the number of symbols that can be transferred in one second. The serial device is also capable of being configured to use a variable number of bits. For example, a device could only need 5 bits to send
its information, thus the data bits would be 5. The number of data bits that the Roomba uses is 8, which corresponds to the number of bits in a byte. This seems to be an arbitrary number, but allows room for additional data to be sent if the Roomba’s sensors are updated. Parity is the error checking that is used to check for transmission errors, but is not utilized in the Roomba. Stop Bits are used to allow the devices on a serial connection to resynchronize, and the Roomba uses 1 Stop Bit. Finally, the flow control on the serial port is another way to help synchronize two devices on the serial connection; however the Roomba does not utilize this feature.

Programming for this project was written in C# and the following code snippet shows how to implement the serial ports in C#.

```csharp
// SerialPort class
using System.IO.Ports;

private static void UseSerialPort()
{
    // Instantiate the communications port
    SerialPort myport = new SerialPort("COM8", 57600, Parity.None, 8, StopBits.One);

    // Open the port for communications
    myport.Open();

    // Write a string
    myport.Write("Hello World");

    // Close the port
    myport.Close();
}
Chapter 2: Research Hardware

Interface to the Roomba:

Through the progression of my research the actual interface has morphed a few times but in the end became a Bluetooth serial interface. The first iteration of the interface was completed with great assistance from Tod Kurt also known as Todbot and his wiring diagram. (5)

![Diagram of the interface](image)

Figure 5: Todbot original Serial Tether

The interface was connected to the Roomba via a Mac mini 8-pin cable, but this proved problematic with some cables as they shorted out the SCI interface, thus resulting in the replacement of one of the two experimental Roombas. Looking for an alternative for the
communications device lead me to the RooTooth developed by Robo Dynamics. (Figure 6)
This is a Bluetooth device, which eliminated the chance of burning out the SCI port again. The
RooTooth can communicate with any Bluetooth enabled device. Robo Dynamics became a
leading provider for iRobot interface devices. They also provide a USB roostick which allows
interfacing the Roomba with the high speed MAC cable, but still limits the range of the Roomba
to the length of the cable or requires attaching the laptop to the Roomba itself. The USB device
according to research conducted at Harvey Mudd College (6) is much faster in polling than the
Bluetooth, but with the limitations on the cable length and after performing several tests, the
Bluetooth handles the polling adequately. None of computers used for testing were Bluetooth

![Figure 6: Robo Dynamics Rootooth adaptor](image)

enabled, so I used a USB Bluetooth adaptor from AZIO. The installation of the AZIO adaptor
was fairly intuitive, but because the documentation on the RooTooth was not clear on the
passkey, communication was impossible until that was established. The default passkey was
simply “default” as published in later manuals. (7) Once the installation of the driver was
complete, the device uses the windows Bluetooth control panel to configure new devices.
Once the passkey is correct the connector enables the two serial ports as shown in figure 7.

**Research Platform:**

For this research two separate Roomba schedulers were purchased. The scheduler was chosen because of all the accessories that are included which allowed for testing of all the capabilities of the Roomba. In addition, it was determined that the scheduler is one of the Roomba’s features that differentiate between all three bump sensors. This is determined by checking for an “E” in the serial number. (1) Almost exclusively all the programming was tested on Roomba 1, which was later affectionately called Bud and marked with a single line on the virtual wall sensor. The second Roomba was nicknamed Lou.
Chapter 3: Research Software

Programming Platform:

The first programming language chosen to attempt communications with the Roomba was Java. Thanks are due Todbot, since he provided an application programming interface (API) for controlling the Roomba. However, there were several issues with his API. He had written it on a MAC and although Java claims portability, there were conflicts with updates to the Java API on Windows platforms and serial communications. When Java upgraded to Java 6, the serial compatibility in a Windows PC was omitted. For this research, therefore, the decision was to use C# as the programming language of choice because it supports serial communications. Several iterations of the program interface were compiled and used to test full functionality of the Roomba. One iteration of the program had an entertaining virtue, but did nothing for the substance of the driving of the Roomba was the musical interface. (Figure 8)

![Figure 8: Version 2.2 Roomba Control Program: Musical tab]
Although this was not implemented for the final version, it allows the user to compose songs on the Roomba and play them back. The functionality of the program which allows the choice of the serial port details is shown in Figure 9.

![Figure 9: Roomba Control Program Version 2.2: Serial Port Setup Tab](image)

Once the serial port is started, the drive control page can be used to set the mode of operation, drive the Roomba, and adjust the color of the lights on the Roomba. (Figure 10)

![Figure 10: Control Program Version 2.2: Drive Control Tab](image)
Finally the direct serial port interface, “the Free Willy tab,” was created to test the exact command needed to check functionality. (Figure 11) This feature was effective in testing all the features of the Roomba and in checking the sensors. All the data from the sensors would be returned in HEX representation and then decoded for the exact meaning. Appendix 1 was used for quick referencing.

![Figure 11: Control Program Version 2.2: Free Willy Tab](image)

After several attempts to use this program on several disjointed location and platforms, my laptop, my home computer, and the school provided computer, there was an issue with the COM port and the USB Bluetooth device. The program would poll the system for the COM ports attached to the computer, but after several changes in locations, simply by unplugging the USB device and putting it into another computer, it would increment the number of COM ports. The second issue arose when erratic symbols would be appended to the COM port. For example, instead of COM port COM3 being returned, COM3i or some other appended character would prevent the Roombas communication. While searching for a fix for this anomaly, I ran across a C# framework written by Kevin Gabbert (8) that had much of the backend already implemented. This framework became the base program for the rest of the experimentation. The start window
(Figure 12A) allows you to set the COM port and start the polling, which automatically polls the sensor data and populates the sensor and packet pages. (Figure 12B)

![Roomba-Term Start Page](image1.png) ![Roomba Sensor Packet](image2.png)

**Figure 12: Roomba-Term Start Page (A) Packet Page (B)**

**SCI Modes:**

The Roomba has four different operating modes: off, passive, safe, and full. The default mode is passive mode. This mode allows the computer to poll the sensor data, execute the standard buttons on the Roomba, define a song, or force the Roomba to go to the charging dock. Safe mode is the operating mode that will automatically stop if a cliff is detected, a wheel drops, or if the Roomba is plugged in. The Roomba reverts to passive mode if any of the triggers happens. Full mode shuts down all of the safety sensors and allows access to all the actuators on the Roomba. The minimum time between shifting from one mode to another is 20ms. The Operational Code (OPCODE) 128 is the first thing that must be sent to the SCI port to activate the communications. This sets up the Roomba for serial communications with an external device.
such as the computer or a microcontroller. The Roomba is then ready to give sensor data or receive commands. The quick reference guide for all OPCODES is in Figure 13.

<table>
<thead>
<tr>
<th>Command</th>
<th>Opcode</th>
<th>Data Byte 1</th>
<th>Data Byte 2</th>
<th>Data Byte 3</th>
<th>Data Byte 4</th>
<th>Etc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>128</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Baud</td>
<td>129</td>
<td>Baud Code</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(0 – 11)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Control</td>
<td>130</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Safe</td>
<td>131</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Full</td>
<td>132</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Power</td>
<td>133</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Spot</td>
<td>134</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Clean</td>
<td>135</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Max</td>
<td>136</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Drive</td>
<td>137</td>
<td>Velocity</td>
<td></td>
<td>Radius</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Motors</td>
<td>138</td>
<td>Motor Bits</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(0 – 7)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Leds</td>
<td>139</td>
<td>Led Bits</td>
<td></td>
<td>Power Color</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(0 – 63)</td>
<td></td>
<td>(0 – 255)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Power Intensity</td>
<td>(0 – 255)</td>
<td></td>
</tr>
<tr>
<td>Song</td>
<td>140</td>
<td>Song Number</td>
<td></td>
<td>Song Length</td>
<td>Note Number</td>
<td>Note Number</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(0 – 15)</td>
<td></td>
<td>(0 – 15)</td>
<td>1 (31 – 127)</td>
<td>2, etc.</td>
</tr>
<tr>
<td>Play</td>
<td>141</td>
<td>Song Number</td>
<td></td>
<td></td>
<td>Note Duration</td>
<td>Note Number</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(0 – 15)</td>
<td></td>
<td></td>
<td>1 (0 – 255)</td>
<td>2, etc.</td>
</tr>
<tr>
<td>Sensors</td>
<td>142</td>
<td>Packet Code</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(0 – 3)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Force-Seeking-Dock</td>
<td>143</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 13: OPCODE for the Roomba
Roomba Drive Control:

The Roomba’s drive control is accessed by sending a series of bytes to the serial port. (1) The Roomba must be in Safe or Full mode to accept this command. The first byte is [137], which instructs the Roomba that a drive command is about to follow. The drive command is a set of four data bytes which are interpreted into two 16 bit signed values using the two’s compliment representation. The first two bytes indicate the average velocity of the drive wheels in millimeters per second (mm/s) with the high byte sent first. The maximum velocity of the Roomba is 500 mm/s forward -500 mm/s in reverse. The second set of bytes is the radius in millimeters that the Roomba should turn. The longer radii makes the Roomba drive straighter and shorter radii makes the Roomba turn shorter. A negative radius will make the Roomba turn to the right, while a positive radius will make the Roomba turn to the left. The maximum values of 2000 to the left and -2000 to the right are allowed. A value of -1 will make the Roomba turn in place to the right and conversely a value of 1 will cause the Roomba to turn in place to the left. The last special case is the value of 32768 which instructs the Roomba to drive straight. However, the actual resulting direction depends on the environment, especially the tactility of the surface it will be traversing. A couple of example commands are as follows:

1. Instruct the Roomba to drive straight at 250 mm/s.

First we would take the 250 and convert it to its high and low byte

250/16 = 15 with a remainder of 10

15/16 = 0 with a remainder of 15

Making the hex representation of the number equal to FA or 0000 0000 1111 1010
Second we would take the 32768 and convert it to its high and low byte

\[
32768 / 16 = 2048 \text{ with a remainder of 0}
\]

\[
2048 / 16 = 128 \text{ with a remainder of 0}
\]

\[
128/16 = 8 \text{ with a remainder of 0}
\]

\[
8/16 = 0 \text{ with a remainder of 8}
\]

Making the hex representation of the number equal to 8000 or 1000 0000 0000 0000

The entire string would be in Hex [89] [00][FA][80][00] which would be in binary

1000 1001 0000 0000 1111 1010 1000 0000 0000 0000.

2. Instruct the Roomba to drive in reverse at -250 mm/s and -300 radius. Since both values are negative then the two compliment must be taken.

First convert the 250 to binary: 0000 0000 1111 1010 invert 1111 1111 0000 0101 and add 1 which

results in 1111 1111 0000 0110 or FF 06 Hex.

Second convert 300 to binary: 0000 0001 0010 1100 invert 1111 1110 1101 0011 and add one which

results in 1111 1110 1101 0100 or FE D4 Hex.

The entire string would be in Hex [89] [FF][06][FE][D4] which would be in binary

1000 1001 1111 1111 0000 0101 1111 1110 1101 0100
The code to implement this is as follows:

```csharp
public bool Drive(Velocity vSpeed, Radius rAngle)
{
    int num = rAngle.ToInt;
    byte byAngleHi = (byte)(num >> 8);
    byte byAngleLo = (byte)(num & 255);
    num = vSpeed.ToInt;
    byte bySpeedHi = (byte)(num >> 8);
    byte bySpeedLo = (byte)(num & 255);
    List<byte> lSend = new List<byte>();
    lSend.Add(OpCode.Drive);
    lSend.Add(bySpeedHi);
    lSend.Add(bySpeedLo);
    lSend.Add(byAngleHi);
    lSend.Add(byAngleLo);
    this.IO.Write(lSend.ToArray(), 0, lSend.Count);
}
```

This creates a byte array and adds the appropriate bytes and then sends them out the serial port.

**Roomba sensor data:**

The Roomba’s sensor data consist of 26 bytes and is returned in one of four methods. The OPCODE 142 must be sent to the Roomba to initiate retrieval of the sensor data. The immediate byte following the OPCODE is the representation of the type of sensor data requested. To retrieve the full 26 byte sensor package a zero would be sent, to retrieve subset 1 a one would be sent, to retrieve subset 2 a two would be sent, or to retrieve subset 3 a three would be sent. The different subsets are broken down as follows.

**Subset 1 includes 10 bytes**

a. Bumps and wheel drops

b. Wall
c. Cliff left

d. Cliff front left

e. Cliff front right

f. Cliff right

g. Virtual wall

h. Motor over currents

i. Dirt detect left

j. Dirt detect right

Subset 2 includes 6 bytes

a. Remote control command

b. Buttons

c. Distance

d. Angle

Subset 3 Includes 10 bytes

a. Charging state

b. Voltage

c. Current

d. Temperature

e. Charge

f. Capacity
The bumps and wheel drops are in the range of 0-31 and are either a 0 or 1 depending on if the sensor has been triggered or not. Each bit position is given the numerical value of $2^N$, where N is the actual bit position of the bit.

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sensor</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>Wheeldrop</td>
<td>Bump Caster</td>
<td>Left</td>
<td>Bump Right</td>
<td>Left</td>
</tr>
</tbody>
</table>

**Figure 14: SCI Bumps and Wheel Drops Byte Breakout**

For example if the caster was down and the left bumper was hit, then the Roomba’s sensor data would return a byte of 18. $2^4 + 2^1 = 18$

**Virtual Wall:**

The virtual wall is a device that sends out an infrared beam which the Roomba can see with the sensor located on the top front of the unit. (Figure 15B) As the Roomba was designed, the virtual wall will stop the Roomba from entering an area that you do not want the Roomba to enter. The downfall to the virtual wall is that it cannot be used as a straight wall. The beam that is emitted does not beam out such as a laser pointer; it actually fans out and gets wider as the distance expands. (Figure 15A) This feature of the virtual wall made the Roomba detect the virtual wall in small confined room constantly.
Bump Sensors:

There are three bump sensors on the front of the Roomba can be used to send a signal to the computer to cause the Roomba to take an action. These are the key percepts the Roomba has to see its environment. The sensor data is sent to the computer via the sensor packet and are translated as the logical true or false.
For the sake of brevity the rest of the OPCODES will not be discussed in detail as they bear no direct affect on the research and are of little value with the exception of giving an audible or visual display of completing a task.
Chapter 4: The Roomba task:

The interesting problem that emerged from the research of the Roomba was when instructing the Roomba to go straight using the default value of 32768 it would very frequently veer to the left or right. The use of odometry was discussed in detail in (6), and while they presented an algorithm to determine distance variations, they also had extreme trouble finding an algorithm for all Roombas. It was decided that it would be interesting to see if the Roomba could self calibrate to adjust for irregularities in the environment. For example, if the Roomba is used on a smooth surface rather than a carpeted surface, how would the Roomba drive differently on the two separate surfaces?

Experiment 1:

Based on the characteristics of how the Roombas traveled, it was determined that a small test track would be the best possible way to track the actual distance Bud would travel before veering off path. Bud was used to see if self calibration is possible. The test track was four feet by eight feet. The Roomba was placed in the center at the two foot mark and was given the command to go forward, represented by the number 32768. The speed of 500 is chosen to represent the maximum speed and the best chance to overcome any friction or obstacles that might impede the forward progress. The byte sequence sent to Bud was [137][1][244][128][0]. Bud was aligned and instructed to go, and in approximately four feet he had veered to a gradual
left and hit the wall. Calculating the distance and time it should have taken to traverse this distance, did not produce the correct results. At 500mm/s, and using 1 Foot = 0.3048 Meters and 1 millimeter / second = 0.0032808399 feet / second, the Roomba should have the capability of going 8.2020975 feet in approximately five seconds. Further exploration of the reason the Roomba did not perform as expected is necessary to compensate for mechanical differences, but is not covered in this experiment.

![Figure 17: Expected results (A) and actual results (B).](image)

A preliminary algorithm adjusted for the inconsistencies was based on the data gathered from the test runs of Bud. The time of the runs varied from four to five seconds, and was very consistent on the distance within a few inches. Observing Bud in the runs, showed about half the distance was the point of no return, ie the point where a simple correction would not help any more. The algorithm became a time based algorithm in which Bud would traverse in a straight line as defined by the manufacturer for the time it took to hit the wall minus the original two feet.
offset, and then adjust in the opposite direction. Implementation of the algorithm produced the following results. Bud went forward for two seconds and stopped and waited for the adjustment. However it was observed that the turning radius for the left and right were different. For example, if Bud were instructed to go 500 mm/s with a radius of 2000, the results did not mirror the results of instructing Bud to go 500 mm/s with a radius of -2000. The actual results of this instruction was Bud went two feet 6 inches to the left and ended approximately one foot from the wall when instructed to go 500 mm/s with an angle of 2000. However when instructed to drive in the opposite direction, Bud generally went straight and veered to the left when instructed to go 500 mm/s with an angle of -2000.

At first I thought that the issue might be specific to Bud, so I replaced Bud with Lou and

Figure 18: Previous code showing packets sent to and received from the Roomba

he proceeded to act in the similar method. I then dissected the code to make sure it was not a systemic problem, and that it was sending the correct code to the Roombas. To test this, I used a
offset, and then adjust in the opposite direction. Implementation of the algorithm produced the following results. Bud went forward for two seconds and stopped and waited for the adjustment. However it was observed that the turning radius for the left and right were different. For example, if Bud were instructed to go 500 mm/s with a radius of 2000, the results did not mirror the results of instructing Bud to go 500 mm/s with a radius of -2000. The actual results of this instruction was Bud went two feet 6 inches to the left and ended approximately one foot from the wall when instructed to go 500 mm/s with an angle of 2000. However when instructed to drive in the opposite direction, Bud generally went straight and veered to the left when instructed to go 500 mm/s with an angle of -2000.

At first I thought that the issue might be specific to Bud, so I replaced Bud with Lou and he proceeded to act in the similar method. I then dissected the code to make sure it was not a systemic problem, and that it was sending the correct code to the Roombas. To test this, I used a

![Figure 18: Previous code showing packets sent to and received from the Roomba](image)

rather than four. Nevertheless, the algorithm held true for the eight foot test track. The Roomba’s ended within 16 – 18 inches from the wall. Compared to the 24 inches they had started at.

Results

Overall, experiment one was a success in capturing data for how the Roomba reacted and pointed out the vast difference in driving on different surfaces which would limit the scope this algorithm would work. However, the Roomba did work well with using the bump sensors to making a decision, similar to a Simple agent (see Appendix C) reacting to percepts in its environment. (9)

A moment of reflection based on Experiment 1

Environmental factors:

The most unpredictable element in this research was the environment. Particularly the surface the Roomba’s test run was done on. The original surface was a medium pile carpet with a high pad. The Roomba consistently veered to the left on this surface. Once the Roomba was working according to the programs specifications, the test was redone on a smooth tiled linoleum floor. The Roomba veered to the right on this type surface. A small low pile throw rug was on the floor and when the Roomba went on the rug the direction of the veering changed causing the Roomba to adjust in the opposite direction without any program instructions. This gave an
interesting twist and demonstrated the unpredictability the environment caused on the experiments. The hypothesis was made that the direction of the carpet was affecting the ability of the Roomba to drive straight; however, after completion of the experiments on several different surface types, the test was rerun on the original surface and the Roomba performed identical to the driving of the original test. These findings gave causality to adjust the original hypothesis to: The ability of the Roomba to drive straight is dependent on the surface effect on the drive wheels and the voltage that is applied to the wheel.

**Electronic and mechanical factors:**

Expanding on the environmental factors, the Roomba’s drive motors simply attach to the wheel with an o-ring. (10) This finding explained a lot about why the Roomba reacted to different surfaces. The wheel is comprised of a rubber surface that has three rubber square tips (Figure 20) in each row that wrap around the whole tire. The higher the pile on the carpet relates to the
amount of carpet surface that come in contact with the entire surface, including carpet that can
go between the square tips causing more friction. Conversely the flatter the surface the Roomba
is driving on, the only surface of the tire making contact with the floor is the square rubber tip
dramatically reducing the friction. The second issue effecting the driving of the Roomba is the o-
ing. It has two potential slippage points, one at the motor and one at the wheel. If either one of
these points loose traction the wheel will fail to rotate accurately.

The final contributor in the Roomba not driving straight is the current supplied to the wheel
motor. Tracking the voltage or current supplied to the wheel motor will be difficult and would
require disassembling the Roomba and attaching a multi-meter to track the voltages, I will leave this to future research. Overall, with these three contributing factors it is almost impossible to ensure that the Roomba will drive predictably straight in an actual environment vs a modeled environment. The quick upgrade to the drive motors would be to supply stepper motors. A stepper motor is unlike a typical direct current(DC) motor which is provided current and simple turns. There is no way to track the accurate position of the turning motor; however, with the stepper motor, the motor will turn in ‘steps’ and give an accurate way to track the exact rotation of the motor. Although, this might also make the Roomba cost prohibitive to use as an introduction to robotics.

**Battery Issues:**

One rather large issue is one regarding the life of the battery. The Roomba is very demanding on the battery. The addition of the Bluetooth adaptor only exacerbates this issue. The only saving grace was the ability to switch out Bud and Lou. However, when both Roomba’s were used for approximately one hour then they would need to be charged for approximately four hours before retesting. The Roomba Scheduler itself was designed with the self charging base seems to be acceptable as the Roomba will vacuum until the battery gets low and then return to the charging base. Although, using the Roomba as a programming platform with attachments such as the Bluetooth, will cause malfunctions in communications if the battery gets low.
Experiment 2: Calibration

Due to the immense variations in the direction of travel on various floors, I thought it would be interesting to see if a semi accurate approximation could be derived from the Roomba. While this was not thoroughly integrated in the code, it was tested and seems to have some merit. The calibration was fairly simple and was implemented using the four foot test track and by calculating the distance and time it took to hit the first bumper. For the test runs with Bud, he went 4.5 feet in eight seconds. Using the test values of 500 mm/s and the command 32768 to go straight, it should have gone 8.202 feet. Adjusting for the distance the Roomba actually drove and the time it took to drive on the current surface, the Roomba's actual conversion rate could be used. Once the time was calculated from the system time, it is divided into the total distance actually traveled to get the new conversion rate based on the speed 500.

Several test runs were done in parallel to the Roomba testing and when the Roomba stopped the total distance was calculated and displayed. The calculated distance and the actual distance varied less than six inches in a twelve foot run.

Results

The program was fairly accurate with respect to short runs. The maximum distance this was tested on was approximately twelve feet. I hypothesize the Roomba will not be as accurate over longer distances due to the amount of arc observed each time the Roomba moves forward.
Experiment 3:

Testing Environment:

The testing environment changed a little over time, but was solidified into a pathway of four feet in width. The distance was chosen arbitrarily at first, but actually proved to be the best distance for testing the Roomba. Any distance over four feet gave the Roomba the opportunity to over veer thus making the test run invalid. The Roomba would actually turn to a degree to which the left bumper would be engaged while the Roomba was veering to the right.

![Diagram A](image1.png)  ![Diagram B](image2.png)

**Figure 22:** Roomba making contact with wall Valid (A) Roomba over Veering Invalid (B)
This will be classified as an invalid test. The floor surface was used to clearly see if the Roomba could traverse different types of materials and still function in the same manner. The types of flooring tested on were medium pile carpet, low pile carpet, linoleum, tile floor with grout lines of varying widths, and hardwood floors.

The task of getting the Roombas to learn to go straight was very intriguing. The formulating of an algorithm to correctly was found to be nontrivial. Since the Roomba reacted differently on different types of surfaces, it is necessary to adjust the driving path or angle to get the Roomba moving again once it hits the wall. Using the data collected from the calibration test, it was clear to see that the Roomba will normally veer to the left or the right and in very few tests did the Roomba tend to go straight, but even then the Roomba usually started to veer within ten feet. Using the testing width of four feet, the Roomba was placed generally in the center of the corridor and instructed to go. The Roomba then went forward and as in the calibration test, it then veered to the left and hit the left bump sensor. Looking at the angle of the Roomba and where it hit the wall, a generalization of the position was approximately 45 degrees off of the original orientation. The question arose of how much time to rotate the Roomba in place would equate to 45 degrees of correction to straighten the Roomba. The rotation of the Roomba would use the equation of an arc distance.

\[ s = r\theta \]

Where \( s \) is the length in inches, \( r \) is the radius, and \( \theta \) is the angle in radians.

To convert degrees to radians:

\[ 1^\circ = \frac{\Pi}{180} \text{ radian} \approx 0.0174533 \text{ radian} \]

The Roomba is 13 inches in diameters making the radius 6.5 inches. Doing the calculations for the distance the Roomba needs to turn is:
S = 6.5 (.01745 * 45) = 5.1 Inches

The Roombas’ ability to turn in place is the command -1 or 1 depending on the direction desired. The speed of rotation and time to rotate are the only unknown variables. Due to the length of the battery life on a full charge and since the value is arbitrary, the value of 500 was chosen. Using the following formula, the total time of the rotation can be determined:

Remember the Roombas speed is millimeters per second. t = time for rotation.

1 meter = 3.2808399 feet, 1 millimeter = 0.001 meters, and 1 foot = 12 inches

t = 500 (3.2808399 * .001) * (5.1 / 12) = .697s

C# has the capability to pause the current thread for a time given in milliseconds. The code to implement this is as such:

```csharp
m_iCurrentSpeed = 500;
Program.UI.CurrentRoomba.Drive((Velocity)m_iCurrentSpeed, (Radius)(m_iCurrentAngle));
Thread.Sleep(697);
```

The Roomba hit the wall and then attempted to rotate, but was still touching the wall which caused excess friction, and not allowing the Roomba to turn properly. Addressing this issue included backing up and then stopping, but backing up introduced new calculations and similar to the issue of friction and straight driving, backing up was also erratic. Since the bump sensor was still making contact with the wall, the Roomba was instructed to backup until the sensor was cleared. This was implemented for all the sensors. The next issue to arise was the speed of the Roomba. While this was not a major issue on the high pile carpet, and was not as noticeable, the speed of the Roomba on a smooth surface or low pile carpet would cause the Roomba to overturn due to sheer momentum. For clarification, if the Roomba was going 500 mm/s and it
hits the left bumper, it does not instantaneously stop, but hits the wall and keeps rotating until all
the potential energy is absorbed by the wall. This over steering throws off the calculations for
the direction correction. The speed was stepped down until the Roomba was able to hit the wall
without significant over steering. The resultant speed was 300 mm/s.

**Results:**

The Roomba proceeded as programmed. When the Roomba hit the left wall, it backed-up until the sensor was cleared and adjusted approximately 45 degrees and started off again. The results were encouraging, as the Roomba was able to make its way through a four foot channel. However, with other objects such as pillars in the way or a simple office chair, the Roomba has a potential for two sensors to be hit or a single sensor to be hit twice which in turn will cause the Roomba to over correct. The overall result of this experiment is deemed a success because the Roomba was able to navigate while adjusting to obstacles.

**Experiment 4:**

Expanding on Experiment 3, can the Roomba learn its own adjustment regardless of the surface it is on? To implement this, the Roomba would have to be given a memory of how much it is to turn. The second main difference is that a goal was given to the Roomba. The goal was to find the virtual wall. The first algorithm used was to track the total number of bumps to the right and total number of bumps to the left. The difference of the two would be the actual
distance the Roomba would rotate. At first, this seemed to work, but after the Roomba adjusted to the right to a degree that the left wall fell on the tangent of the path, then the Roomba would overcorrect to the opposite direction. The flaw in the algorithm is that it only kept one value for the turning radius. The algorithm was changed to track the total correction in both directions.

**Test Track A:**

In order to test this, the multi-surface floor was used with a length of approximately fifty feet. The surface comprised of low pile carpet with intermittent tile surface every ten feet. Bud tested this track and instead of the normal veering to the left, he went to the right. The hallway width was more than the experimental four foot limit, and as expected Bud over steered until the left bump sensor mad contact with the right wall making this an invalid test. To rectify this, an obstacle was placed approximately four feet from the left wall and the test was re-run. Bud then veered to the right impacting the right obstacle and then backup corrected and then continued forward. Figure 23 shows the path Bud drove while in the test. As one can see the Roomba hit the wall multiple times in a short distance and as it learned how much to correct, Bud would drive longer before coming in contact with the right obstacle again. Since this hallway was sufficiently long enough, I was able to test the opposite wall as well. Once the Roomba has hit the opposite wall, then the Roomba then learned the opposite wall turning radius. Bud again repeated the multiple hits on the left wall unit it ended at the end of the hallway with contact on the center bump sensor.
Figure 23: Bud traversing 50 Foot Hallway

Experiment 4 Expanded

Test Track B:

Track B consisted of an ‘L-shape’ path as depicted in Figure 24. Lou was used on this
Figure 23: Bud traversing 50 Foot Hallway

Experiment 4 Expanded

Test Track B:

Track B consisted of an ‘L-shape’ path as depicted in Figure 24. Lou was used on this
Track C:

Track C consisted of a decline to see if the Roomba would react differently with an incline. The track was in the same shape as track B but had the decline as the first leg of the track. The Ramp was a standard ramp as used for wheelchairs. Bud was chosen for this track and preformed much as Lou had performed on the flat track. Bud also found the goal of the virtual wall. Once the test runs were complete, Bud was run in the opposite direction with the only change to the program being to go right on contact with the center bump sensor. Again the steady incline did not have a noticeable effect on the driving, but may have slowed down the speed at which he ascended.

Track D:

Track D consisted of two mini golf greens at Rockwood Golf Park. The two greens were two and eleven, as they were the only with significant turns to test the Roomba. Lane two was rather incident free with the exception of the ball stop on the incline to keep the ball from rolling backwards and at the end of the path was area where the width exceeded the four feet tolerance. The ball stop put the Roomba in a bind so it could no drive forward a small push forward was all it took to overcome the friction. The width change as expected caused the Roomba to overturn and since it was biased to go right, it began to backtrack. This was mitigated by limiting the width to four feet.
Path eleven was a more interesting test as it formed a loop with significant inclines and declines. The Roomba was affected by this incline but still managed to find the goal. The issue
with the incline was it was concave as well in the center of the incline and as the Roomba reached the wall; the outer wheel would slip a little but then gain traction enough to rotate.
Figure 27: Bud on Hole 2 path
Experiment 4 results and conclusions

Overall without incident, Bud and Lou were both able to navigate through a path that conformed to the width constraint regardless of inclines or turns.
Chapter 5: Conclusion

The Roomba as designed with minimal instruction can be used as a simple agent. The program acting as the memory of the agent can be taught to react to the environment it is placed in by responding to the percepts. While the Roomba scheduler is limited to a minimal number of percepts, future series of Roombas will contain the ability to add percepts that will expand the capability for the agent to perceive the environment it resides in. The perception of this additional information will give the agent a greater knowledge of the environment. The downfall of using the current line of Roombas is the drive train. The Roomba exhibits great awareness given a confined area to work in, but to expand the area that the Roomba operate within, the motors would need to be changed to a stepper motor or similar motor that will allow for greater accuracy. Even the stepper motor cannot be 100% correct, because if it should hit a patch of ground with little friction, the wheel could still turn but the Roomba would not drive as expected.
Chapter 6: Future Work

The Roomba has a plethora of experiments it can be used in, and while this paper focuses on only the drive control, it is the first essential step to understand before expanding the project into guidance of robots. The Roomba can be used as an experimental test bed for various real world simulations of modeled environments. While the Roomba scheduler is limited by the limit of the width of a track due to over veering, the new Roomba create has additional inputs that can be used to monitor other serial devices. Should an inferred device that can pick up various signals be used in conjunction with a device that emits the proper signal, the Roomba should be able to find that device and act upon it based on the percepts. The Roomba as designed still remains an outstanding platform to introduce simple agents and the environment they are in.
Appendix A: HEX conversion table.

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Hex(Byte)</th>
<th>Decimal</th>
<th>HEX(Byte)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0000</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>-3</td>
<td>FFFD</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>-6</td>
<td>FFFA</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>-9</td>
<td>FFF7</td>
</tr>
<tr>
<td>12</td>
<td>C</td>
<td>-12</td>
<td>FFF4</td>
</tr>
<tr>
<td>15</td>
<td>F</td>
<td>-15</td>
<td>FFF1</td>
</tr>
<tr>
<td>18</td>
<td>12</td>
<td>-18</td>
<td>FFEE</td>
</tr>
<tr>
<td>21</td>
<td>15</td>
<td>-21</td>
<td>FFEB</td>
</tr>
<tr>
<td>24</td>
<td>18</td>
<td>-24</td>
<td>FFE8</td>
</tr>
<tr>
<td>27</td>
<td>1B</td>
<td>-27</td>
<td>FFE5</td>
</tr>
<tr>
<td>30</td>
<td>1E</td>
<td>-30</td>
<td>FFE2</td>
</tr>
<tr>
<td>33</td>
<td>21</td>
<td>-33</td>
<td>FFDF</td>
</tr>
<tr>
<td>36</td>
<td>24</td>
<td>-36</td>
<td>FFDC</td>
</tr>
<tr>
<td>39</td>
<td>27</td>
<td>-39</td>
<td>FFD9</td>
</tr>
<tr>
<td>42</td>
<td>2A</td>
<td>-42</td>
<td>FFD6</td>
</tr>
<tr>
<td>45</td>
<td>2D</td>
<td>-45</td>
<td>FFD3</td>
</tr>
<tr>
<td>48</td>
<td>30</td>
<td>-48</td>
<td>FFD0</td>
</tr>
<tr>
<td>51</td>
<td>33</td>
<td>-51</td>
<td>FFCD</td>
</tr>
<tr>
<td>54</td>
<td>36</td>
<td>-54</td>
<td>FFCA</td>
</tr>
<tr>
<td>57</td>
<td>39</td>
<td>-57</td>
<td>FFC7</td>
</tr>
<tr>
<td>60</td>
<td>3C</td>
<td>-60</td>
<td>FFC4</td>
</tr>
<tr>
<td>63</td>
<td>3F</td>
<td>-63</td>
<td>FFC1</td>
</tr>
<tr>
<td>66</td>
<td>42</td>
<td>-66</td>
<td>FFBE</td>
</tr>
<tr>
<td>69</td>
<td>45</td>
<td>-69</td>
<td>FFBB</td>
</tr>
<tr>
<td>72</td>
<td>48</td>
<td>-72</td>
<td>FFB8</td>
</tr>
<tr>
<td>75</td>
<td>4B</td>
<td>-75</td>
<td>FFB5</td>
</tr>
<tr>
<td>78</td>
<td>4E</td>
<td>-78</td>
<td>FFBC</td>
</tr>
<tr>
<td>81</td>
<td>51</td>
<td>-81</td>
<td>FFAF</td>
</tr>
<tr>
<td>84</td>
<td>54</td>
<td>-84</td>
<td>FFAC</td>
</tr>
<tr>
<td>87</td>
<td>57</td>
<td>-87</td>
<td>FFA9</td>
</tr>
<tr>
<td>90</td>
<td>5A</td>
<td>-90</td>
<td>FFA6</td>
</tr>
<tr>
<td>93</td>
<td>5D</td>
<td>-93</td>
<td>FFA3</td>
</tr>
<tr>
<td>96</td>
<td>60</td>
<td>-96</td>
<td>FFA0</td>
</tr>
<tr>
<td>99</td>
<td>63</td>
<td>-99</td>
<td>FF9D</td>
</tr>
<tr>
<td>102</td>
<td>66</td>
<td>-102</td>
<td>FF9A</td>
</tr>
<tr>
<td>105</td>
<td>69</td>
<td>-105</td>
<td>FF97</td>
</tr>
<tr>
<td>108</td>
<td>6C</td>
<td>-108</td>
<td>FF94</td>
</tr>
<tr>
<td>111</td>
<td>6F</td>
<td>-111</td>
<td>FF91</td>
</tr>
<tr>
<td>114</td>
<td>72</td>
<td>-114</td>
<td>FF8E</td>
</tr>
<tr>
<td>Hex</td>
<td>Dec</td>
<td>Hex</td>
<td>Dec</td>
</tr>
<tr>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
</tr>
<tr>
<td>117</td>
<td>75</td>
<td>-117</td>
<td>FF8B</td>
</tr>
<tr>
<td>123</td>
<td>7B</td>
<td>-123</td>
<td>FF85</td>
</tr>
<tr>
<td>129</td>
<td>81</td>
<td>-129</td>
<td>FF7F</td>
</tr>
<tr>
<td>135</td>
<td>87</td>
<td>-135</td>
<td>FF79</td>
</tr>
<tr>
<td>141</td>
<td>8D</td>
<td>-141</td>
<td>FF73</td>
</tr>
<tr>
<td>147</td>
<td>93</td>
<td>-147</td>
<td>FF6D</td>
</tr>
<tr>
<td>153</td>
<td>99</td>
<td>-153</td>
<td>FF67</td>
</tr>
<tr>
<td>159</td>
<td>9F</td>
<td>-159</td>
<td>FF61</td>
</tr>
<tr>
<td>165</td>
<td>A5</td>
<td>-165</td>
<td>FF5B</td>
</tr>
<tr>
<td>171</td>
<td>AB</td>
<td>-171</td>
<td>FF55</td>
</tr>
<tr>
<td>177</td>
<td>B1</td>
<td>-177</td>
<td>FF4F</td>
</tr>
<tr>
<td>183</td>
<td>B7</td>
<td>-183</td>
<td>FF49</td>
</tr>
<tr>
<td>189</td>
<td>BD</td>
<td>-189</td>
<td>FF43</td>
</tr>
<tr>
<td>195</td>
<td>C3</td>
<td>-195</td>
<td>FF3D</td>
</tr>
<tr>
<td>201</td>
<td>C9</td>
<td>-201</td>
<td>FF37</td>
</tr>
<tr>
<td>207</td>
<td>CF</td>
<td>-207</td>
<td>FF31</td>
</tr>
<tr>
<td>213</td>
<td>D5</td>
<td>-213</td>
<td>FF2B</td>
</tr>
<tr>
<td>219</td>
<td>DB</td>
<td>-219</td>
<td>FF25</td>
</tr>
<tr>
<td>225</td>
<td>E1</td>
<td>-225</td>
<td>FF1F</td>
</tr>
<tr>
<td>231</td>
<td>E7</td>
<td>-231</td>
<td>FF19</td>
</tr>
<tr>
<td>237</td>
<td>ED</td>
<td>-237</td>
<td>FF13</td>
</tr>
<tr>
<td>243</td>
<td>F3</td>
<td>-243</td>
<td>FF0D</td>
</tr>
<tr>
<td>Address</td>
<td>Value</td>
<td>Offset</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------</td>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>246</td>
<td>F6</td>
<td>-246</td>
<td>FF0A</td>
</tr>
<tr>
<td>249</td>
<td>F9</td>
<td>-249</td>
<td>FF07</td>
</tr>
<tr>
<td>252</td>
<td>FC</td>
<td>-252</td>
<td>FF04</td>
</tr>
<tr>
<td>255</td>
<td>FF</td>
<td>-255</td>
<td>FF01</td>
</tr>
<tr>
<td>258</td>
<td>102</td>
<td>-258</td>
<td>FEF9</td>
</tr>
<tr>
<td>261</td>
<td>105</td>
<td>-261</td>
<td>FEF8</td>
</tr>
<tr>
<td>264</td>
<td>108</td>
<td>-264</td>
<td>FEF5</td>
</tr>
<tr>
<td>267</td>
<td>10E</td>
<td>-267</td>
<td>FEF2</td>
</tr>
<tr>
<td>270</td>
<td>111</td>
<td>-270</td>
<td>FEEF</td>
</tr>
<tr>
<td>273</td>
<td>114</td>
<td>-270</td>
<td>FEEC</td>
</tr>
<tr>
<td>276</td>
<td>117</td>
<td>-270</td>
<td>FEEC</td>
</tr>
<tr>
<td>279</td>
<td>11A</td>
<td>-270</td>
<td>FEE6</td>
</tr>
<tr>
<td>282</td>
<td>11D</td>
<td>-270</td>
<td>FEE3</td>
</tr>
<tr>
<td>285</td>
<td>120</td>
<td>-270</td>
<td>FEE0</td>
</tr>
<tr>
<td>288</td>
<td>123</td>
<td>-270</td>
<td>FED7</td>
</tr>
<tr>
<td>291</td>
<td>126</td>
<td>-270</td>
<td>FED4</td>
</tr>
<tr>
<td>294</td>
<td>129</td>
<td>-270</td>
<td>FED1</td>
</tr>
<tr>
<td>297</td>
<td>132</td>
<td>-270</td>
<td>FECE</td>
</tr>
<tr>
<td>300</td>
<td>135</td>
<td>-270</td>
<td>FECB</td>
</tr>
<tr>
<td>303</td>
<td>138</td>
<td>-270</td>
<td>FEC8</td>
</tr>
<tr>
<td>306</td>
<td>13B</td>
<td>-270</td>
<td>FEC5</td>
</tr>
<tr>
<td>309</td>
<td>13E</td>
<td>-270</td>
<td>FEC2</td>
</tr>
<tr>
<td>312</td>
<td>141</td>
<td>-270</td>
<td>FEBF</td>
</tr>
<tr>
<td>315</td>
<td>144</td>
<td>-270</td>
<td>FEBC</td>
</tr>
<tr>
<td>318</td>
<td>147</td>
<td>-270</td>
<td>FEB9</td>
</tr>
<tr>
<td>321</td>
<td>14A</td>
<td>-270</td>
<td>FEB6</td>
</tr>
<tr>
<td>324</td>
<td>150</td>
<td>-270</td>
<td>FEB0</td>
</tr>
<tr>
<td>327</td>
<td>153</td>
<td>-270</td>
<td>FEAD</td>
</tr>
<tr>
<td>330</td>
<td>156</td>
<td>-270</td>
<td>FEAA</td>
</tr>
<tr>
<td>333</td>
<td>159</td>
<td>-270</td>
<td>FEA7</td>
</tr>
<tr>
<td>336</td>
<td>15C</td>
<td>-270</td>
<td>FEA4</td>
</tr>
<tr>
<td>339</td>
<td>15F</td>
<td>-270</td>
<td>FEA1</td>
</tr>
<tr>
<td>342</td>
<td>162</td>
<td>-270</td>
<td>FE9E</td>
</tr>
<tr>
<td>345</td>
<td>165</td>
<td>-270</td>
<td>FE9B</td>
</tr>
<tr>
<td>348</td>
<td>168</td>
<td>-270</td>
<td>FE98</td>
</tr>
<tr>
<td>363</td>
<td>16B</td>
<td>-270</td>
<td>FE95</td>
</tr>
<tr>
<td>366</td>
<td>16E</td>
<td>-270</td>
<td>FE92</td>
</tr>
<tr>
<td>369</td>
<td>171</td>
<td>-270</td>
<td>FE8F</td>
</tr>
<tr>
<td>372</td>
<td>174</td>
<td>-270</td>
<td>FE8C</td>
</tr>
<tr>
<td>Address</td>
<td>Value</td>
<td>Address</td>
<td>Value</td>
</tr>
<tr>
<td>---------</td>
<td>-------</td>
<td>---------</td>
<td>-------</td>
</tr>
<tr>
<td>375</td>
<td>177</td>
<td>-375</td>
<td>FE89</td>
</tr>
<tr>
<td>402</td>
<td>192</td>
<td>-402</td>
<td>FE6E</td>
</tr>
<tr>
<td>432</td>
<td>1B0</td>
<td>-432</td>
<td>FE50</td>
</tr>
<tr>
<td>459</td>
<td>1CB</td>
<td>-459</td>
<td>FE35</td>
</tr>
</tbody>
</table>
Appendix B: C# Source Code for RoombaSCI:Types

frmStart
using System;
using System.IO;
using System.Drawing;
using System.Windows.Forms;
using System.Collections.Generic;
using System.Xml;
using System.Xml.Serialization;

using RoombaSCI;
using Kevin_Logging;

namespace roomba_term
{
    public partial class frmStart : roomba_term.frmMenu
    {
        #region Constants
        protected const char c_sDot = '.';
        protected const string c_sVoltage = "Voltage: ";
        protected const string c_sCurrent = "Current: ";
        protected const string c_sChargeState = "Charge State: ";
        protected const string c_sIsCurrent = "Roomba.IsCurrent = ";
        protected const string c_sFinished = "Finished"
        protected const string c_sOff = "Off";
        protected const string c_sTick = "Tick.."; // + " Interval: ";
        protected const string c_sNoRead = "No Read";
        protected const string c_sStopped = "Stopped";
        protected const string c_sSuspended = "Suspended"; // Means that another process in Roomba-Term has stopped polling for its own needs
        protected const string c_sRecording = "Recording Macro";
        protected const string c_sNotRecording = "Not Recording";
        protected const string c_sExecuting = "Executing";
        protected const string c_sNotExecuting = "NotExecuting";
        protected const string c_sRoombaTerm = "Roomba-Term: ";
        protected const string c_sLoadingConfig = "Loading Config File: ";
        #endregion
        #region Member Variables
        static int m_iFormUpdated_DisplayLag = 0;
        static bool m_bPluggedInFlasher = false;
        static bool m_bRecordingFlasher = false;
        static bool m_bExecutingFlasher = false;
        #endregion
    }
}
public frmStart()
{
    InitializeComponent();
    Program.UI = new RoombaUI();
}

public double p_dFormUpdated;

public double FormUpdated
{
    get
    {
        return (this.p_dFormUpdated);
    }
    set
    {
        this.p_dFormUpdated = value;
    }
}

#region Events

    #region Form

    private void frmStart_Load(object sender, EventArgs e)
    {
        Program.UI.CurrentRoomba = new Roomba_Poller();
        this.LoadForm();
    }

    private void frmStart_FormClosing(object sender, FormClosingEventArgs e)
    {
        this._Stop(false, true);
    }

    #endregion

    #region Buttons

    private void btnShutDown_Click(object sender, EventArgs e)
    {
        this.Close();
    }

    private void btnMinimize_Click(object sender, EventArgs e)
    {
        this.WindowState = FormWindowState.Minimized;
    }

    #endregion

    #region Timers

    private void tForm_Timer_Tick(object sender, EventArgs e)
    {
    }

    #endregion

private void explanationOfThisFormToolstripMenuItem_Click(object sender, EventArgs e)
{
    string explanation;

    explanation = "This form is the startup & current status form. It must be running for the other forms to work. 
    ";
    explanation += "This app is presented as a sample application that uses the C# RoombaSCI framework created by Kevin Gabbert. 
    ";
    explanation += "Double-Click on the boxes in the upper right of the Start Form to start/Stop communication to Roomba. 
    ";
    explanation += "You can Double-Click on any of the boxes on this Form to start/Stop their primary function. 
    ";
    explanation += "The colored boxes in the upper right are there to give you feedback on this application's connection with Roomba. 
    ";
    explanation += "Red is obvious. something is not working. Green shows you that something is working. Light green means things are working. 
    ";
    explanation += "working too, but it shows where this program is making an assumption. For example, If Roomba keeps giving this app back 
    ";
    explanation += "the same thing over and over again, then for speed purposes, this program will not do any evaluation or parsing of new data that is similar. 
    ";
    explanation += "In that case, light green is shown. I thought it helpful to show it to you so that you can know what is going on. When a box is the same color as the form, that means nothing is happening there. It may be that this program has nothing to parse, or there is no data being received. 
    ";
    explanation += "The blue text in the parse box means that the data it is parsing is not new. Black text = new data. The time taken to parse is expressed in Ticks. 
    ";
    explanation += "This form uses COM1 by default. You will need to change the port if Roomba is set up to a different COM Port. 
    ";
    explanation += "The logs & macro filenames are automatically generated with a unique name in the application folder, or bin dir. Use the menu to open the latest.";
    MessageBox.Show(explanation);
}

private void restartConnectionToolstripMenuItem_Click(object sender, EventArgs e)
{
    this._Start();
}

private void stopToolstripMenuItem_Click(object sender, EventArgs e)
{
    this._Stop(true, true);
}

private void startTimerToolstripMenuItem_Click(object sender, EventArgs e)
{
    Log.This("Starting Form Timer via Menu", this.Name, Program.UI.Config.Log.StartForm);
}
this.tForm_Timer.Start();

private void stopTimerToolStripMenuItem_Click(object sender, EventArgs e)
{
    Log.This("Stopping Form Timer via Menu", this.Name, Program.UI.Config.Log.StartForm);
    this.tForm_Timer.Stop();
}

private void configureToolStripMenuItem_Click(object sender, EventArgs e)
{
    string sOpen = "tabPolling";
    Log.This("Open Form via Menu: " + sOpen, this.Name, Program.UI.Config.Log.StartForm);
    Program.UI.Open_Config_Form(this, sOpen);
}

private void openToolStripMenuItem1_Click(object sender, EventArgs e)
{
    Log.This("Open Log via Menu", this.Name, Program.UI.Config.Log.StartForm);
    RoombaUI.ShowFile(Program.UI.LogPath);
}

private void clearToolStripMenuItem_Click(object sender, EventArgs e)
{
    int iDeletedFiles = 0;

    Log.This("Clear Form Logs Requested", this.Name, Program.UI.Config.Log.StartForm);

    string[] sLogFiles = Directory.GetFiles(Path.GetDirectoryName(Application.ExecutablePath), "roomba-term.EXE Log*" );

    Log.This(sLogFiles.Length + " Log Files Found.", this.Name, Program.UI.Config.Log.StartForm);

    foreach (string sCurrent in sLogFiles)
    {
        try
        {
            //Don't shoot the food!
            if (sCurrent != Program.UI.LogPath)
            {
                File.Delete(sCurrent);
                Log.This(sCurrent + " Deleted.", this.Name, Program.UI.Config.Log.StartForm);
                iDeletedFiles += 1;
            }
            else
            {
                Log.This("This log file ignored: " + sCurrent, this.Name, Program.UI.Config.Log.StartForm);
            }
        }
    }
catch (Exception ex)
{
    //Keep going
    Log.This("Delete File Error: " + ex.Message, this.Name,
    Program.UI.Config.Log.StartForm);
}

string sLogFilesDeleted = iDeletedFiles.ToString() + " Log Files Deleted."
;
Log.This(sLogFilesDeleted, this.Name, Program.UI.Config.Log.StartForm);
MessageBox.Show(sLogFilesDeleted);

private void openCurrentToolStripMenuItem_Click(object sender, EventArgs e)
{
    Log.This("Open Macro via Menu", this.Name, Program.UI.Config.Log.StartForm);
    RoombaUI.ShowFile(Program.UI.MacroPath);
}

private void loadToolStripMenuItem_Click(object sender, EventArgs e)
{
    Log.This("Execute Saved Macro Requested", this.Name,
    Program.UI.Config.Log.StartForm);

    //Create a new ShowOpenFile Dialog
    DialogResult result = loadMacro.ShowDialog();
    string selectedFile = loadMacro.FileName;

    Program.UI.CurrentRoomba.Macro.Execute(selectedFile);
}

#region labels
private void llSet_COMM_Port_LinkClicked(object sender, LinkLabelLinkClickedEventArgs e)
{
    string sOpen = "tabCOMM";
    Log.This("Open Form via Menu: " + sOpen, this.Name, Program.UI.Config.Log.StartForm);
    Program.UI.Open_Config_Form(this, sOpen);
}
#endregion

private void insetControls_DoubleClick(object sender, EventArgs e)
{
    if (!Program.UI.Started)
    {
        Log.This("Double-Click Start", this.Name, Program.UI.Config.Log.StartForm);
        this._Start();
    }
}
else
{
    Log.This("Double-Click Stop", this.Name, Program.UI.Config.Log.StartForm);
    this._Stop(true, true);
}

if (Program.UI.CurrentRoomba == null)
{
    Program.UI.CurrentRoomba = new Roomba_Poller();
}

private void lblRecording_DoubleClick(object sender, EventArgs e)
{
    this.ToggleRecordMacro();
}
private void lblMacroExecuting_DoubleClick(object sender, EventArgs e)
{
    this.ToggleExecuteMacro();
}

private void lblRecording_MouseHover(object sender, EventArgs e)
{
   ToolTip1.SetToolTip(this.lblRecording, "double-click to Start/Stop Recording Macro");
}
private void lblMacroExecuting_MouseHover(object sender, EventArgs e)
{
    ToolTip1.SetToolTip(this.lblRecording, "double-click to Start/Stop Macro Execution");
}

private void lblChargeState_MouseHover(object sender, EventArgs e)
{
    ToolTip1.SetToolTip(this.lblChargeState, "This control isn't perfect yet. it falsely reports sometimes.");
}
private void lblPoller_MouseHover(object sender, EventArgs e)
{
    ToolTip1.SetToolTip(this.lblChargeState, "If Roomba won't communicate, the battery might be low.");
}

#endregion

#endregion

#endregion

private void LoadForm()
{

loadToolstripMenuItem.Enabled = false;

Program.UI.Setup_Log(this.lblVersion.Text);

Log.DebugMode = Program.UI.DebugMode = this.chkAppDebugMode.Checked;
Log.This(c_sRoombaTerm + this.lblVersion.Text, this.Name, true);

this.tForm_Timer.Interval = (Program.UI.Config.Forms.StartForm.Timer);
Log(This("Starting Form Timer - Form_Load", this.Name, true);
this.tForm_Timer.Start();

string path = Path.GetDirectoryName(System.Reflection.Assembly.GetExecutingAssembly().Location);
string sConfig_File = path + @"\config.xml";

Log.This(c_sLoadingConfig + sConfig_File, this.Name, true);

if (File.Exists(sConfig_File))
{
    roomba_term.Config_Settings csSavedData;
    XmlSerializer s = new XmlSerializer(typeof(roomba_term.Config_Settings));

    TextReader trRead = new StreamReader(sConfig_File);
    csSavedData = (roomba_term.Config_Settings)s.Deserialize(trRead);
    trRead.Close();

    Program.UI.Config = csSavedData;

    Log(This("Config File loaded.", this.Name, Program.UI.Config.Log.StartForm);

    this.lblCOMM_Port_Used.Text = Program.UI.Config.COMM.ConnectedTo;
}

public void Tick()
{
    tForm_Timer.Stop();

    bool bSuspended = (Program.UI.Suspended);
    Log(This(c_sTick, this.Name, Program.UI.Config.Log.StartForm_Timer);

    if (Program.UI.CurrentRoomba != null)
    {
        this.Check_If_Recording();
        this.Check_If_Executing();

        this.Check_Roomba_Object();

        if (!Program.UI.Suspended)
        {

    }
this.Check_Form_Connection();

//If frmConfig.Config.Do Battery Check is checked...
if (Program.UI.Config.Forms.StartForm.Battery_Check)
{
    Program.UI.Check_Battery(Program.UI.CurrentRoomba.Sensors);
}

this.lblFormTimer.Text = lblFormTimer.Text + c_sDot;
if (this.lblFormTimer.Text.Length > 19)
{
    this.lblFormTimer.Text = "";
}

if (bSuspended)
{
    this._Suspend();
}
else
{
    if (Program.UI.Open_For_Restart)
    {
        this._Start();

        Program.UI.Open_For_Restart = false;
    }
}

this.tForm_Timer.Interval = (Program.UI.Config.Forms.StartForm.Timer);
tForm_Timer.Start();

public void _Start()
{
    this._Stop(false, false);

    try
    {
        this.lblFormUpdated.Text = "Starting..";
        this.lblPoller.Text = "Starting..";
        this.lblCOMM_Port_Used.Text = Program.UI.Config.COMM.ConnectedTo;
        Program.UI.Start_Connection(false, true, this.lblVersion.Text); //we set up the log earlier so we
can capture form events.
        this.lblCOMM_Port_Used.Text = Program.UI.CurrentRoomba.IO.PortName;

        //Start Form Timer
        this.tForm_Timer.Interval = (Program.UI.Config.Polling.Frequency);
Log.This("Restarting Form Timer Interval: " + Program.UI.Config.Polling.Frequency.ToString(), this.Name, Program.UI.Config.Log.StartForm);
this.tForm_Timer.Start();

Program.UI.Statistics.ConnectedTime = new TimeSpan();
startToolStripMenuItem.Enabled = false;
stopToolStripMenuItem.Enabled = true;
loadToolStripMenuItem.Enabled = true; //Load Macro

} catch (Exception ex)
{
    MessageBox.Show(ex.Message);
}

GC.Collect();

public void _Stop(bool bUpdate_Indicator, bool bDestroyRoombaObj)
{
    loadToolStripMenuItem.Enabled = false; //Load Macro

    Log.This("Stop Roomba. ", this.Name, Program.UI.Config.Log.StartForm);
    Program.UI.Stop_Connection(bDestroyRoombaObj);

    if (bUpdate_Indicator)
    {
        this.lblIsCurrent.Text = frmStart.c_sStopped;
        this.lblIsCurrent.BackColor = Color.Transparent;

        this.lblFormUpdated.BackColor = Color.Transparent;
        this.lblFormUpdated.Text = c_sStopped;

        this.lblPoller.BackColor = Color.Transparent;
        this.lblPoller.Text = "No Obj";
    }

    GC.Collect();
}

public void _Suspend()
{
    if (Program.UI.CurrentRoomba.Automatic_Polling) {
        Program.UI.CurrentRoomba.Automatic_Polling = false; 
    }

    this.lblIsCurrent.Text = frmStart.c_sSuspended;
    this.lblIsCurrent.BackColor = Color.Yellow;

    this.lblFormUpdated.BackColor = Color.Yellow;
}
```csharp
this.lblFormUpdated.Text = c_sSuspended;
}

public void Check_Form_Connection()
{
    if (Program.UI.CurrentRoomba.Sensors != null)
    {
        DateTime dtNow = DateTime.Now;
        DateTime dtLastUpdated = Program.UI.CurrentRoomba.Sensors.LastUpdated;
        TimeSpan tsResult = dtNow - dtLastUpdated;
        double dForm_Updated = (dtNow - dtLastUpdated).TotalMilliseconds;

        this.FormUpdated = dForm_Updated;

        Log.This("FormUpdated: " + this.FormUpdated.ToString(), this.Name,
                   Program.UI.Config.Log.StartForm_Timer);

        if ((this.FormUpdated > 0) & (this.FormUpdated < 10000))
        {
            this.lblFormUpdated.Text = this.FormUpdated.ToString();
            m_iFormUpdated_DisplayLag = 0;

            if (this.FormUpdated < 500)
            {
                if (Program.UI.CurrentRoomba.New_State)
                {
                    this.lblFormUpdated.BackColor = Color.Green;
                }
                else
                {
                    this.lblFormUpdated.BackColor = Color.LightGreen;
                }
            }
            else
            {
                lblFormUpdated.BackColor = Color.Transparent;
            }
        }
        else
        {
            m_iFormUpdated_DisplayLag++; 
            if (m_iFormUpdated_DisplayLag >
                   Program.UI.Config.Forms.StartForm.FormUpdated_DisplayLag)
            {
                this.lblFormUpdated.Text = frmStart.c_sNoRead;
                this.lblFormUpdated.BackColor = Color.Red;
                this.lblChargeState.BackColor = Color.Transparent;
                this.lblChargeState.Text = null;

                Program.UI.Started = false;
            }
        }
    }
    Program.UI.Started = true;
}
```

public void Check_Roomba_Object()
{
    if (Program.UI.CurrentRoomba.New_State)
    {
        this.lblIsCurrent.ForeColor = Color.Black;
    }
    else
    {
        this.lblIsCurrent.ForeColor = Color.Blue;
    }

    Sensors snRoomba = Program.UI.CurrentRoomba.Sensors;

    if (Program.UI.CurrentRoomba.Polling)
    {
        this.lblPoller.BackColor = Color.Green;
        lblPoller.Text = Program.UI.CurrentRoomba.PollTicks.ToString();

        if (Program.UI.Started)
        {
            this.Check_If_Plugged_In(Program.UI.CurrentRoomba.Sensors);
        }
    }
    else
    {
        lblPoller.BackColor = Color.Red;
        lblPoller.Text = c_sOff;
    }

    if (snRoomba != null)
    {
        Log.This(c_sIsCurrent + snRoomba.IsCurrent.ToString() + " " +
        this.FormUpdated.ToString(), this.Name, Program.UI.Config.Log.StartForm_Timer);
    }

    if (snRoomba.IsCurrent)
    {
        if (snRoomba.LastUpdated != new DateTime())
        {
            this.SetIsCurrentConnected(snRoomba);
        }
    }
    else
    {
        this.toolTip1.SetToolTip(lblIsCurrent, null);
        lblIsCurrent.BackColor = Color.Transparent;
    }
}
```csharp
else
{
    lblIsCurrent.Text = frmStart.c_sNoRead;
    lblIsCurrent.BackColor = Color.Red;
}

private void setIsCurrentConnected(Sensors snRoomba)
{
    this.toolTip1.SetToolTip(lblIsCurrent, null);
    this.lblIsCurrent.Text = snRoomba.ParseTime.Ticks.ToString();

    if (Program.UI.CurrentRoomba.New_State)
    {
        this.lblIsCurrent.BackColor = Color.Green;
    }
    else
    {
        this.lblIsCurrent.BackColor = Color.LightGreen;
    }

    this.toolTip1.SetToolTip(lblIsCurrent, snRoomba.LastUpdated.ToString());
    this.lblLastCurrent.Text = snRoomba.LastUpdated.ToString();
}

private void RecordFlasher(bool bRecording)
{
    if (bRecording)
    {
        this.lblMacroExecuting.Enabled = false;
        this.lblRecording.Text = c_sRecording;

        m_bRecordingFlasher = !m_bRecordingFlasher;
        if (m_bRecordingFlasher) { this.lblRecording.BackColor = Color.Green; } else { this.lblRecording.BackColor = Color.Transparent; }
    }
    else
    {
        this.lblMacroExecuting.Enabled = true;
    }
}

private void ExecuteFlasher(bool bExecuting)
{
    if (bExecuting)
    {
        this.lblRecording.Enabled = false;
        this.lblMacroExecuting.Text = c_sExecuting;

        m_bExecutingFlasher = !m_bExecutingFlasher;
        if (m_bExecutingFlasher) { this.lblMacroExecuting.BackColor = Color.Green; } else { this.lblMacroExecuting.BackColor = Color.Transparent; }
    }
    else
    {
        this.lblMacroExecuting.Enabled = true;
    }
}
if (m_bExecutingFlasher) { this.lblMacroExecuting.BackColor = Color.Green; } else {
    this.lblMacroExecuting.BackColor = Color.Transparent;
}
else {
    this.lblRecording.Enabled = true;
    this.lblMacroExecuting.BackColor = Color.Transparent;
}
}

private void ToggleRecordMacro()
{
    this.lblRecording.BackColor = Color.Transparent;
    this.lblRecording.Text = "";

    if (Program.UI.CurrentRoomba != null)
    {
        if (!Program.UI.CurrentRoomba.Macro.Recording)
        {
            try
            {
                Program.UI.CurrentRoomba.Macro.Record();

                this.lblRecording.BackColor = Color.Green;
                this.lblRecording.Text = c_sRecording;
            }
            catch (Exception ex)
            {
                if (ex.GetType() == typeof(RoombaSCI.MacroException))
                {
                    this.lblRecording.Text = ex.Message;
                    this.lblRecording.BackColor = Color.Red;
                }
                }
            else
            
                Program.UI.CurrentRoomba.Macro.Stop();
                this.lblRecording.BackColor = Color.Transparent;
                this.lblRecording.Text = c_sNotRecording;
            }
        }
        }
    }

    Log.This("Macro Recording " + Program.UI.CurrentRoomba.Macro.Recording.ToString(),
        this.Name, Program.UI.Config.Log.StartForm);
    }
}
private void ToggleExecuteMacro()
{
    this.lblMacroExecuting.BackColor = Color.Transparent;
    this.lblMacroExecuting.Text = "";
if (!Program.UI.CurrentRoomba.Macro.Executing)
{
    try
    {
        Program.UI.CurrentRoomba.Macro.Execute();
        this.lblMacroExecuting.BackColor = Color.Green;
        this.lblMacroExecuting.Text = c_sExecuting;
    }
    catch (Exception ex)
    {
        if (ex.GetType() == typeof(RoombaSCI.MacroException))
        {
            this.lblMacroExecuting.Text = ex.Message;
            this.lblMacroExecuting.BackColor = Color.Red;
        }
    }
}
else
{
    Program.UI.CurrentRoomba.Macro.Stop();
    this.lblMacroExecuting.BackColor = Color.Transparent;
    this.lblMacroExecuting.Text = c_sNotExecuting;
}
Log.This("Macro Recording " + Program.UI.CurrentRoomba.Macro.Recording.ToString(),
this.Name, Program.UI.Config.Log.StartForm);
}

private void PluggedIn(bool bPluggedIn)
{
    if (bPluggedIn)
    {
        m_bPluggedInFlasher = !m_bPluggedInFlasher;
        if (m_bPluggedInFlasher) { this.lblChargeState.BackColor = Color.Yellow; } else { this.lblChargeState.BackColor = Color.Transparent; }
    }
    else
    {
        this.lblChargeState.BackColor = Color.Transparent;
    }
}
private void PluggedIn(bool bPluggedIn, bool noFlash)
{
    if (bPluggedIn)
    {
        if (!noFlash)
        {
            m_bPluggedInFlasher = !m_bPluggedInFlasher;
        }
    }
    else
    {
        m_bPluggedInFlasher = bPluggedIn;
    }

    if (m_bPluggedInFlasher) { this.lblChargeState.BackColor = Color.Yellow; } else {
        this.lblChargeState.BackColor = Color.Transparent;
    }

    else
    {
        this.lblChargeState.BackColor = Color.Transparent;
        this.lblChargeState.Text = Charge_State_Description.Indeterminate;
    }

private void Check_If_Plugged_In(Sensors rsCurrentSensorPoll)
{
    byte chargeState = rsCurrentSensorPoll.Packet.Charging_State;
    this.PopulateChargeState(chargeState, rsCurrentSensorPoll);

    Log.This(c_sChargeState + this.lblChargeState.Text + c_sVoltage +
    rsCurrentSensorPoll.Packet.Voltage.ToString() + c_sCurrent +
    this.Name, Program.UI.Config.Log.StartForm_Charging);
}

private void Check_If_Recording()
{
    if (Program.UI.CurrentRoomba.Macro != null)
    {
        this.RecordFlasher(Program.UI.CurrentRoomba.Macro.Recording);
    }
    else
    {
        this.RecordFlasher(false);
    }
}

private void Check_If_Executing()
{
    if (Program.UI.CurrentRoomba.Macro != null)
    {
        this.ExecuteFlasher(Program.UI.CurrentRoomba.Macro.Executing);
        if (Program.UI.CurrentRoomba.Macro.Finished)
        {
            this.lblMacroExecuting.Text = c_sFinished;
            this.ExecuteFlasher(false);
        }
    }
    else
    {
        this.ExecuteFlasher(false);
    }
public void PopulateChargeState(byte byChargingState, Sensors rsCurrentSensorPoll)
{
    bool bPluggedIn = false;

    Application.DoEvents();

    if (byChargingState == Charging_State.Not_Charging)
    {
        this.lblChargeState.Text = Charge_State_Description.Not_Charging;
        bPluggedIn = false;
    }
    else if (byChargingState == Charging_State.Charging_Recovery)
    {
        this.lblChargeState.Text = Charge_State_Description.Charging_Recovery;
        bPluggedIn = true;
    }
    else if (byChargingState == Charging_State.Charging)
    {
        this.lblChargeState.Text = Charge_State_Description.Charging;
        bPluggedIn = true;
    }
    else if (byChargingState == Charging_State.Trickle_Charging)
    {
        this.lblChargeState.Text = Charge_State_Description.Trickle_Charging;
        bPluggedIn = true;
    }
    else if (byChargingState == Charging_State.Waiting)
    {
        this.lblChargeState.Text = Charge_State_Description.Waiting;
        bPluggedIn = true;
    }
    else if (byChargingState == Charging_State.Charging_Error)
    {
        this.lblChargeState.Text = Charge_State_Description.Charging_Error;
        bPluggedIn = true;
    }
    else
    {
        this.lblChargeState.Text = "Error";
        bPluggedIn = false;
    }

    this.PluggedIn(bPluggedIn);

    if (rsCurrentSensorPoll.Packet.Current < 0)
    {
        this.lblChargeState.Text = Charge_State_Description.Plugged_In;
        this.PluggedIn(true, true);
    }
namespace roomba_term
{
    public partial class frmDrive : roomba_term.frmMenu
    {
        delegate void SetTextCallback();
        private Thread timerThread = null;

        public frmDrive()
        {
            InitializeComponent();
        }

        #region Member variables
        private int m_iCurrentSpeed = 0;
        private int m_iCurrentAngle = 32768;
        private bool m_bPluggedInFlasher = false;
        #endregion

        #region Properties
        private bool p_bDebugMode;
        public bool DebugMode
        {
            get
            {
                return (this.p_bDebugMode);
            }
            set
            {
                this.p_bDebugMode = value;
            }
        }
        #endregion

        #region Event Handlers
        #region Form
        private void frmDrive_Load(object sender, EventArgs e)
        {
            frmDriveControl();
            using(System.Drawing;
            using(System.Threading;
            using(System.Windows.Forms;
            using(System.Collections.Generic;
            using Kevin_Logging;
            using RoombaSCI;

        #endregion
        #endregion
this.lblError.Visible = chkShowErrors.Checked;
this.Form_Timer = new RoombaSCI.Timer();
this.Form_Timer.Period = (int)this.udFormDisplay.Value;
this.Form_Timer.Tick += new EventHandler(OnTimedEvent);
this.Form_Timer.Start();
this.PASSIVE(true);
this.KeyPreview = true;
}
private void frmDrive_FormClosing(object sender, FormClosingEventArgs e)
{
    this.Form_Timer.Stop();
    Application.DoEvents();
    this.driveToolStripMenuItem.Enabled = true;
    Program.Menu_Cache.Remove(this.Handle);
    Program.UI.BorgMyMenu(this);
}
private void frmDrive_KeyDown(object sender, KeyEventArgs e)
{
    this.GetKey(e);
}
#endregion
#region Timers
private void OnTimedEvent(object sender, System.EventArgs e)
{
    this.Form_Timer.Stop();
    this.Display_SensorInfo();
    this.getThesisSensors();
    this.Form_Timer.Start();
}
#endregion
#region CheckBoxen
private void chkDebugConnection_CheckedChanged(object sender, EventArgs e)
{
    this.DebugMode = this.chkDebugConnection.Checked;
}
private void chkShowErrors_CheckedChanged(object sender, EventArgs e)
{
    this.lblError.Visible = chkShowErrors.Checked;
}
private void tMain_Brush_CheckedChanged(object sender, EventArgs e)
byte bMotorSettings;
if (!tMain_Brush.Checked)
{
    bMotorSettings = Convert.ToByte(Motor.Main_Brush.Off);
}
else
{
    bMotorSettings = Convert.ToByte(Motor.Main_Brush.On);
}

bool bSuccess = Program.UI.CurrentRoomba.Motor_Action(bMotorSettings);

private void tVacuum_CheckedChanged(object sender, EventArgs e)
{
    byte bMotorSettings;
    if (!this.tVacuum.Checked)
    {
        bMotorSettings = Convert.ToByte(Motor.Vacuum.Off);
    }
    else
    {
        bMotorSettings = Convert.ToByte(Motor.Vacuum.On);
    }

    bool bSuccess = Program.UI.CurrentRoomba.Motor_Action(bMotorSettings);
}

private void tSideBrush_CheckedChanged(object sender, EventArgs e)
{
    byte bMotorSettings;
    if (!this.tSideBrush.Checked)
    {
        bMotorSettings = Convert.ToByte(Motor.Side_Brush.Off);
    }
    else
    {
        bMotorSettings = Convert.ToByte(Motor.Side_Brush.On);
    }

    bool bSuccess = Program.UI.CurrentRoomba.Motor_Action(bMotorSettings);
}

#endregion
#region MenuItems

private void restartConnectionToolStripMenuItem_Click(object sender, EventArgs e)
{
    this.Form_Timer.Start();
}

private void stopToolStripMenuItem_Click(object sender, EventArgs e)
{
    this.Form_Timer.Stop();
}  
private void clearToolStripMenuItem_Click(object sender, EventArgs e)
{
    this.Clear();
}

private void PASSIVEToolStripMenuItem_Click(object sender, EventArgs e)
{
    this.PASSIVE(true);
}

private void SAFEToolStripMenuItem_Click(object sender, EventArgs e)
{
    this.SAFE(true);
}

private void FULLToolStripMenuItem_Click(object sender, EventArgs e)
{
    this.FULL(true);
}

#endregion
#region Numeric UpDowns

private void udSpeed_MouseDoubleClick(object sender, MouseEventArgs e)
{
    if (this.DebugMode)
    {
        try
        {
            Program.UI.CurrentRoomba.Drive((Velocity)udSpeed.Value, (int)udRotate.Value);
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
        }
    }
}

private void udFormDisplay_ValueChanged(object sender, EventArgs e)
{
    this.Form_Timer.Period = (int)this.udFormDisplay.Value;
}

#endregion
#region Labels

private void lPassive_DoubleClick(object sender, EventArgs e)
{
    this.PASSIVE(true);
}

private void lSafe_DoubleClick(object sender, EventArgs e)
{
this.SAFE(true);
}
private void lFull_DoubleClick(object sender, EventArgs e)
{
    this.FULL(true);
}

private void lPassive_MouseHover(object sender, EventArgs e)
{
    this.lPassive.Font = new Font("Microsoft Sans Serif", 8, System.Drawing.FontStyle.Bold);
}
private void lSafe_MouseHover(object sender, EventArgs e)
{
    this.lSafe.Font = new Font("Microsoft Sans Serif", 8, System.Drawing.FontStyle.Bold);
}
private void lFull_MouseHover(object sender, EventArgs e)
{
    this.lFull.Font = new Font("Microsoft Sans Serif", 8, System.Drawing.FontStyle.Bold);
}

private void lPassive_MouseLeave(object sender, EventArgs e)
{
    this.lPassive.Font = new Font("Microsoft Sans Serif", 8, System.Drawing.FontStyle.Regular);
}
private void lSafe_MouseLeave(object sender, EventArgs e)
{
    this.lSafe.Font = new Font("Microsoft Sans Serif", 8, System.Drawing.FontStyle.Regular);
}
private void lFull_MouseLeave(object sender, EventArgs e)
{
    this.lFull.Font = new Font("Microsoft Sans Serif", 8, System.Drawing.FontStyle.Regular);
}

#endregion
#region Hardware UI
private void btnPower_Click(object sender, EventArgs e)
{
    Program.UI.CurrentRoomba.Execute(OpCode.Power);
}
private void btnSpot_Click(object sender, EventArgs e)
{
    bool bSuccess = Program.UI.CurrentRoomba.Execute(OpCode.Spot);
}
#endregion
#endregion Buttons
private void btnApply_Click(object sender, EventArgs e)
{
    Program.UI.Config.Save();
}
#region Hardware UI
private void btnClean_Click(object sender, EventArgs e)
{
    bool bSuccess = Program.UI.CurrentRoomba.Execute(OpCode.Clean);
}

private void btnMax_Click(object sender, EventArgs e)
{
    bool bSuccess = Program.UI.CurrentRoomba.Execute(OpCode.Max);
}

#region Button BugFix
private void tSideBrush_Leave(object sender, EventArgs e)
{
    tabControl1.Select(); //Set focus to a control that won't eat my cursor keydown events
    Application.DoEvents();
}

private void tMain_Brush_Leave(object sender, EventArgs e)
{
    tabControl1.Select(); //Set focus to a control that won't eat my cursor keydown events
    Application.DoEvents();
}

private void tVacuum_Leave(object sender, EventArgs e)
{
    tabControl1.Select(); //Set focus to a control that won't eat my cursor keydown events
    Application.DoEvents();
}

#endregion

#region
public void OFF(bool setMode)
{
    if (setMode) { Program.UI.CurrentRoomba.SetMode(SCI_Mode.Off); }

    this.lOff.BackColor = Color.Yellow;
    this.lSafe.BackColor = Color.Transparent;
    this.lPassive.BackColor = Color.Transparent;
    this.lFull.BackColor = Color.Transparent;
    this.EnableButtons(false);
}

public void PASSIVE(bool setMode)
{
    if (Program.UI.CurrentRoomba != null)
    {
        if (setMode) { Program.UI.CurrentRoomba.SetMode(SCI_Mode.Passive); }
    }
}
this.EnableButtons(true);
this.Set_Buttons(false);

this.lPassive.BackColor = Color.Cyan;
this.lSafe.BackColor = Color.Transparent;
this.lFull.BackColor = Color.Transparent;
this.lOff.BackColor = Color.Transparent;
}

public void SAFE(bool setMode)
{
  if(Program.UI.CurrentRoomba != null)
  {
    if (setMode) { Program.UI.CurrentRoomba.SetMode(SCI_Mode.Passive); }

    Application.DoEvents();
    if (setMode) { Program.UI.CurrentRoomba.SetMode(SCI_Mode.Safe); }

    this.EnableButtons(true);
    this.Set_Buttons(false);

    this.lSafe.BackColor = Color.Cyan;

    this.lFull.BackColor = Color.Transparent;
    this.lPassive.BackColor = Color.Transparent;
    this.lOff.BackColor = Color.Transparent;
  }
}

public void FULL(bool setMode)
{
  if (Program.UI.CurrentRoomba != null)
  {
    if (setMode) { Program.UI.CurrentRoomba.SetMode(SCI_Mode.Passive); }

    Application.DoEvents();
    if (setMode) { Program.UI.CurrentRoomba.SetMode(SCI_Mode.Full); }

    this.EnableButtons(true);
    this.Set_Buttons(false);

    this.lFull.BackColor = Color.Cyan;

    this.lSafe.BackColor = Color.Transparent;
    this.lPassive.BackColor = Color.Transparent;
    this.lOff.BackColor = Color.Transparent;
  }
}

public void HandleKeys(Keys kDown)
{
this.pRotateLeft.BackColor = Color.Transparent;
this.pRotateRight.BackColor = Color.Transparent;
this.pFWD.BackColor = Color.Transparent;
this.pBack.BackColor = Color.Transparent;

try
{
    switch (kDown)
    {
    case Keys.Right:
        this.pRotateRight.BackColor = Color.Blue;
        if (m_iCurrentAngle == 32768) { m_iCurrentAngle = 0; }

        if ((m_iCurrentAngle - (int)this.udRotateStep.Value) > Radius.Maximum_Right)
        {
            if (m_iCurrentSpeed >= 0)
            {
                m_iCurrentAngle -= (int)this.udRotateStep.Value;
            }
            else
            {
                m_iCurrentAngle += (int)this.udRotateStep.Value;
            }
        }
        break;

    case Keys.Left:
        this.pRotateLeft.BackColor = Color.Blue;
        if (m_iCurrentAngle == 32768) { m_iCurrentAngle = 0; }

        if (m_iCurrentSpeed >= 0)
        {
            m_iCurrentAngle += (int)this.udRotateStep.Value;
        }
        else
        {
            m_iCurrentAngle -= (int)this.udRotateStep.Value;
        }
        break;

    case Keys.Up:
        this.pFWD.BackColor = Color.Blue;
        if ((m_iCurrentSpeed + (int)this.udSpeedStep.Value) < Velocity.Maximum_Forward)
        {
            m_iCurrentSpeed += (int)this.udSpeedStep.Value;
        }

        if (chkAutoStraighten.Checked)
        {
        }
m_iCurrentAngle = 32768;
}

break;

case Keys.Down:
    this.pBack.BackColor = Color.Blue;
    
    if ((m_iCurrentSpeed - (int)this.udSpeedStep.Value) > Velocity.Maximum_Reverse)
    {
        this.udSpeed.Font = new Font("Microsoft Sans Serif", 8,
System.Drawing.FontStyle.Regular);
        m_iCurrentSpeed -= (int)this.udSpeedStep.Value;
    }
    else
    {
        this.udSpeed.Font = new Font("Microsoft Sans Serif", 8,
System.Drawing.FontStyle.Bold);
    }

    if (chkAutoStraighten.Checked)
    {
        m_iCurrentAngle = 32768;
    }

    break;

case Keys.Space:
    m_iCurrentSpeed = 0;
    
    if (chkAutoStraighten.Checked)
    {
        m_iCurrentAngle = 32768;
    }

    break;

this.udSpeed.Value = m_iCurrentSpeed;
this.udRotate.Value = m_iCurrentAngle;

Log.This("Drive Form: Velocity: " + m_iCurrentSpeed.ToString() + "  Radius: " +
m_iCurrentAngle.ToString(), this.Name, Program.UI.Config.Log.DriveForm);
Program.UI.CurrentRoomba.Drive((Velocity)m_iCurrentSpeed, (Radius)(m_iCurrentAngle));

} catch (Exception ex)
{
    MessageBox.Show(ex.Message);
}

private void GetKey(KeyEventArgs e)
if (Program.UI.Started)
{
    Keys kDown = e.KeyCode;
    this.HandleKeys(kDown);
    e.Handled = true;
}
else
{
}

public void LockDown_Form(bool Lock)
{
    this.EnableButtons(Lock);
}
public void EnableButtons(bool bEnabled)
{
    this.tSideBrush.Enabled = bEnabled;
    this.tMain_Brush.Enabled = bEnabled;
    this.tVacuum.Enabled = bEnabled;
}
public void Set_Buttons(bool bChecked)
{
    this.tSideBrush.Checked = bChecked;
    this.tMain_Brush.Checked = bChecked;
    this.tVacuum.Checked = bChecked;
}
public void Display_SensorInfo()
{
    this.timerThread = null;

    if (Program.UI.CurrentRoomba != null)
    {
        if (Program.UI.CurrentRoomba.Sensors != null)
        {
            if (Program.UI.CurrentRoomba.Sensors.LastUpdated != new DateTime())
            {
                this.timerThread = new Thread(new ThreadStart(this.UpdateForm));
                this.timerThread.Start();
            }
            else
            {
                if (chkAccurateSensors.Checked)
                {
                    this.Clear();
                }
            }
        }
    }
    else
    {
    }
}
private void UpdateForm()
{
    if (this.InvokeRequired)
    {
        try
        {
            SetTextCallback d = new SetTextCallback(UpdateForm);
            this.Invoke(d, new object[] { });
        }
        catch (Exception ex)
        {
        }
    }
    else
    {
        if (Program.UI.CurrentRoomba != null)
        {
            if (Program.UI.CurrentRoomba.Sensors.LastUpdated != new DateTime())
            {
                if (Program.UI.CurrentRoomba.Sensors.IsCurrent)
                {
                    if (chkDebugConnection.Checked)
                    {
                        this.lblSensorParse.Text = lblSensorParse.Text + "c";
                        this.lblError.Text = "";
                    }
                    this.Check_If_Plugged_In(Program.UI.CurrentRoomba.Sensors);
                }
                else
                {
                    if (chkDebugConnection.Checked)
                    {

this.lblSensorParse.Text = lblSensorParse.Text + "x";
this.lblError.Text = "Packet Data not Current";
}
}

this.Show_SensorData(Program.UI.CurrentRoomba.Sensors);
this.CheckMode(Program.UI.CurrentRoomba);

else
{
    if (chkDebugConnection.Checked)
    {
        this.lblSensorParse.Text = lblSensorParse.Text + "-";
        this.lblError.Text = "No new Packet Data";
    }
}
}
else
{
    if (chkDebugConnection.Checked)
    {
        this.lblSensorParse.Text = lblSensorParse.Text + "X";
        this.lblError.Text = "No Roomba Started or Connected";
    }
}

if (this.lblSensorParse.Text.Length > 35)
{
    this.lblSensorParse.Text = this.lblSensorParse.Text.Substring(1,
    this.lblSensorParse.Text.Length - 1);
}

private void CheckMode(Roomba_Poller roomba_Poller)
{
    switch (roomba_Poller.Current_Mode)
    {
        case SCI_Mode.Off:
            this.OFF(false);
            break;

        case SCI_Mode.Safe:
            this.SAFE(false);
            break;

        case SCI_Mode.Full:
            this.FULL(false);
            break;
    }
}
break;

case SCI_Mode.Passive:
    this.PASSIVE(false);
    break;
}
}

private void Check_If_Plugged_In(RoombaSCI.Sensors rsCurrentSensorPoll)
{
    byte chargeState = rsCurrentSensorPoll.Packet.Charging_State;
    this.PopulateChargeState(chargeState);
}

//Populates all sensor indicators on the form from Roomba's Sensor structure
private void Show_SensorData(Sensors rsCurrentSensorPoll)
{
    try
    {
        Program.UI.SetPictureBox(rsCurrentSensorPoll(Packet.Buttons.Power, this.pPower);
        Program.UI.SetPictureBox(rsCurrentSensorPoll(Packet.Buttons.Spot, this.pSpot);
        Program.UI.SetPictureBox(rsCurrentSensorPoll(Packet.Buttons.Clean, this.pClean);
        Program.UI.SetPictureBox(rsCurrentSensorPoll(Packet.Buttons.Max, this.pMax);
        Program.UI.SetPictureBox(rsCurrentSensorPoll(Packet.Cliff.Left, this.pCliffLeft);
        Program.UI.SetPictureBox(rsCurrentSensorPoll(Packet.Cliff.Right, this.pCliffRight);
        Program.UI.SetPictureBox(rsCurrentSensorPoll(Packet.Cliff.FrontLeft, this.pCliffFrontLeft);
        Program.UI.SetPictureBox(rsCurrentSensorPoll(Packet.Cliff.FrontRight, this.pCliffFrontRight);
        Program.UI.SetPictureBox(rsCurrentSensorPoll(Packet.Bump.Left, this.pBump_Left);
        Program.UI.SetPictureBox(rsCurrentSensorPoll(Packet.Bump.Right, this.pBump_Right);
        Program.UI.SetPictureBox(rsCurrentSensorPoll(Packet.WheelDrop.Left, this.pWheelDrop_Left);
        Program.UI.SetPictureBox(rsCurrentSensorPoll(Packet.WheelDrop.Right, this.pWheelDrop_Right);
        Program.UI.SetPictureBox(rsCurrentSensorPoll(Packet.WheelDrop.Caster, this.pWheelDrop_Caster);
        Program.UI.SetPictureBox(rsCurrentSensorPoll(Packet.OverCurrent.Left_Wheel, this.pDriveLeft_Overcurrent);
        Program.UI.SetPictureBox(rsCurrentSensorPoll(Packet.OverCurrent.Right_Wheel, this.pDriveRight_OverCurrent);
        Program.UI.SetPictureBox(rsCurrentSensorPoll(Packet.OverCurrent.Main_Brush, this.pMainBrush_Overcurrent);
        Program.UI.SetPictureBox(rsCurrentSensorPoll(Packet.OverCurrent.Side_Brush, this.pSideBrush_Overcurrent);
        Program.UI.SetPictureBox(rsCurrentSensorPoll(Packet.OverCurrent.Vacuum, this.pVacuum_OverCurrent);
Program.UI.SetPictureBox(rsCurrentSensorPoll.Packet.Virtual_Wall, this.pVirtual_Wall);
Program.UI.SetPictureBox(rsCurrentSensorPoll.Packet.Wall, this.pWallDetect);

Program.UI.SetPictureBox(rsCurrentSensorPoll.Packet.Buttons.Power, this.pPower);
Program.UI.SetPictureBox(rsCurrentSensorPoll.Packet.Buttons.Spot, this.pSpot);
Program.UI.SetPictureBox(rsCurrentSensorPoll.Packet.Buttons.Clean, this.pClean);
Program.UI.SetPictureBox(rsCurrentSensorPoll.Packet.Buttons.Max, this.pMax);
}
catch (Exception ex)
{
    this.lblError.Text = ex.Message;
}
}

public void PopulateChargeState(byte byChargingState)
{
    if (byChargingState == Charging_State.Not_Charging)
    {
        this.PluggedIn(false);
    }
    else if (byChargingState == Charging_State.Charging_Recovery)
    {
        this.PluggedIn(true);
    }
    else if (byChargingState == Charging_State.Charging)
    {
        this.PluggedIn(true);
    }
    else if (byChargingState == Charging_State.Trickle_Charging)
    {
        this.PluggedIn(true);
    }
    else if (byChargingState == Charging_State.Waiting)
    {
        this.PluggedIn(true);
    }
    else if (byChargingState == Charging_State.Charging_Error)
    {
        this.PluggedIn(true);
    }
    else
    {
    }
}

private void PluggedIn(bool bPluggedIn)
{
    if (!((Program.UI.CurrentRoomba.Current_Mode == SCI_Mode.Off)))
    {
        this.EnableButtons(!bPluggedIn);
    }
private void Clear()
{
    this.pBump_Left.BackColor = Color.Transparent;
    this.pBump_Right.BackColor = Color.Transparent;
    this.pCliffLeft.BackColor = Color.Transparent;
    this.pCliffFrontLeft.BackColor = Color.Transparent;
    this.pCliffFrontRight.BackColor = Color.Transparent;
    this.pCliffRight.BackColor = Color.Transparent;
    this.pVirtual_Wall.BackColor = Color.Transparent;
    this.pWallDetect.BackColor = Color.Transparent;
    this.pWheelDrop_Caster.BackColor = Color.Transparent;
    this.pWheelDrop_Left.BackColor = Color.Transparent;
    this.pWheelDrop_Right.BackColor = Color.Transparent;
    this.pDriveLeft_Overcurrent.BackColor = Color.Transparent;
    this.pDriveRight_OverCurrent.BackColor = Color.Transparent;
    this.lblDirt_Detect_Left.BackColor = Color.Transparent;
    this.lblDirt_Detect_Right.BackColor = Color.Transparent;
    this.pMainBrush_Overcurrent.BackColor = Color.Transparent;
    this.pSideBrush_Overcurrent.BackColor = Color.Transparent;
    this.pVacuum_OverCurrent.BackColor = Color.Transparent;
}

private void chkAccurateSensors_CheckedChanged(object sender, EventArgs e)
{
}

#region Sam's Thesis
public const int Straight = 32768;
private DateTime ThesisStartTime = DateTime.Now;
private TimeSpan RoombaTimeSpan;
private bool thesisGo = false;
private double thesisConversionadjustment = 0;
private double metersToFeetPerSec = 3.2808; //1 m/s = 3.2808 feet/s
private bool calibrate = false;
private int wentRightCount = 0;
private int wentLeftCount = 0;
private int wentBothCount = 0;
private int adjustangle = 0;
private bool reverseToClear = false;
#endregion
private bool wentright = false;
private bool wentleft = false;
private bool wentBoth = false;

#region

private void getThesisSensors()
{
    if (Program.UI.CurrentRoomba != null)
    {
        {
            RoombaTimeSpan = DateTime.Now - ThesisStartTime;
            thesisRoombaStop();
            thesisRoombaReverse(true);
            wentBoth = true;
        }
        {
            RoombaTimeSpan = DateTime.Now - ThesisStartTime;
            thesisRoombaStop();
            thesisRoombaReverse(true);
            wentleft = true;
        }
        {
            RoombaTimeSpan = DateTime.Now - ThesisStartTime;
            thesisRoombaStop();
            thesisRoombaReverse(true);
            wentright = true;
        }
        {
            RoombaTimeSpan = DateTime.Now - ThesisStartTime;
            thesisRoombaStop();
            calibrate = false;
        }
        {
            RoombaTimeSpan = DateTime.Now - ThesisStartTime;
            thesisRoombaStop();
            calibrate = false;
        }
        if (reverseToClear)
        {
            {
                thesisRoombaStop();
            }
        }
    }
#endregion
RoombaCalulations(RoombaTimeSpan.Seconds);
thesisRoombaStraight();
reverseToClear = false;
}

if (Program.UI.CurrentRoomba.Sensors.Packet.Virtual_Wall)
{
thesisRoombaStop();
}

private void btnthesisGo_Click(object sender, EventArgs e)
{
this.FULL(true);
thesisGo = true;
ThesisStartTime = DateTime.Now;
thesisRoombaStraight();
}

#region Thesis Roomba Calculations

private void RoombaCalulations(int timeinSecs)
{
    double tempNum = .500 * metersToFeetPerSec * timeinSecs;
    if (wentright)
    {
        wentRightCount += 50;
        m_iCurrentAngle = 1;
        wentright = false;
        adjustangle = wentRightCount;
    }
    else if (wentleft)
    {
        wentLeftCount += 50;
        m_iCurrentAngle = -1;
        wentleft = false;
        adjustangle = wentLeftCount;
    }
    else if (wentBoth)
    {
        m_iCurrentAngle = -1;
        wentBoth = false;
        adjustangle = 1000;
    }
}
Log.This("The Adjusted Angle was: " + adjustangle);

m_iCurrentSpeed = 100;
Program.UI.CurrentRoomba.Drive((Velocity)m_iCurrentSpeed, (Radius)(m_iCurrentAngle));
Thread.Sleep(adjustangle);
thesisRoombaStop();
//MessageBox.Show("The time in Seconds is: " + timeinSecs + " Should have gone: " + tempNum + " Based on adjusted number: " + thesisConversionadjustment * timeinSecs);
}
#endif

#region Thesis Roomba Commands

private void thesisRoombaStraight(int timetoRun)
{
    m_iCurrentAngle = Straight;
m_iCurrentSpeed = 300;
Program.UI.CurrentRoomba.Drive((Velocity)m_iCurrentSpeed, (Radius)(m_iCurrentAngle));
Thread.Sleep(timetoRun);
}

private void thesisRoombaReverse(bool isTrue)
{
    m_iCurrentAngle = Straight;
m_iCurrentSpeed = -100;
Program.UI.CurrentRoomba.Drive((Velocity)m_iCurrentSpeed, (Radius)(m_iCurrentAngle));
reverseToClear = isTrue;
}

private void thesisRoombaStraight()
{
    m_iCurrentAngle = Straight;
m_iCurrentSpeed = 200;
Program.UI.CurrentRoomba.Drive((Velocity)m_iCurrentSpeed, (Radius)(m_iCurrentAngle));
}

private void thesisRoombaStop()
{
    m_iCurrentAngle = Straight;
m_iCurrentSpeed = 0;
Program.UI.CurrentRoomba.Drive((Velocity)m_iCurrentSpeed, (Radius)(m_iCurrentAngle));
}
#endif

private void btnCalibrateFloor_Click(object sender, EventArgs e)
{
    this.FULL(true);
calibrate = true;
ThesisStartTime = DateTime.Now;
thesisRoombaStraight();

#endregion

private void thesis_textchanged_calibration(object sender, EventArgs e)
{
    try
    {
        double tempNumb = double.Parse(txbxactualDistance.Text);
        thesisConversionadjustment = tempNumb / RoombaTimeSpan.Seconds;
    }
    catch
    {
    }
}

private void btnReset_Click(object sender, EventArgs e)
{
    adjustangle = 0;
    wentBothCount = 0;
    wentleft = false;
    wentright = false;
    wentRightCount = 0;
    wentLeftCount = 0;
}
namespace roomba_term
{
    // This form will write to config, and tell it to save
    public partial class frmConfig : roomba_term.frmMenu
    {
        public frmConfig()
        {
            InitializeComponent();
        }
    }

    #region Events
    #region Form

    private void Config_Load(object sender, EventArgs e)
    {
        this.cCOM_Port.Items.Clear();
        this.cCOM_Port.Text = "";
        this.cCOM_Port.Items.AddRange(Program.UI.GetPorts().ToArray());

        Program.UI.Config.Forms.StartForm.Battery_Check =
        this.chkStart_Form_Battery_Check.Checked;

        this.SyncToFile();
    }

    private void frmConfig_FormClosing(object sender, FormClosingEventArgs e)
    {
    }

    private void frmConfig_FormClosed(object sender, FormClosedEventArgs e)
    {
        this.configToolStripMenuItem.Enabled = true;
        Program.Menu_Cache.Remove(this.Handle);
        Program.UI.BorgMyMenu(this);
    }

    #endregion
    #endregion

    #region Comboboxes
private void cCOM_Port_SelectedIndexChanged(object sender, EventArgs e)
{
    Program.UI.Config.COMM.ConnectedTo = this.cCOM_Port.Text;
}
#endregion
#region Checkboxes
private void chkReadSensorObj_CheckedChanged(object sender, EventArgs e)
{
}
private void chkPollSensors_CheckedChanged(object sender, EventArgs e)
{
    Program.UI.Config.Polling.Sensors = this.chkPollSensors.Checked;
    Program.UI.CurrentRoomba.Automatic_Polling = this.chkPollSensors.Checked;
    this.udPollFrequency.Enabled = this.chkPollSensors.Checked;
}
private void chkStart_Form_Battery_Check_CheckedChanged(object sender, EventArgs e)
{
    Program.UI.Config.Forms.StartForm.Battery_Check =
    this.chkStart_Form_Battery_Check.Checked;
}
//Logging
private void chkRoombaUILog_CheckedChanged(object sender, EventArgs e)
{
    Program.UI.Config.Log.RoombaUI = this.chkRoombaUILog.Checked;
}
private void chkStartFormTimerLog_CheckedChanged(object sender, EventArgs e)
{
    Program.UI.Config.Log.StartForm_Timer = this.chkStartFormTimerLog.Checked;
}
private void chkStartFormChargeLog_CheckedChanged(object sender, EventArgs e)
{
    Program.UI.Config.Log.StartForm_Charging = this.chkStartFormChargeLog.Checked;
}
private void chkLogDriveActions_CheckedChanged(object sender, EventArgs e)
{
    Program.UI.Config.Log.DriveForm = this.chkLogDriveActions.Checked;
}
private void chkRoombaPollerLog_CheckedChanged(object sender, EventArgs e)
{
    Program.UI.Config.Log.Roomba_Poller = this.chkRoombaPollerLog.Checked;
    Program.UI.CurrentRoomba.Poller_LogPermission =
    Program.UI.Config.Log.Roomba_Poller;
}
private void chkRoombaSCILog_CheckedChanged(object sender, EventArgs e)
{
Program.UI.Config.Log.LogSCICommands = chkRoombaSCILOG.Checked;
Program.UI.CurrentRoomba.LogSCICommands = Program.UI.Config.Log.LogSCICommands;
}
private void chkLogRoombaIO_CheckedChanged(object sender, EventArgs e)
{
    Program.UI.Config.Log.Roomba_IO = this.chkLogRoombaIO.Checked;
    Program.UI.CurrentRoomba.LogIO = Program.UI.Config.Log.Roomba_IO;
}
private void chkLogPacketData_CheckedChanged(object sender, EventArgs e)
{
    Program.UI.Config.Log.Roomba_PacketData = this.chkLogPacketData.Checked;
}

#endregion
#region Buttons
private void btnApply_Click(object sender, EventArgs e)
{
    Program.UI.Config.COMM.ConnectedTo = this.cCOM_Port.Text;
    Program.UI.Config.Save();
    this.Close();
}

#endregion
#region UD Controls
private void udStartForm_Timer_ValueChanged(object sender, EventArgs e)
{
    Program.UI.Config.Forms.StartForm.Timer = (int)udStartForm_Timer.Value;
}
private void udPollFrequency_ValueChanged(object sender, EventArgs e)
{
    Program.UI.Config.Polling.Frequency = (int)this.udPollFrequency.Value;
    if (Program.UI.CurrentRoomba.ConnectionTime != null)
    {
    }
}
private void udIsCurrent_Threshold_ValueChanged(object sender, EventArgs e)
{
    if (Program.UI.CurrentRoomba.ConnectionTime != null)
    {
        Program.UI.Config.Sensors.IsCurrent_Threshold = (int)this.udIsCurrent_Threshold.Value;
    }
}
private void openToolStripMenuItem_Click(object sender, EventArgs e)
{
}
#endregion
namespace roomba_term
{
    partial class frmMenu
    {
        private System.ComponentModel.IContainer components = null;

        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code

        private void InitializeComponent()
        {
            this.components = new System.ComponentModel.Container();
            this.Menu = new System.Windows.Forms.MenuStrip();
            this.formToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
            this.sensorsToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
            this.sensorPacketToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
            this.driveToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
            this.configToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
            this.tForm_Timer = new System.Windows.Forms.Timer(this.components);
            this.toolTip1 = new System.Windows.Forms.ToolTip(this.components);
            this.Menu.SuspendLayout();
            this.SuspendLayout();

            // Menu
            this.Menu.Items.AddRange(new System.Windows.Forms.ToolStripItem[] { this.formToolStripMenuItem });
            this.Menu.Location = new System.Drawing.Point(0, 0);
            this.Menu.Name = "Menu";
            this.Menu.Size = new System.Drawing.Size(364, 26);
            this.Menu.TabIndex = 0;
            this.Menu.Text = "Menu";

            // formToolStripMenuItem
            this.formToolStripMenuItem.DropDownItems.AddRange(new System.Windows.Forms.ToolStripItem[] { this.sensorsToolStripMenuItem, this.sensorPacketToolStripMenuItem, this.driveToolStripMenuItem, this.configToolStripMenuItem });
            this.formToolStripMenuItem.Name = "formToolStripMenuItem";
            this.formToolStripMenuItem.Size = new System.Drawing.Size(104, 22);
            this.formToolStripMenuItem.Text = "form";
            this.formToolStripMenuItem.Click += new System.EventHandler(this.formToolStripMenuItem_Click);

            // sensorsToolStripMenuItem
            this.sensorsToolStripMenuItem.Name = "sensorsToolStripMenuItem";
            this.sensorsToolStripMenuItem.Size = new System.Drawing.Size(162, 22);
            this.sensorsToolStripMenuItem.Text = "Sensors Menu";
            this.sensorsToolStripMenuItem.Click += new System.EventHandler(this.sensorsToolStripMenuItem_Click);

            // sensorPacketToolStripMenuItem
            this.sensorPacketToolStripMenuItem.Name = "sensorPacketToolStripMenuItem";
            this.sensorPacketToolStripMenuItem.Size = new System.Drawing.Size(185, 22);
            this.sensorPacketToolStripMenuItem.Text = "Sensor Packet Menu";
            this.sensorPacketToolStripMenuItem.Click += new System.EventHandler(this.sensorPacketToolStripMenuItem_Click);

            // driveToolStripMenuItem
            this.driveToolStripMenuItem.Name = "driveToolStripMenuItem";
            this.driveToolStripMenuItem.Size = new System.Drawing.Size(91, 22);
            this.driveToolStripMenuItem.Text = "Drive Menu";
            this.driveToolStripMenuItem.Click += new System.EventHandler(this.driveToolStripMenuItem_Click);

            // configToolStripMenuItem
            this.configToolStripMenuItem.Name = "configToolStripMenuItem";
            this.configToolStripMenuItem.Size = new System.Drawing.Size(94, 22);
            this.configToolStripMenuItem.Text = "Config Menu";
            this.configToolStripMenuItem.Click += new System.EventHandler(this.configToolStripMenuItem_Click);

            // tForm_Timer
            this.tForm_Timer.Interval = 1;
            this.tForm_Timer.Tick += new System.EventHandler(this.tForm_Timer_Tick);

            // timer1
            this.timer1.Interval = 1;
            this.timer1.Tick += new System.EventHandler(this.timer1_Tick);

            this.tForm_Timer.Start();

            // toolTip1
            this.toolTip1.ShowDelay = 500;

            this.Menu.ResumeLayout(false);
            this.ResumeLayout(false);

            // Form Load
            this.FormClosing += new System.Windows.Forms.FormClosingEventHandler(this.frmMenu_FormClosing);

            // Form Closing
            this.ResumeLayout(false);
        }

        #endregion Windows Form Designer generated code

        private void InitializeComponent()
        {
            this.components = new System.ComponentModel.Container();
            this.Menu = new System.Windows.Forms.MenuStrip();
            this.formToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
            this.sensorsToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
            this.sensorPacketToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
            this.driveToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
            this.configToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
            this.tForm_Timer = new System.Windows.Forms.Timer(this.components);
            this.toolTip1 = new System.Windows.Forms.ToolTip(this.components);
            this.Menu.SuspendLayout();
            this.SuspendLayout();

            // Menu
            this.Menu.Items.AddRange(new System.Windows.Forms.ToolStripItem[] { this.formToolStripMenuItem });
            this.Menu.Location = new System.Drawing.Point(0, 0);
            this.Menu.Name = "Menu";
            this.Menu.Size = new System.Drawing.Size(364, 26);
            this.Menu.TabIndex = 0;
            this.Menu.Text = "Menu";

            // formToolStripMenuItem
            this.formToolStripMenuItem.DropDownItems.AddRange(new System.Windows.Forms.ToolStripItem[] { this.sensorsToolStripMenuItem, this.sensorPacketToolStripMenuItem, this.driveToolStripMenuItem, this.configToolStripMenuItem });
            this.formToolStripMenuItem.Name = "formToolStripMenuItem";
            this.formToolStripMenuItem.Size = new System.Drawing.Size(104, 22);
            this.formToolStripMenuItem.Text = "form";
            this.formToolStripMenuItem.Click += new System.EventHandler(this.formToolStripMenuItem_Click);

            // sensorsToolStripMenuItem
            this.sensorsToolStripMenuItem.Name = "sensorsToolStripMenuItem";
            this.sensorsToolStripMenuItem.Size = new System.Drawing.Size(162, 22);
            this.sensorsToolStripMenuItem.Text = "Sensors Menu";
            this.sensorsToolStripMenuItem.Click += new System.EventHandler(this.sensorsToolStripMenuItem_Click);

            // sensorPacketToolStripMenuItem
            this.sensorPacketToolStripMenuItem.Name = "sensorPacketToolStripMenuItem";
            this.sensorPacketToolStripMenuItem.Size = new System.Drawing.Size(185, 22);
            this.sensorPacketToolStripMenuItem.Text = "Sensor Packet Menu";
            this.sensorPacketToolStripMenuItem.Click += new System.EventHandler(this.sensorPacketToolStripMenuItem_Click);

            // driveToolStripMenuItem
            this.driveToolStripMenuItem.Name = "driveToolStripMenuItem";
            this.driveToolStripMenuItem.Size = new System.Drawing.Size(91, 22);
            this.driveToolStripMenuItem.Text = "Drive Menu";
            this.driveToolStripMenuItem.Click += new System.EventHandler(this.driveToolStripMenuItem_Click);

            // configToolStripMenuItem
            this.configToolStripMenuItem.Name = "configToolStripMenuItem";
            this.configToolStripMenuItem.Size = new System.Drawing.Size(94, 22);
            this.configToolStripMenuItem.Text = "Config Menu";
            this.configToolStripMenuItem.Click += new System.EventHandler(this.configToolStripMenuItem_Click);

            // tForm_Timer
            this.tForm_Timer.Interval = 1;
            this.tForm_Timer.Tick += new System.EventHandler(this.tForm_Timer_Tick);

            // timer1
            this.timer1.Interval = 1;
            this.timer1.Tick += new System.EventHandler(this.timer1_Tick);

            this.tForm_Timer.Start();

            // toolTip1
            this.toolTip1.ShowDelay = 500;

            this.Menu.ResumeLayout(false);
            this.ResumeLayout(false);

            // Form Load
            this.FormClosing += new System.Windows.Forms.FormClosingEventHandler(this.frmMenu_FormClosing);

            // Form Closing
            this.ResumeLayout(false);
        }

        // Form Load
        this.FormClosing += new System.Windows.Forms.FormClosingEventHandler(this.frmMenu_FormClosing);

        // Form Closing
        this.ResumeLayout(false);
    }
}

this.configToolStripMenuItem.Name = "configToolStripMenuItem";
this.configToolStripMenuItem.Text = "Config";
this.configToolStripMenuItem.Click += new System.EventHandler(this.configToolStripMenuItem_Click);

// sensorsToolStripMenuItem
this.sensorsToolStripMenuItem.Name = "sensorsToolStripMenuItem";
this.sensorsToolStripMenuItem.Text = "Sensors";
this.sensorsToolStripMenuItem.Click += new System.EventHandler(this.sensorsToolStripMenuItem_Click);

// sensorPacketToolStripMenuItem
this.sensorPacketToolStripMenuItem.Name = "sensorPacketToolStripMenuItem";
this.sensorPacketToolStripMenuItem.Text = "Packet";
this.sensorPacketToolStripMenuItem.Click += new System.EventHandler(this.sensorPacketToolStripMenuItem_Click);

// driveToolStripMenuItem
this.driveToolStripMenuItem.Name = "driveToolStripMenuItem";
this.driveToolStripMenuItem.Text = "Drive";
this.driveToolStripMenuItem.Click += new System.EventHandler(this.driveToolStripMenuItem_Click);

System.EventHandler(this.driveToolStripMenuItem_Click);

// frmMenu
this.AutoScaleDimensions = new System.Drawing.SizeF(8F, 16F);
this.ClientSize = new System.Drawing.Size(364, 150);
this.Controls.Add(this.Menu);
this.MainMenuStrip = this.Menu;
this.Name = "frmMenu";
this.Text = "Menu";
this.FormClosed += new System.Windows.Forms.FormClosedEventHandler(this.frmMenu_FormClosed);
this.Load += new System.EventHandler(this.frmMenu_Load);
this.Menu.ResumeLayout(false);
this.Menu.PerformLayout();
thisResumeLayout(false);
this.PerformLayout();

#endregion

private System.Windows.Forms.MenuStrip Menu;
public System.Windows.Forms.ToolStripMenuItem formToolStripMenuItem;
public System.Windows.Forms.ToolStripMenuItem sensorsToolStripMenuItem;
public System.Windows.Forms.ToolStripMenuItem sensorPacketToolStripMenuItem;
public System.Windows.Forms.ToolStripMenuItem driveToolStripMenuItem;
public System.Windows.Forms.ToolStripMenuItem configToolStripMenuItem;
private System.Windows.Forms.Timer tForm_Timer;
private System.Windows.Forms.ToolTip toolTip1;

namespace roomba_term
{
    partial class frmPacket
    {
        private System.ComponentModel.IContainer components = null;

        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code

        private void InitializeComponent()
        {
            this.components = new System.ComponentModel.Container();
            this.pSyncBytes = new System.Windows.Forms.Panel();
            this.cmPacketMenu = new System.Windows.Forms.ContextMenuStrip(this.components);
            this.displayToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
            this.startToolStripMenuItem1 = new System.Windows.Forms.ToolStripMenuItem();
            this.stopToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
            this.clearToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
            this.outputPacketDataToFileToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
            this.startToolStripMenuItem2 = new System.Windows.Forms.ToolStripMenuItem();
            this.stopToolStripMenuItem1 = new System.Windows.Forms.ToolStripMenuItem();
            this.lblSync5 = new System.Windows.Forms.Label();
            //add InitializeComponent code here
        }

        #endregion

        //private void InitializeComponent()
        {
            this.components = new System.ComponentModel.Container();
            this.pSyncBytes = new System.Windows.Forms.Panel();
            this.cmPacketMenu = new System.Windows.Forms.ContextMenuStrip(this.components);
            this.displayToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
            this.startToolStripMenuItem1 = new System.Windows.Forms.ToolStripMenuItem();
            this.stopToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
            this.clearToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
            this.outputPacketDataToFileToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
            this.startToolStripMenuItem2 = new System.Windows.Forms.ToolStripMenuItem();
            this.stopToolStripMenuItem1 = new System.Windows.Forms.ToolStripMenuItem();
            this.lblSync5 = new System.Windows.Forms.Label();
            //add InitializeComponent code here
        }
    }

    //namespace roomba_term
    {
        partial class frmPacket
        {
            private System.ComponentModel.IContainer components = null;

            protected override void Dispose(bool disposing)
            {
                if (disposing && (components != null))
                {
                    components.Dispose();
                }
                base.Dispose(disposing);
            }

            #region Windows Form Designer generated code

            private void InitializeComponent()
            {
                this.components = new System.ComponentModel.Container();
                this.pSyncBytes = new System.Windows.Forms.Panel();
                this.cmPacketMenu = new System.Windows.Forms.ContextMenuStrip(this.components);
                this.displayToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
                this.startToolStripMenuItem1 = new System.Windows.Forms.ToolStripMenuItem();
                this.stopToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
                this.clearToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
                this.outputPacketDataToFileToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
                this.startToolStripMenuItem2 = new System.Windows.Forms.ToolStripMenuItem();
                this.stopToolStripMenuItem1 = new System.Windows.Forms.ToolStripMenuItem();
                this.lblSync5 = new System.Windows.Forms.Label();
                //add InitializeComponent code here
            }

            #endregion

            private void InitializeComponent()
            {
                this.components = new System.ComponentModel.Container();
                this.pSyncBytes = new System.Windows.Forms.Panel();
                this.cmPacketMenu = new System.Windows.Forms.ContextMenuStrip(this.components);
                this.displayToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
                this.startToolStripMenuItem1 = new System.Windows.Forms.ToolStripMenuItem();
                this.stopToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
                this.clearToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
                this.outputPacketDataToFileToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
                this.startToolStripMenuItem2 = new System.Windows.Forms.ToolStripMenuItem();
                this.stopToolStripMenuItem1 = new System.Windows.Forms.ToolStripMenuItem();
                this.lblSync5 = new System.Windows.Forms.Label();
                //add InitializeComponent code here
            }
        }
    }
}
this.label158 = new System.Windows.Forms.Label();
this.lblSync25 = new System.Windows.Forms.Label();
this.label160 = new System.Windows.Forms.Label();
this.lblSync24 = new System.Windows.Forms.Label();
this.label162 = new System.Windows.Forms.Label();
this.lblSync23 = new System.Windows.Forms.Label();
this.label164 = new System.Windows.Forms.Label();
this.lblSync22 = new System.Windows.Forms.Label();
this.label166 = new System.Windows.Forms.Label();
this.label168 = new System.Windows.Forms.Label();
this.label169 = new System.Windows.Forms.Label();
this.label170 = new System.Windows.Forms.Label();
this.label171 = new System.Windows.Forms.Label();
this.label172 = new System.Windows.Forms.Label();
this.label173 = new System.Windows.Forms.Label();
this.label174 = new System.Windows.Forms.Label();
this.label175 = new System.Windows.Forms.Label();
this.lblSync0 = new System.Windows.Forms.Label();
this.label177 = new System.Windows.Forms.Label();
this.label178 = new System.Windows.Forms.Label();
this.lblSync21 = new System.Windows.Forms.Label();
this.lblSync1 = new System.Windows.Forms.Label();
this.label181 = new System.Windows.Forms.Label();
this.label182 = new System.Windows.Forms.Label();
this.lblSync20 = new System.Windows.Forms.Label();
this.label184 = new System.Windows.Forms.Label();
this.lblSync7 = new System.Windows.Forms.Label();
this.label186 = new System.Windows.Forms.Label();
this.lblSync6 = new System.Windows.Forms.Label();
this.label188 = new System.Windows.Forms.Label();
this.label189 = new System.Windows.Forms.Label();
this.lblSync4 = new System.Windows.Forms.Label();
this.label191 = new System.Windows.Forms.Label();
this.lblSync3 = new System.Windows.Forms.Label();
this.label193 = new System.Windows.Forms.Label();
this.lblSync2 = new System.Windows.Forms.Label();
this.lblSync19 = new System.Windows.Forms.Label();
this.lblSync18 = new System.Windows.Forms.Label();
this.lblSync17 = new System.Windows.Forms.Label();
this.lblSync16 = new System.Windows.Forms.Label();
this.lblSync8 = new System.Windows.Forms.Label();
this.lblSync15 = new System.Windows.Forms.Label();
this.lblSync9 = new System.Windows.Forms.Label();
this.lblSync14 = new System.Windows.Forms.Label();
this.lblSync10 = new System.Windows.Forms.Label();
this.lblSync13 = new System.Windows.Forms.Label();
this.lblSync11 = new System.Windows.Forms.Label();
this.lblSync12 = new System.Windows.Forms.Label();
this.lblRaw5 = new System.Windows.Forms.Label();
this.label2 = new System.Windows.Forms.Label();
this.lblRaw25 = new System.Windows.Forms.Label();
this.label4 = new System.Windows.Forms.Label();
this.lblRaw24 = new System.Windows.Forms.Label();
this.label6 = new System.Windows.Forms.Label();
this.lblRaw23 = new System.Windows.Forms.Label();
this.label8 = new System.Windows.Forms.Label();
this.lblRaw22 = new System.Windows.Forms.Label();
this.label10 = new System.Windows.Forms.Label();
this.label11 = new System.Windows.Forms.Label();
this.label12 = new System.Windows.Forms.Label();
this.label13 = new System.Windows.Forms.Label();
this.label14 = new System.Windows.Forms.Label();
this.label15 = new System.Windows.Forms.Label();
this.label16 = new System.Windows.Forms.Label();
this.label17 = new System.Windows.Forms.Label();
this.label18 = new System.Windows.Forms.Label();
this.lblRaw0 = new System.Windows.Forms.Label();
this.label20 = new System.Windows.Forms.Label();
this.label21 = new System.Windows.Forms.Label();
this.lblRaw21 = new System.Windows.Forms.Label();
this.lblRaw1 = new System.Windows.Forms.Label();
this.label24 = new System.Windows.Forms.Label();
this.label25 = new System.Windows.Forms.Label();
this.lblRaw20 = new System.Windows.Forms.Label();
this.label27 = new System.Windows.Forms.Label();
this.lblRaw7 = new System.Windows.Forms.Label();
this.label29 = new System.Windows.Forms.Label();
this.lblRaw6 = new System.Windows.Forms.Label();
this.label31 = new System.Windows.Forms.Label();
this.label32 = new System.Windows.Forms.Label();
this.lblRaw4 = new System.Windows.Forms.Label();
this.label34 = new System.Windows.Forms.Label();
this.lblRaw3 = new System.Windows.Forms.Label();
this.label36 = new System.Windows.Forms.Label();
this.lblRaw2 = new System.Windows.Forms.Label();
this.label38 = new System.Windows.Forms.Label();
this.lblRaw19 = new System.Windows.Forms.Label();
this.label40 = new System.Windows.Forms.Label();
this.lblRaw18 = new System.Windows.Forms.Label();
this.label42 = new System.Windows.Forms.Label();
this.lblRaw17 = new System.Windows.Forms.Label();
this.lblRaw16 = new System.Windows.Forms.Label();
this.lblRaw8 = new System.Windows.Forms.Label();
this.lblRaw15 = new System.Windows.Forms.Label();
this.lblRaw9 = new System.Windows.Forms.Label();
this.lblRaw14 = new System.Windows.Forms.Label();
this.lblRaw10 = new System.Windows.Forms.Label();
this.lblRaw13 = new System.Windows.Forms.Label();
this.pSyncBytes.Controls.Add(this.label178);
this.pSyncBytes.Controls.Add(this.lblSync21);
this.pSyncBytes.Controls.Add(this.lblSync1);
this.pSyncBytes.Controls.Add(this.label181);
this.pSyncBytes.Controls.Add(this.label182);
this.pSyncBytes.Controls.Add(this.lblSync20);
this.pSyncBytes.Controls.Add(this.label184);
this.pSyncBytes.Controls.Add(this.lblSync7);
this.pSyncBytes.Controls.Add(this.pSyncBytes.Controls.Add(this.label186);
this.pSyncBytes.Controls.Add(this.lblSync6);
this.pSyncBytes.Controls.Add(this.label188);
this.pSyncBytes.Controls.Add(this.lblSync3);
this.pSyncBytes.Controls.Add(this.label191);
this.pSyncBytes.Controls.Add(this.lblSync2);
this.pSyncBytes.Controls.Add(this.label193);
this.pSyncBytes.Controls.Add(this.lblSync19);
this.pSyncBytes.Controls.Add(this.label195);
this.pSyncBytes.Controls.Add(this.lblSync18);
this.pSyncBytes.Controls.Add(this.label197);
this.pSyncBytes.Controls.Add(this.lblSync17);
this.pSyncBytes.Controls.Add(this.lblSync16);
this.pSyncBytes.Controls.Add(this.lblSync8);
this.pSyncBytes.Controls.Add(this.lblSync15);
this.pSyncBytes.Controls.Add(this.lblSync9);
this.pSyncBytes.Controls.Add(this.lblSync14);
this.pSyncBytes.Controls.Add(this.lblSync10);
this.pSyncBytes.Controls.Add(this.lblSync13);
this.pSyncBytes.Controls.Add(this.lblSync11);
this.pSyncBytes.Controls.Add(this.lblSync12);
this.pSyncBytes.Enabled = false;
this.pSyncBytes.Location = new System.Drawing.Point(5, 25);
this.pSyncBytes.Name = "pSyncBytes";
this.pSyncBytes.Size = new System.Drawing.Size(603, 115);
this.pSyncBytes.TabIndex = 220;

// cmPacketMenu
//
this.cmPacketMenu.Items.AddRange(new System.Windows.Forms.ToolStripItem[] {
    this.displayToolStripMenuItem,
    this.outputPacketDataToFileToolStripMenuItem,
    this.explanationOfThisFormToolStripMenuItem1});
this.cmPacketMenu.Name = "cmConnectionMenu";
this.cmPacketMenu.Size = new System.Drawing.Size(246, 92);

// displayToolStripMenuItem
//
this.displayToolStripMenuItem.DropDownItems.AddRange(new System.Windows.Forms.ToolStripItem[] {
    // other ToolStripMenuItem instances
    new System.Windows.Forms.ToolStripMenuItem[] { 
        // other ToolStripMenuItem instances
    });
this.cmPacketMenu.Items.AddRange(new System.Windows.Forms.ToolStripItem[] {
    // other ToolStripMenuItem instances
    new System.Windows.Forms.ToolStripMenuItem[] { 
        // other ToolStripMenuItem instances
    });
this.startToolStripMenuItem1, this.stopToolStripMenuItem, this.clearToolStripMenuItem); this.displayToolStripMenuItem.Name = "displayToolStripMenuItem"; this.displayToolStripMenuItem.Size = new System.Drawing.Size(245, 22); this.displayToolStripMenuItem.Text = "Display"; // // startToolStripMenuItem1 // this.startToolStripMenuItem1.Name = "startToolStripMenuItem1"; this.startToolStripMenuItem1.Size = new System.Drawing.Size(152, 22); this.startToolStripMenuItem1.Text = "Start"; this.startToolStripMenuItem1.Click += new System.EventHandler(this.restartConnectionToolStripMenuItem_Click); // // stopToolStripMenuItem // this.stopToolStripMenuItem.Name = "stopToolStripMenuItem"; this.stopToolStripMenuItem.Size = new System.Drawing.Size(152, 22); this.stopToolStripMenuItem.Text = "Stop"; this.stopToolStripMenuItem.Click += new System.EventHandler(this.stopToolStripMenuItem_Click); // // clearToolStripMenuItem // this.clearToolStripMenuItem.Name = "clearToolStripMenuItem"; this.clearToolStripMenuItem.Size = new System.Drawing.Size(152, 22); this.clearToolStripMenuItem.Text = "Clear"; this.clearToolStripMenuItem.Click += new System.EventHandler(this.clearToolStripMenuItem_Click); // // outputPacketDataToFileToolStripMenuItem // this.outputPacketDataToFileToolStripMenuItem.DropDownItems.AddRange(new System.Windows.Forms.ToolStripItem[] { this.startToolStripMenuItem2, this.stopToolStripMenuItem1 }); this.outputPacketDataToFileToolStripMenuItem.Name = "outputPacketDataToFileToolStripMenuItem"; this.outputPacketDataToFileToolStripMenuItem.Size = new System.Drawing.Size(245, 22); this.outputPacketDataToFileToolStripMenuItem.Text = "Output to File"; this.outputPacketDataToFileToolStripMenuItem.Click += new System.EventHandler(this.outputPacketDataToFileToolStripMenuItem_Click); // // startToolStripMenuItem2 // this.startToolStripMenuItem2.Name = "startToolStripMenuItem2"; this.startToolStripMenuItem2.Size = new System.Drawing.Size(152, 22); this.startToolStripMenuItem2.Text = "Start"; // // stopToolStripMenuItem1 // this.stopToolStripMenuItem1.Name = "stopToolStripMenuItem1";
this.stopToolStripMenuItem1.Size = new System.Drawing.Size(152, 22);
this.stopToolStripMenuItem1.Text = "Stop";

//
// lblSync5
//
this.lblSync5.ContextMenuStrip = this.cmPacketMenu;
this.lblSync5.Location = new System.Drawing.Point(244, 31);
this.lblSync5.Name = "lblSync5";
this.lblSync5.Size = new System.Drawing.Size(37, 18);
this.lblSync5.TabIndex = 169;

//
// label158
//
this.label158.AutoSize = true;
this.label158.ContextMenuStrip = this.cmPacketMenu;
this.label158.Location = new System.Drawing.Point(199, 60);
this.label158.Name = "label158";
this.label158.Size = new System.Drawing.Size(32, 17);
this.label158.TabIndex = 121;
this.label158.Text = "[17]";

//
// lblSync25
//
this.lblSync25.ContextMenuStrip = this.cmPacketMenu;
this.lblSync25.Location = new System.Drawing.Point(556, 85);
this.lblSync25.Name = "lblSync25";
this.lblSync25.Size = new System.Drawing.Size(37, 18);
this.lblSync25.TabIndex = 209;

//
// label160
//
this.label160.AutoSize = true;
this.label160.ContextMenuStrip = this.cmPacketMenu;
this.label160.Location = new System.Drawing.Point(14, 60);
this.label160.Name = "label160";
this.label160.Size = new System.Drawing.Size(32, 17);
this.label160.TabIndex = 113;
this.label160.Text = "[13]";

//
// lblSync24
//
this.lblSync24.ContextMenuStrip = this.cmPacketMenu;
this.lblSync24.Location = new System.Drawing.Point(510, 85);
this.lblSync24.Name = "lblSync24";
this.lblSync24.Size = new System.Drawing.Size(37, 18);
this.lblSync24.TabIndex = 208;

//
// label162
//
this.label162.AutoSize = true;
this.label162.ContextMenuStrip = this.cmPacketMenu;
this.label162.Location = new System.Drawing.Point(558, 10);
this.label162.Name = "label162";
this.label162.Size = new System.Drawing.Size(32, 17);
this.label162.TabIndex = 111;
this.label162.Text = "[12]";

//
// lblSync23
//
this.lblSync23.ContextMenuStrip = this.cmPacketMenu;
this.lblSync23.Location = new System.Drawing.Point(465, 85);
this.lblSync23.Name = "lblSync23";
this.lblSync23.Size = new System.Drawing.Size(37, 18);
this.lblSync23.TabIndex = 207;

//
// label164
//
this.label164.AutoSize = true;
this.label164.ContextMenuStrip = this.cmPacketMenu;
this.label164.Location = new System.Drawing.Point(512, 10);
this.label164.Name = "label164";
this.label164.Size = new System.Drawing.Size(32, 17);
this.label164.TabIndex = 109;
this.label164.Text = "[11]";
// // lblSync22
//
this.lblSync22.ContextMenuStrip = this.cmPacketMenu;
this.lblSync22.Location = new System.Drawing.Point(421, 85);
this.lblSync22.Name = "lblSync22";
this.lblSync22.Size = new System.Drawing.Size(37, 18);
this.lblSync22.TabIndex = 206;

// // label166
//
this.label166.AutoSize = true;
this.label166.ContextMenuStrip = this.cmPacketMenu;
this.label166.Location = new System.Drawing.Point(291, 60);
this.label166.Name = "label166";
this.label166.Size = new System.Drawing.Size(32, 17);
this.label166.TabIndex = 125;
this.label166.Text = "[19]";

// // label168
//
this.label168.AutoSize = true;
this.label168.ContextMenuStrip = this.cmPacketMenu;
this.label168.Location = new System.Drawing.Point(246, 60);
this.label168.Name = "label168";
this.label168.Size = new System.Drawing.Size(32, 17);
this.label168.TabIndex = 123;
this.label168.Text = "[18]";

// // label169
//
this.label169.AutoSize = true;
this.label169.ContextMenuStrip = this.cmPacketMenu;
this.label169.Location = new System.Drawing.Point(558, 60);
this.label169.Name = "label169";
this.label169.Size = new System.Drawing.Size(32, 17);
this.label169.TabIndex = 188;
this.label169.Text = "[25]";


this.label173.Text = "[23]";

//
// label174
this.label174.AutoSize = true;
this.label174.ContextMenuStrip = this.cmPacketMenu;
this.label174.Location = new System.Drawing.Point(60, 60);
this.label174.Name = "label174";
this.label174.Size = new System.Drawing.Size(32, 17);
this.label174.TabIndex = 115;
this.label174.Text = "[14]";

//
// label175
this.label175.AutoSize = true;
this.label175.ContextMenuStrip = this.cmPacketMenu;
this.label175.Location = new System.Drawing.Point(423, 60);
this.label175.Name = "label175";
this.label175.Size = new System.Drawing.Size(32, 17);
this.label175.TabIndex = 185;
this.label175.Text = "[22]";

//
// lblSync0
this.lblSync0.ContextMenuStrip = this.cmPacketMenu;
this.lblSync0.Location = new System.Drawing.Point(12, 31);
this.lblSync0.Name = "lblSync0";
this.lblSync0.Size = new System.Drawing.Size(37, 18);
this.lblSync0.TabIndex = 130;

//
// label177
this.label177.AutoSize = true;
this.label177.ContextMenuStrip = this.cmPacketMenu;
this.label177.Location = new System.Drawing.Point(379, 60);
this.label177.Name = "label177";
this.label177.Size = new System.Drawing.Size(32, 17);
this.label177.TabIndex = 184;
this.label177.Text = "[21]";
//
// label178
//
this.label178.Anchor =
this.label178.AutoSize = true;
this.label178.ContextMenuStrip = this.cmPacketMenu;
this.label178.Location = new System.Drawing.Point(18, 10);
this.label178.Name = "label178";
this.label178.Size = new System.Drawing.Size(24, 17);
this.label178.TabIndex = 131;
this.label178.Text = "[0]";
//
// lblSync21
//
this.lblSync21.Anchor =
this.lblSync21.ContextMenuStrip = this.cmPacketMenu;
this.lblSync21.Location = new System.Drawing.Point(377, 85);
this.lblSync21.Name = "lblSync21";
this.lblSync21.Size = new System.Drawing.Size(37, 18);
this.lblSync21.TabIndex = 183;
//
// lblSync1
//
this.lblSync1.Anchor =
this.lblSync1.ContextMenuStrip = this.cmPacketMenu;
this.lblSync1.Location = new System.Drawing.Point(58, 31);
this.lblSync1.Name = "lblSync1";
this.lblSync1.Size = new System.Drawing.Size(37, 18);
this.lblSync1.TabIndex = 132;
//
// label181
//
this.label181.Anchor =
this.label181.AutoSize = true;
this.label181.ContextMenuStrip = this.cmPacketMenu;
this.label181.Location = new System.Drawing.Point(335, 60);
this.label181.Name = "label181";
this.label181.Size = new System.Drawing.Size(32, 17);
this.label181.TabIndex = 182;
this.label181.Text = "[20]";
//
// label182
//
this.label182.AutoSize = true;
this.label182.ContextMenuStrip = this.cmPacketMenu;
this.label182.Location = new System.Drawing.Point(64, 10);
this.label182.Name = "label182";
this.label182.Size = new System.Drawing.Size(24, 17);
this.label182.TabIndex = 133;
this.label182.Text = "[1]";
//
// lblSync20
//
this.lblSync20.ContextMenuStrip = this.cmPacketMenu;
this.lblSync20.Location = new System.Drawing.Point(333, 85);
this.lblSync20.Name = "lblSync20";
this.lblSync20.Size = new System.Drawing.Size(37, 18);
this.lblSync20.TabIndex = 181;
//
// label184
//
this.label184.AutoSize = true;
this.label184.ContextMenuStrip = this.cmPacketMenu;
this.label184.Location = new System.Drawing.Point(111, 10);
this.label184.Name = "label184";
this.label184.Size = new System.Drawing.Size(24, 17);
this.label184.TabIndex = 135;
this.label184.Text = "[2]";
//
// lblSync7
//
this.lblSync7.ContextMenuStrip = this.cmPacketMenu;
this.lblSync7.Location = new System.Drawing.Point(333, 31);
this.lblSync7.Name = "lblSync7";
this.lblSync7.Size = new System.Drawing.Size(37, 18);
this.lblSync7.TabIndex = 171;
//
// label186
//
this.label186.Anchor =
this.label186.AutoSize = true;
this.label186.ContextMenuStrip = this.cmsPacketMenu;
this.label186.Location = new System.Drawing.Point(157, 10);
this.label186.Name = "label186";
this.label186.Size = new System.Drawing.Size(24, 17);
this.label186.TabIndex = 137;
this.label186.Text = "[3]";

//
// lblSync6
//
this.lblSync6.Anchor =
this.lblSync6.ContextMenuStrip = this.cmsPacketMenu;
this.lblSync6.Location = new System.Drawing.Point(289, 31);
this.lblSync6.Name = "lblSync6";
this.lblSync6.Size = new System.Drawing.Size(37, 18);
this.lblSync6.TabIndex = 170;

//
// label188
//
this.label188.Anchor =
this.label188.AutoSize = true;
this.label188.ContextMenuStrip = this.cmsPacketMenu;
this.label188.Location = new System.Drawing.Point(203, 10);
this.label188.Name = "label188";
this.label188.Size = new System.Drawing.Size(24, 17);
this.label188.TabIndex = 139;
this.label188.Text = "[4]";

//
// label189
//
this.label189.Anchor =
this.label189.AutoSize = true;
this.label189.ContextMenuStrip = this.cmsPacketMenu;
this.label189.Location = new System.Drawing.Point(250, 10);
this.label189.Name = "label189";
this.label189.Size = new System.Drawing.Size(24, 17);
this.label189.TabIndex = 141;
this.label189.Text = "[5]";

//
// lblSync4
//
this.lblSync4.ContextMenuStrip = this.cmPacketMenu;  
this.lblSync4.Location = new System.Drawing.Point(197, 31);  
this.lblSync4.Name = "lblSync4";
this.lblSync4.Size = new System.Drawing.Point(37, 18);
this.lblSync4.TabIndex = 168;
//
// label191
//
this.label191.AutoSize = true;  
this.label191.ContextMenuStrip = this.cmPacketMenu;  
this.label191.Location = new System.Drawing.Point(295, 10);  
this.label191.Name = "label191";
this.label191.Size = new System.Drawing.Size(24, 17);
this.label191.TabIndex = 143;
this.label191.Text = "[6]";
//
// lblSync3
//
this.lblSync3.ContextMenuStrip = this.cmPacketMenu;  
this.lblSync3.Location = new System.Drawing.Point(151, 31);  
this.lblSync3.Name = "lblSync3";
this.lblSync3.Size = new System.Drawing.Point(37, 18);
this.lblSync3.TabIndex = 167;
//
// label193
//
this.label193.AutoSize = true;  
this.label193.ContextMenuStrip = this.cmPacketMenu;  
this.label193.Location = new System.Drawing.Point(339, 10);  
this.label193.Name = "label193";
this.label193.Size = new System.Drawing.Size(24, 17);
this.label193.TabIndex = 145;
this.label193.Text = "[7]";
//
// lblSync2
this.lblSync2.ContextMenuStrip = this.cmPacketMenu;
this.lblSync2.Location = new System.Drawing.Point(105, 31);
this.lblSync2.Name = "lblSync2";
this.lblSync2.Size = new System.Drawing.Size(37, 18);
this.lblSync2.TabIndex = 166;
//
// label195
//
this.label195.AutoSize = true;
this.label195.ContextMenuStrip = this.cmPacketMenu;
this.label195.Location = new System.Drawing.Point(383, 10);
this.label195.Name = "label195";
this.label195.Size = new System.Drawing.Size(24, 17);
this.label195.TabIndex = 147;
this.label195.Text = "[8]";
//
// lblSync19
//
this.lblSync19Anchor =
this.lblSync19.ContextMenuStrip = this.cmPacketMenu;
this.lblSync19.Location = new System.Drawing.Point(289, 85);
this.lblSync19.Name = "lblSync19";
this.lblSync19.Size = new System.Drawing.Size(37, 18);
this.lblSync19.TabIndex = 165;
//
// label197
//
this.label197Anchor =
this.label197.AutoSize = true;
this.label197.ContextMenuStrip = this.cmPacketMenu;
this.label197.Location = new System.Drawing.Point(427, 10);
this.label197.Name = "label197";
this.label197.Size = new System.Drawing.Size(24, 17);
this.label197.TabIndex = 149;
this.label197.Text = "[9]";
this.lblSync18.ContextMenuStrip = this.cmPacketMenu;
this.lblSync18.Location = new System.Drawing.Point(244, 85);
this.lblSync18.Name = "lblSync18";
this.lblSync18.Size = new System.Drawing.Size(37, 18);
this.lblSync18.TabIndex = 164;

this.label199.AutoSize = true;
this.label199.ContextMenuStrip = this.cmPacketMenu;
this.label199.Location = new System.Drawing.Point(467, 10);
this.label199.Name = "label199";
this.label199.Size = new System.Drawing.Size(32, 17);
this.label199.TabIndex = 151;
this.label199.Text = "[10]";

this.lblSync17.ContextMenuStrip = this.cmPacketMenu;
this.lblSync17.Location = new System.Drawing.Point(197, 85);
this.lblSync17.Name = "lblSync17";
this.lblSync17.Size = new System.Drawing.Size(37, 18);
this.lblSync17.TabIndex = 163;

this.lblSync16.ContextMenuStrip = this.cmPacketMenu;
this.lblSync16.Location = new System.Drawing.Point(151, 85);
this.lblSync16.Name = "lblSync16";
this.lblSync16.Size = new System.Drawing.Size(37, 18);
this.lblSync16.TabIndex = 162;

this.lblSync8.ContextMenuStrip = this.cmPacketMenu;
this.lblSync8.Location = new System.Drawing.Point(151, 85);
this.lblSync8.Name = "lblSync8";
this.lblSync8.Size = new System.Drawing.Size(37, 18);
this.lblSync8.TabIndex = 159;
// // lblSync15
// // lblSync9
// // lblSync14
// // lblSync10
this.lblSync10.ContextMenuStrip = this.cmPacketMenu;
this.lblSync10.Location = new System.Drawing.Point(465, 31);
this.lblSync10.Name = "lblSync10";
this.lblSync10.Size = new System.Drawing.Size(37, 18);
this.lblSync10.TabIndex = 156;
//
// lblSync13
//
this.lblSync13.ContextMenuStrip = this.cmPacketMenu;
this.lblSync13.Location = new System.Drawing.Point(12, 85);
this.lblSync13.Name = "lblSync13";
this.lblSync13.Size = new System.Drawing.Size(37, 18);
this.lblSync13.TabIndex = 159;
//
// lblSync11
//
this.lblSync11.ContextMenuStrip = this.cmPacketMenu;
this.lblSync11.Location = new System.Drawing.Point(510, 31);
this.lblSync11.Name = "lblSync11";
this.lblSync11.Size = new System.Drawing.Size(37, 18);
this.lblSync11.TabIndex = 157;
//
// lblSync12
//
this.lblSync12.ContextMenuStrip = this.cmPacketMenu;
this.lblSync12.Location = new System.Drawing.Point(556, 31);
this.lblSync12.Name = "lblSync12";
this.lblSync12.Size = new System.Drawing.Size(37, 18);
this.lblSync12.TabIndex = 158;
//
// pRawBytes
//
this.pRawBytes.Controls.Add(this.lblRaw5);
this.pRawBytes.Controls.Add(this.label2);
this.pRawBytes.Controls.Add(this.lblRaw25);
this.pRawBytes.Controls.Add(this.label4);
this.pRawBytes.Controls.Add(this.lblRaw24);
this.pRawBytes.Controls.Add(this.label6);
this.pRawBytes.Controls.Add(this.lblRaw23);
this.pRawBytes.Controls.Add(this.label8);
this.pRawBytes.Controls.Add(this.lblRaw22);
this.pRawBytes.Controls.Add(this.label10);
this.pRawBytes.Controls.Add(this.label11);
this.pRawBytes.Controls.Add(this.label12);
this.pRawBytes.Controls.Add(this.label13);
this.pRawBytes.Controls.Add(this.label14);
this.pRawBytes.Controls.Add(this.label15);
this.pRawBytes.Controls.Add(this.label16);
this.pRawBytes.Controls.Add(this.label17);
this.pRawBytes.Controls.Add(this.label18);
this.pRawBytes.Controls.Add(this.lblRaw0);
this.pRawBytes.Controls.Add(this.label20);
this.pRawBytes.Controls.Add(this.label21);
this.pRawBytes.Controls.Add(this.lblRaw21);
this.pRawBytes.Controls.Add(this.lblRaw1);
this.pRawBytes.Controls.Add(this.label24);
this.pRawBytes.Controls.Add(this.label25);
this.pRawBytes.Controls.Add(this.lblRaw20);
this.pRawBytes.Controls.Add(this.label27);
this.pRawBytes.Controls.Add(this.lblRaw7);
this.pRawBytes.Controls.Add(this.label29);
this.pRawBytes.Controls.Add(this.lblRaw6);
this.pRawBytes.Controls.Add(this.label31);
this.pRawBytes.Controls.Add(this.label32);
this.pRawBytes.Controls.Add(this.lblRaw4);
this.pRawBytes.Controls.Add(this.lblRaw3);
this.pRawBytes.Controls.Add(this.label36);
this.pRawBytes.Controls.Add(this.lblRaw2);
this.pRawBytes.Controls.Add(this.label38);
this.pRawBytes.Controls.Add(this.lblRaw19);
this.pRawBytes.Controls.Add(this.label40);
this.pRawBytes.Controls.Add(this.lblRaw18);
this.pRawBytes.Controls.Add(this.label42);
this.pRawBytes.Controls.Add(this.lblRaw17);
this.pRawBytes.Controls.Add(this.lblRaw16);
this.pRawBytes.Controls.Add(this.lblRaw8);
this.pRawBytes.Controls.Add(this.lblRaw15);
this.pRawBytes.Controls.Add(this.lblRaw9);
this.pRawBytes.Controls.Add(this.lblRaw14);
this.pRawBytes.Controls.Add(this.lblRaw10);
this.pRawBytes.Controls.Add(this.lblRaw13);
this.pRawBytes.Controls.Add(this.lblRaw11);
this.pRawBytes.Controls.Add(this.lblRaw12);
this.pRawBytes.Enabled = false;
this.pRawBytes.Location = new System.Drawing.Point(4, 24);
this.pRawBytes.Name = "pRawBytes";
this.pRawBytes.Size = new System.Drawing.Size(603, 115);
this.pRawBytes.TabIndex = 221;
// // lblRaw5
//
this.lblRaw5.Anchor =
this.lblRaw5.Location = new System.Drawing.Point(244, 31);
this.lblRaw5.Name = "lblRaw5";
this.lblRaw5.Size = new System.Drawing.Size(37, 18);
this.lblRaw5.TabIndex = 169;

// // label2
//
this.label2.Anchor =
this.label2.AutoSize = true;
this.label2.Location = new System.Drawing.Point(199, 60);
this.label2.Name = "label2";
this.label2.Size = new System.Drawing.Size(32, 17);
this.label2.TabIndex = 121;
this.label2.Text = "[17]";

// // lblRaw25
//
this.lblRaw25.Anchor =
this.lblRaw25.Location = new System.Drawing.Point(556, 85);
this.lblRaw25.Name = "lblRaw25";
this.lblRaw25.Size = new System.Drawing.Size(37, 18);
this.lblRaw25.TabIndex = 209;

// // label4
//
this.label4.Anchor =
this.label4.AutoSize = true;
this.label4.Location = new System.Drawing.Point(14, 60);
this.label4.Name = "label4";
this.label4.Size = new System.Drawing.Size(32, 17);
this.label4.TabIndex = 113;
this.label4.Text = "[13]";

// // lblRaw24
//
    this.lblRaw24.Location = new System.Drawing.Point(510, 85);
    this.lblRaw24.Name = "lblRaw24";
    this.lblRaw24.Size = new System.Drawing.Size(37, 18);
    this.lblRaw24.TabIndex = 208;
    //
    // label6
    //
    this.label6.AutoSize = true;
    this.label6.Location = new System.Drawing.Point(558, 10);
    this.label6.Name = "label6";
    this.label6.Size = new System.Drawing.Size(32, 17);
    this.label6.TabIndex = 111;
    this.label6.Text = "[12]";
    //
    // lblRaw23
    //
    this.lblRaw23.Location = new System.Drawing.Point(465, 85);
    this.lblRaw23.Name = "lblRaw23";
    this.lblRaw23.Size = new System.Drawing.Size(37, 18);
    this.lblRaw23.TabIndex = 207;
    //
    // label8
    //
    this.label8.AutoSize = true;
    this.label8.Location = new System.Drawing.Point(512, 10);
    this.label8.Name = "label8";
    this.label8.Size = new System.Drawing.Size(32, 17);
    this.label8.TabIndex = 109;
    this.label8.Text = "[11]";
    //
    // lblRaw22
    //
    this.lblRaw22.Location = new System.Drawing.Point(421, 85);
this.lblRaw22.Name = "lblRaw22";
this.lblRaw22.Size = new System.Drawing.Size(37, 18);
this.lblRaw22.TabIndex = 206;

//
// label10
//
this.label10.AutoSize = true;
this.label10.Location = new System.Drawing.Point(291, 60);
this.label10.Name = "label10";
this.label10.Size = new System.Drawing.Size(32, 17);
this.label10.TabIndex = 125;
this.label10.Text = "[19]";

//
// label11
//
this.label11.AutoSize = true;
this.label11.Location = new System.Drawing.Point(246, 60);
this.label11.Name = "label11";
this.label11.Size = new System.Drawing.Size(32, 17);
this.label11.TabIndex = 123;
this.label11.Text = "[18]";

//
// label12
//
this.label12.AutoSize = true;
this.label12.Location = new System.Drawing.Point(558, 60);
this.label12.Name = "label12";
this.label12.Size = new System.Drawing.Size(32, 17);
this.label12.TabIndex = 188;
this.label12.Text = "[25]";

//
// label13
//
this.label13.AutoSize = true;
this.label13.Location = new System.Drawing.Point(153, 60);
this.label13.Name = "label13";
this.label13.Size = new System.Drawing.Size(32, 17);
this.label13.TabIndex = 119;
this.label13.Text = "[16]";
// label14
//
this.label14.AutoSize = true;
this.label14.Location = new System.Drawing.Point(512, 60);
this.label14.Name = "label14";
this.label14.Size = new System.Drawing.Size(32, 17);
this.label14.TabIndex = 187;
this.label14.Text = "[24]";

// label15
//
this.label15.AutoSize = true;
this.label15.Location = new System.Drawing.Point(107, 60);
this.label15.Name = "label15";
this.label15.Size = new System.Drawing.Size(32, 17);
this.label15.TabIndex = 117;
this.label15.Text = "[15]";

// label16
//
this.label16.AutoSize = true;
this.label16.Location = new System.Drawing.Point(467, 60);
this.label16.Name = "label16";
this.label16.Size = new System.Drawing.Size(32, 17);
this.label16.TabIndex = 186;
this.label16.Text = "[23]";

// label17
//
this.label17.AutoSize = true;
this.label17.Location = new System.Drawing.Point(60, 60);
this.label17.Name = "label17";
this.label17.Size = new System.Drawing.Size(32, 17);
this.label17.TabIndex = 115;
this.label17.Text = "[14]";

// label18
//
this.label18.AutoSize = true;
this.label18.Location = new System.Drawing.Point(423, 60);
this.label18.Name = "label18";
this.label18.Size = new System.Drawing.Size(32, 17);
this.label18.TabIndex = 185;
this.label18.Text = ":[22];"
//
// lblRaw0
//
this.lblRaw0.Location = new System.Drawing.Point(12, 31);
this.lblRaw0.Name = "lblRaw0";
this.lblRaw0.Size = new System.Drawing.Size(37, 18);
this.lblRaw0.TabIndex = 130;
//
// label20
//
this.label20.AutoSize = true;
this.label20.Location = new System.Drawing.Point(379, 60);
this.label20.Name = "label20";
this.label20.Size = new System.Drawing.Size(32, 17);
this.label20.TabIndex = 184;
this.label20.Text = ":[21];"
//
// label21
//
this.label21.AutoSize = true;
this.label21.Location = new System.Drawing.Point(18, 10);
this.label21.Name = "label21";
this.label21.Size = new System.Drawing.Size(24, 17);
this.label21.TabIndex = 131;
this.label21.Text = ":[0];"
//
// lblRaw21
//
this.lblRaw21.Location = new System.Drawing.Point(377, 85);
this.lblRaw21.Name = "lblRaw21";
this.lblRaw21.Size = new System.Drawing.Size(37, 18);
this.lblRaw21.TabIndex = 183;

this.lblRaw1.Location = new System.Drawing.Point(58, 31);
this.lblRaw1.Name = "lblRaw1";
this.lblRaw1.Size = new System.Drawing.Size(37, 18);
this.lblRaw1.TabIndex = 132;

this.label24.AutoSize = true;
this.label24.Location = new System.Drawing.Point(335, 60);
this.label24.Name = "label24";
this.label24.Size = new System.Drawing.Size(32, 17);
this.label24.TabIndex = 182;
this.label24.Text = "[20]";

this.label25.AutoSize = true;
this.label25.Location = new System.Drawing.Point(64, 10);
this.label25.Name = "label25";
this.label25.Size = new System.Drawing.Size(24, 17);
this.label25.TabIndex = 133;
this.label25.Text = "[1]";

this.lblRaw20.Location = new System.Drawing.Point(333, 85);
this.lblRaw20.Name = "lblRaw20";
this.lblRaw20.Size = new System.Drawing.Size(37, 18);
this.lblRaw20.TabIndex = 181;
// label27
//
this.label27.Anchor =
this.label27.AutoSize = true;
this.label27.Location = new System.Drawing.Point(111, 10);
this.label27.Name = "label27";
this.label27.Size = new System.Drawing.Size(24, 17);
this.label27.TabIndex = 135;
this.label27.Text = "[2]";
//
// lblRaw7
//
this.lblRaw7.Anchor =
this.lblRaw7.Location = new System.Drawing.Point(333, 31);
this.lblRaw7.Name = "lblRaw7";
this.lblRaw7.Size = new System.Drawing.Size(37, 18);
this.lblRaw7.TabIndex = 171;
//
// label29
//
this.label29.Anchor =
this.label29.AutoSize = true;
this.label29.Location = new System.Drawing.Point(157, 10);
this.label29.Name = "label29";
this.label29.Size = new System.Drawing.Size(24, 17);
this.label29.TabIndex = 137;
this.label29.Text = "[3]";
//
// lblRaw6
//
this.lblRaw6.Anchor =
this.lblRaw6.Location = new System.Drawing.Point(289, 31);
this.lblRaw6.Name = "lblRaw6";
this.lblRaw6.Size = new System.Drawing.Size(37, 18);
this.lblRaw6.TabIndex = 170;
//
// label31
//
this.label31.Anchor =
this.label31.AutoSize = true;
this.label31.Location = new System.Drawing.Point(203, 10);
this.label31.Name = "label31";
this.label31.Size = new System.Drawing.Size(24, 17);
this.label31.TabIndex = 139;
this.label31.Text = ":[4]";
//
// label32
//
this.label32.AutoSize = true;
this.label32.Location = new System.Drawing.Point(250, 10);
this.label32.Name = "label32";
this.label32.Size = new System.Drawing.Size(24, 17);
this.label32.TabIndex = 141;
this.label32.Text = "[5]";
//
// lblRaw4
//
this.lblRaw4.Location = new System.Drawing.Point(197, 31);
this.lblRaw4.Name = "lblRaw4";
this.lblRaw4.Size = new System.Drawing.Size(37, 18);
//
// label34
//
this.label34.AutoSize = true;
this.label34.Location = new System.Drawing.Point(295, 10);
this.label34.Name = "label34";
this.label34.Size = new System.Drawing.Size(24, 17);
this.label34.TabIndex = 143;
this.label34.Text = "[6]";
//
// lblRaw3
//
this.lblRaw3.Location = new System.Drawing.Point(151, 31);
this.lblRaw3.Name = "lblRaw3";
this.lblRaw3.Size = new System.Drawing.Size(37, 18);
this.lblRaw3.TabIndex = 167;
//
// label36
//
this.label36.Anchor =
this.label36.AutoSize = true;
this.label36.Location = new System.Drawing.Point(339, 10);
this.label36.Name = "label36"
this.label36.Size = new System.Drawing.Size(24, 17);
this.label36.TabIndex = 145;
this.label36.Text = "[7]"
;
//
// lblRaw2
//
this.lblRaw2.Anchor =
this.lblRaw2.Location = new System.Drawing.Point(105, 31);
this.lblRaw2.Name = "lblRaw2"
this.lblRaw2.Size = new System.Drawing.Size(37, 18);
this.lblRaw2.TabIndex = 166;
//
// label38
//
this.label38.Anchor =
this.label38.AutoSize = true;
this.label38.Location = new System.Drawing.Point(383, 10);
this.label38.Name = "label38"
this.label38.Size = new System.Drawing.Size(24, 17);
this.label38.TabIndex = 147;
this.label38.Text = "[8]"
;
//
// lblRaw19
//
this.lblRaw19.Anchor =
this.lblRaw19.Location = new System.Drawing.Point(289, 85);
this.lblRaw19.Name = "lblRaw19"
this.lblRaw19.Size = new System.Drawing.Size(37, 18);
this.lblRaw19.TabIndex = 165;
//
// label40
//
this.label40.AutoSize = true;
this.label40.Location = new System.Drawing.Point(427, 10);
this.label40.Name = "label40";
this.label40.Size = new System.Drawing.Size(24, 17);
this.label40.TabIndex = 149;
this.label40.Text = "[9]";

this.lblRaw18.Location = new System.Drawing.Point(244, 85);
this.lblRaw18.Name = "lblRaw18";
this.lblRaw18.Size = new System.Drawing.Size(37, 18);
this.lblRaw18.TabIndex = 164;

this.label42.AutoSize = true;
this.label42.Location = new System.Drawing.Point(467, 10);
this.label42.Name = "label42";
this.label42.Size = new System.Drawing.Size(32, 17);
this.label42.TabIndex = 151;
this.label42.Text = "[10]";

this.lblRaw17.Location = new System.Drawing.Point(197, 85);
this.lblRaw17.Name = "lblRaw17";
this.lblRaw17.Size = new System.Drawing.Size(37, 18);
this.lblRaw17.TabIndex = 163;

this.lblRaw16.Location = new System.Drawing.Point(151, 85);
this.lblRaw16.Name = "lblRaw16";
this.lblRaw16.Size = new System.Drawing.Size(37, 18);
this.lblRaw16.TabIndex = 150;
this.lblRaw16.Name = "lblRaw16";
this.lblRaw16.Size = new System.Drawing.Size(37, 18);
this.lblRaw16.TabIndex = 162;

//
// lblRaw8
//
this.lblRaw8.Location = new System.Drawing.Point(377, 31);
this.lblRaw8.Name = "lblRaw8";
this.lblRaw8.Size = new System.Drawing.Size(37, 18);
this.lblRaw8.TabIndex = 154;

//
// lblRaw15
//
this.lblRaw15.Location = new System.Drawing.Point(105, 85);
this.lblRaw15.Name = "lblRaw15";
this.lblRaw15.Size = new System.Drawing.Size(37, 18);
this.lblRaw15.TabIndex = 161;

//
// lblRaw9
//
this.lblRaw9.Location = new System.Drawing.Point(421, 31);
this.lblRaw9.Name = "lblRaw9";
this.lblRaw9.Size = new System.Drawing.Size(37, 18);
this.lblRaw9.TabIndex = 155;

//
// lblRaw14
//
this.lblRaw14.Location = new System.Drawing.Point(58, 85);
this.lblRaw14.Name = "lblRaw14";
this.lblRaw14.Size = new System.Drawing.Size(37, 18);
this.lblRaw14.TabIndex = 160;

//
// lblRaw10
//
this.lblRaw10.Location = new System.Drawing.Point(465, 31);
this.lblRaw10.Name = "lblRaw10";
this.lblRaw10.Size = new System.Drawing.Size(37, 18);
this.lblRaw10.TabIndex = 156;

this.lblRaw13.Location = new System.Drawing.Point(12, 85);
this.lblRaw13.Name = "lblRaw13";
this.lblRaw13.Size = new System.Drawing.Size(37, 18);
this.lblRaw13.TabIndex = 159;

this.lblRaw11.Location = new System.Drawing.Point(510, 31);
this.lblRaw11.Name = "lblRaw11";
this.lblRaw11.Size = new System.Drawing.Size(37, 18);
this.lblRaw11.TabIndex = 157;

this.lblRaw12.Location = new System.Drawing.Point(556, 31);
this.lblRaw12.Name = "lblRaw12";
this.lblRaw12.Size = new System.Drawing.Size(37, 18);
this.lblRaw12.TabIndex = 158;

this.chkShowRawBytes.Location = new System.Drawing.Point(5, 3);
this.chkShowRawBytes.Name = "chkShowRawBytes";
this.chkShowRawBytes.RightToLeft = System.Windows.Forms.RightToLeft.No;
this.chkShowRawBytes.Size = new System.Drawing.Size(140, 21);
this.chkShowRawBytes.TabIndex = 222;
this.chkShowRawBytes.Text = "Show Raw Bytes";
this.chkShowRawBytes.UseVisualStyleBackColor = true;
```csharp
this.chkShowRawBytes.CheckedChanged += new System.EventHandler(this.chkShowRawBytes_CheckedChanged);
//
// chkShowSyncBytes
//
this.chkShowSyncBytes.Checked = true;
this.chkShowSyncBytes.Location = new System.Drawing.Point(6, 4);
this.chkShowSyncBytes.Name = "chkShowSyncBytes";
this.chkShowSyncBytes.RightToLeft = System.Windows.Forms.RightToLeft.No;
this.chkShowSyncBytes.Size = new System.Drawing.Size(140, 21);
this.chkShowSyncBytes.TabIndex = 223;
this.chkShowSyncBytes.Text = "Show Sync Bytes";
this.chkShowSyncBytes.UseVisualStyleBackColor = true;
this.chkShowSyncBytes.CheckedChanged += new System.EventHandler(this.chkShowSyncBytes_CheckedChanged);
//
// tabControl1
//
this.tabControl1.ContextMenuStrip = this.cmPacketMenu;
this.tabControl1.Controls.Add(this.tabSyncBytes);
this.tabControl1.Controls.Add(this.tabRawBytes);
this.tabControl1.Controls.Add(this.tabConfig);
this.tabControl1.Location = new System.Drawing.Point(1, 31);
this.tabControl1.Name = "tabControl1";
this.tabControl1.SelectedIndex = 0;
this.tabControl1.Size = new System.Drawing.Size(622, 174);
this.tabControl1.TabIndex = 296;
//
// tabSyncBytes
//
this.tabSyncBytes.ContextMenuStrip = this.cmPacketMenu;
this.tabSyncBytes.Controls.Add(this.pSyncBytes);
this.tabSyncBytes.Controls.Add(this.chkShowSyncBytes);
this.tabSyncBytes.Location = new System.Drawing.Point(4, 25);
this.tabSyncBytes.Name = "tabSyncBytes";
this.tabSyncBytes.Padding = new System.Windows.Forms.Padding(3);
this.tabSyncBytes.Size = new System.Drawing.Size(614, 145);
this.tabSyncBytes.TabIndex = 0;
this.tabSyncBytes.Text = "Sync";
this.tabSyncBytes.UseVisualStyleBackColor = true;
//
// tabRawBytes
//
this.tabRawBytes.Controls.Add(this.chkShowRawBytes);
this.tabRawBytes.Controls.Add(this.pRawBytes);
this.tabRawBytes.Location = new System.Drawing.Point(4, 25);
this.tabRawBytes.Name = "tabRawBytes";
this.tabRawBytes.Padding = new System.Windows.Forms.Padding(3);
this.tabRawBytes.Size = new System.Drawing.Size(614, 145);
this.tabRawBytes.TabIndex = 1;
```

this.tabRawBytes.Text = "Raw Bytes";
this.tabRawBytes.UseVisualStyleBackColor = true;

//
// tabConfig
//
this.tabConfig.Controls.Add(this.chkDebugConnection);
this.tabConfig.Controls.Add(this.chkIgnore_IsCurrent);
this.tabConfig.Controls.Add(this.udFormDisplay);
this.tabConfig.Controls.Add(this.label1);
this.tabConfig.Location = new System.Drawing.Point(4, 25);
this.tabConfig.Name = "tabConfig";
this.tabConfig.Size = new System.Drawing.Size(614, 145);
this.tabConfig.TabIndex = 2;
this.tabConfig.Text = "Config";
this.tabConfig.UseVisualStyleBackColor = true;

//
// chkDebugConnection
//
this.chkDebugConnection.AutoSize = true;
this.chkDebugConnection.Location = new System.Drawing.Point(52, 45);
this.chkDebugConnection.Name = "chkDebugConnection";
this.chkDebugConnection.RightToLeft = System.Windows.Forms.RightToLeft.Yes;
this.chkDebugConnection.Size = new System.Drawing.Size(144, 21);
this.chkDebugConnection.TabIndex = 303;
this.chkDebugConnection.Text = "Debug Connection";
this.chkDebugConnection.UseVisualStyleBackColor = true;

//
// chkIgnore_IsCurrent
//
this.chkIgnore_IsCurrent.Checked = true;
this.chkIgnore_IsCurrent.Location = new System.Drawing.Point(57, 67);
this.chkIgnore_IsCurrent.Name = "chkIgnore_IsCurrent";
this.chkIgnore_IsCurrent.RightToLeft = System.Windows.Forms.RightToLeft.Yes;
this.chkIgnore_IsCurrent.Size = new System.Drawing.Size(140, 21);
this.chkIgnore_IsCurrent.TabIndex = 300;
this.chkIgnore_IsCurrent.Text = "Ignore IsCurrent";
this.chkIgnore_IsCurrent.UseVisualStyleBackColor = true;

//
// udFormDisplay
//
this.udFormDisplay.Location = new System.Drawing.Point(149, 9);
this.udFormDisplay.Maximum = new decimal(new int[] { 32768, 0, 0, 0 });
this.udFormDisplay.Name = "udFormDisplay";
this.udFormDisplay.Size = new System.Drawing.Size(46, 22);
this.udFormDisplay.TabIndex = 299;
this.udFormDisplay.Value = new decimal(new int[] {
this.udFormDisplay.ValueChanged += new System.EventHandler(this.udFormDisplay_ValueChanged);
//
// label1
//
this.label1.AutoSize = true;
this.label1.Location = new System.Drawing.Point(8, 11);
this.label1.Name = "label1";
this.label1.Size = new System.Drawing.Size(135, 17);
this.label1.TabIndex = 298;
this.label1.Text = "Form Display Speed";
//
// lblError
//
this.lblError.Location = new System.Drawing.Point(111, 212);
this.lblError.Name = "lblError";
this.lblError.Size = new System.Drawing.Size(508, 43);
this.lblError.TabIndex = 210;
this.lblError.TextAlign = System.Drawing.ContentAlignment.MiddleLeft;
//
// chkShowErrors
//
this.chkShowErrors.AutoSize = true;
this.chkShowErrors.Location = new System.Drawing.Point(3, 210);
this.chkShowErrors.Name = "chkShowErrors";
this.chkShowErrors.RightToLeft = System.Windows.Forms.RightToLeft.Yes;
this.chkShowErrors.Size = new System.Drawing.Size(104, 21);
this.chkShowErrors.TabIndex = 307;
this.chkShowErrors.Text = "Show Errors";
this.chkShowErrors.CheckedChanged += new System.EventHandler(this.chkShowErrors_CheckedChanged);
System.EventHandler(this.chkShowErrors_CheckedChanged);
}
private System.Windows.Forms.Label label160;
private System.Windows.Forms.Label lblSync24;
private System.Windows.Forms.Label label162;
private System.Windows.Forms.Label lblSync23;
private System.Windows.Forms.Label label164;
private System.Windows.Forms.Label lblSync22;
private System.Windows.Forms.Label label166;
private System.Windows.Forms.Label label168;
private System.Windows.Forms.Label label169;
private System.Windows.Forms.Label label170;
private System.Windows.Forms.Label label172;
private System.Windows.Forms.Label label173;
private System.Windows.Forms.Label label174;
private System.Windows.Forms.Label label175;
private System.Windows.Forms.Label lblSync0;
private System.Windows.Forms.Label label177;
private System.Windows.Forms.Label label178;
private System.Windows.Forms.Label lblSync21;
private System.Windows.Forms.Label label181;
private System.Windows.Forms.Label label182;
private System.Windows.Forms.Label lblSync20;
private System.Windows.Forms.Label label184;
private System.Windows.Forms.Label lblSync21;
private System.Windows.Forms.Label label186;
private System.Windows.Forms.Label lblSync6;
private System.Windows.Forms.Label label188;
private System.Windows.Forms.Label label189;
private System.Windows.Forms.Label lblSync4;
private System.Windows.Forms.Label label193;
private System.Windows.Forms.Label lblSync3;
private System.Windows.Forms.Label label195;
private System.Windows.Forms.Label lblSync2;
private System.Windows.Forms.Label label197;
private System.Windows.Forms.Label lblSync19;
private System.Windows.Forms.Label label199;
private System.Windows.Forms.Label lblSync18;
private System.Windows.Forms.Label label199;
private System.Windows.Forms.Label lblSync17;
private System.Windows.Forms.Label label196;
private System.Windows.Forms.Label lblSync8;
private System.Windows.Forms.Label label195;
private System.Windows.Forms.Label lblSync15;
private System.Windows.Forms.Label label193;
private System.Windows.Forms.Label lblSync9;
private System.Windows.Forms.Label label194;
private System.Windows.Forms.Label lblSync14;
private System.Windows.Forms.Label label190;
private System.Windows.Forms.Label lblSync13;
private System.Windows.Forms.Label label191;
private System.Windows.Forms.Label lblRaw5;
private System.Windows.Forms.Label label2;
private System.Windows.Forms.Label label4;
private System.Windows.Forms.Label lblRaw24;
private System.Windows.Forms.Label label6;
private System.Windows.Forms.Label label8;
private System.Windows.Forms.Label lblRaw22;
private System.Windows.Forms.Label label10;
private System.Windows.Forms.Label label11;
private System.Windows.Forms.Label label12;
private System.Windows.Forms.Label label13;
private System.Windows.Forms.Label label14;
private System.Windows.Forms.Label label15;
private System.Windows.Forms.Label label16;
private System.Windows.Forms.Label label17;
private System.Windows.Forms.Label label18;
private System.Windows.Forms.Label lblRaw0;
private System.Windows.Forms.Label label20;
private System.Windows.Forms.Label label21;
private System.Windows.Forms.Label lblRaw21;
private System.Windows.Forms.Label lblRaw1;
private System.Windows.Forms.Label lblRaw20;
private System.Windows.Forms.Label label27;
private System.Windows.Forms.Label lblRaw7;
private System.Windows.Forms.Label label29;
private System.Windows.Forms.Label lblRaw6;
private System.Windows.Forms.Label label31;
private System.Windows.Forms.Label label32;
private System.Windows.Forms.Label lblRaw4;
private System.Windows.Forms.Label label34;
private System.Windows.Forms.Label lblRaw3;
private System.Windows.Forms.Label label36;
private System.Windows.Forms.Label lblRaw2;
private System.Windows.Forms.Label label38;
private System.Windows.Forms.Label lblRaw19;
private System.Windows.Forms.Label label40;
private System.Windows.Forms.Label lblRaw18;
private System.Windows.Forms.Label label42;
private System.Windows.Forms.Label lblRaw17;
private System.Windows.Forms.Label lblRaw16;
private System.Windows.Forms.Label lblRaw8;
private System.Windows.Forms.Label lblRaw15;
private System.Windows.Forms.Label lblRaw9;
private System.Windows.Forms.Label lblRaw14;
private System.Windows.Forms.Label lblRaw10;
private System.Windows.Forms.Label lblRaw13;
private System.Windows.Forms.Label lblRaw12;
private System.Windows.Forms.CheckBox chkShowRawBytes;
private System.Windows.Forms.CheckBox chkShowSyncBytes;
private System.Windows.Forms.TabControl tabControl1;
private System.Windows.Forms.TabPage tabSyncBytes;
private System.Windows.Forms.TabPage tabRawBytes;
private System.Windows.Forms.Label lblError;
private System.Windows.Forms.TabPage tabConfig;
private System.Windows.Forms.Label label1;
private System.Windows.Forms.Label lblSensorParse;
private System.Windows.Forms.CheckBox chkIgnore_IsCurrent;
private System.Windows.Forms.ToolStripMenuItem displayToolStripMenuItem;
private System.Windows.Forms.ToolStripMenuItem startToolStripMenuItem1;
private System.Windows.Forms.ToolStripMenuItem stopToolStripMenuItem;
private System.Windows.Forms.ToolStripMenuItem clearToolStripMenuItem;
private System.Windows.Forms.ToolStripMenuItem outputPacketDataToFileToolStripMenuItem;
private System.Windows.Forms.ToolStripMenuItem startToolStripMenuItem2;
private System.Windows.Forms.ToolStripMenuItem stopToolStripMenuItem1;
private System.Windows.Forms.CheckBox chkShowErrors;
private System.Windows.Forms.CheckBox chkDebugConnection;
private System.Windows.Forms.ToolStripMenuItem explanationOfThisFormToolStripMenuItem1;
)
private System.Windows.Forms.ToolStripMenuItem explanationOfThisFormToolStripMenuItem1;
)
namespace roomba_term
{
    partial class frmSensors
    {
        private System.ComponentModel.IContainer components = null;

        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code

        private void InitializeComponent()
        {
            this.components = new System.ComponentModel.Container();
            this.gSensors = new System.Windows.Forms.GroupBox();
            this.cmSensorMenu = new System.Windows.Forms.ContextMenuStrip(this.components);
            this.displayToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
            this.startToolStripMenuItem1 = new System.Windows.Forms.ToolStripMenuItem();
            this.stopToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
            this.clearToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
            this.outputPacketDataToFileToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
            this.startToolStripMenuItem2 = new System.Windows.Forms.ToolStripMenuItem();
            this.stopToolStripMenuItem1 = new System.Windows.Forms.ToolStripMenuItem();
            this.lblChargeStateRaw = new System.Windows.Forms.Label();
            this.lblButtonsRaw = new System.Windows.Forms.Label();
            this.lblBumps_WheelDrops = new System.Windows.Forms.Label();
            this.label6 = new System.Windows.Forms.Label();
            this.label5 = new System.Windows.Forms.Label();
            this.label4 = new System.Windows.Forms.Label();
            this.label2 = new System.Windows.Forms.Label();
            this.label1 = new System.Windows.Forms.Label();
            this.label76 = new System.Windows.Forms.Label();
            this.lblButtons = new System.Windows.Forms.Label();
            this.label68 = new System.Windows.Forms.Label();
            this.lblVirtual_Wall = new System.Windows.Forms.Label();
            this.label70 = new System.Windows.Forms.Label();
            this.lblCliff_Right = new System.Windows.Forms.Label();
            this.label72 = new System.Windows.Forms.Label();
            this.lblCliff_Front_Right = new System.Windows.Forms.Label();
            this.label45 = new System.Windows.Forms.Label();
            this.lblCliff_Front_Left = new System.Windows.Forms.Label();

        }

        #endregion
    }
}

this.label41 = new System.Windows.Forms.Label();
this.lblCliff_Left = new System.Windows.Forms.Label();
this.label37 = new System.Windows.Forms.Label();
this.lblWall = new System.Windows.Forms.Label();
this.label35 = new System.Windows.Forms.Label();
this.lblBumps_WheelDropsRaw = new System.Windows.Forms.Label();
this.lblTempF = new System.Windows.Forms.Label();
this.label3 = new System.Windows.Forms.Label();
this.lblChargeState = new System.Windows.Forms.Label();
this.label38 = new System.Windows.Forms.Label();
this.lblVoltage = new System.Windows.Forms.Label();
this.lblCurrent = new System.Windows.Forms.Label();
this.lblTemp = new System.Windows.Forms.Label();
this.lblCharge = new System.Windows.Forms.Label();
this.lblCapacity = new System.Windows.Forms.Label();
this.label32 = new System.Windows.Forms.Label();
this.lblDistance_Traveled = new System.Windows.Forms.Label();
this.lblAngle_Traveled = new System.Windows.Forms.Label();
this.label34 = new System.Windows.Forms.Label();
this.label29 = new System.Windows.Forms.Label();
this.lblRemote = new System.Windows.Forms.Label();
this.label78 = new System.Windows.Forms.Label();
this.lblDirt_Detector_Right = new System.Windows.Forms.Label();
this.label80 = new System.Windows.Forms.Label();
this.lblDirt_Detector_Left = new System.Windows.Forms.Label();
this.lblBytesRCVD = new System.Windows.Forms.Label();
this.lblSensorParse = new System.Windows.Forms.Label();
this.tabControl1 = new System.Windows.Forms.TabControl();
this.tabDisplay = new System.Windows.Forms.TabPage();
this.groupBox1 = new System.Windows.Forms.GroupBox();
this.lblMotorOvercurrentsRaw = new System.Windows.Forms.Label();
this.label8 = new System.Windows.Forms.Label();
this.lblMotorOvercurrents = new System.Windows.Forms.Label();
this.udFormDisplay = new System.Windows.Forms.NumericUpDown();
this.label7 = new System.Windows.Forms.Label();
this.cmSensorMenu.SuspendLayout();
this.cmSensorMenu.SuspendLayout();
this.tabControl1.SuspendLayout();
this.tabDisplay.SuspendLayout();
this.groupBox1.SuspendLayout();
this.tabConfig.SuspendLayout();
((System.ComponentModel.ISupportInitialize)(this.udFormDisplay)).BeginInit();
this.SuspendLayout();

// gSensors
//
this.gSensors.BackColor = System.Drawing.Color.Transparent;
this.gSensors.ContextMenuStrip = this.cmSensorMenu;
this.gSensors.Controls.Add(this.lblMotorOvercurrentsRaw);
this.gSensors.Controls.Add(this.lblChargeStateRaw);
this.gSensors.Controls.Add(this.label8);
this.gSensors.Controls.Add(this.lblButtonsRaw);
this.gSensors.Controls.Add(this.lblMotorOvercurrents);
this.gSensors.Controls.Add(this.lblBumps_WheelDrops);
this.gSensors.Controls.Add(this.label6);
this.gSensors.Controls.Add(this.label5);
this.gSensors.Controls.Add(this.label4);
this.gSensors.Controls.Add(this.label2);
this.gSensors.Controls.Add(this.label1);
this.gSensors.Controls.Add(this.label76);
this.gSensors.Controls.Add(this.lblButtons);
this.gSensors.Controls.Add(this.label68);
this.gSensors.Controls.Add(this.lblVirtual_Wall);
this.gSensors.Controls.Add(this.label70);
this.gSensors.Controls.Add(this.lblCliff_Right);
this.gSensors.Controls.Add(this.label72);
this.gSensors.Controls.Add(this.lblCliff_Front_Right);
this.gSensors.Controls.Add(this.label45);
this.gSensors.Controls.Add(this.lblCliff_Front_Left);
this.gSensors.Controls.Add(this.label41);
this.gSensors.Controls.Add(this.lblCliff_Left);
this.gSensors.Controls.Add(this.label37);
this.gSensors.Controls.Add(this.lblWall);
this.gSensors.Controls.Add(this.label35);
this.gSensors.Controls.Add(this.lblBumps_WheelDropsRaw);
this.gSensors.Controls.Add(this.lblTempF);
this.gSensors.Controls.Add(this.label3);
this.gSensors.Controls.Add(this.lblChargeState);
this.gSensors.Controls.Add(this.label38);
this.gSensors.Controls.Add(this.lblVoltage);
this.gSensors.Controls.Add(this.lblCurrent);
this.gSensors.Controls.Add(this.lblTemp);
this.gSensors.Controls.Add(this.lblCharge);
this.gSensors.Controls.Add(this.lblCapacity);
this.gSensors.Controls.Add(this.label32);
this.gSensors.Controls.Add(this.lblDistance_Traveled);
this.gSensors.Controls.Add(this.lblAngle_Traveled);
this.gSensors.ForeColor = System.Drawing.SystemColors.ControlText;
this.gSensors.Location = new System.Drawing.Point(5, 2);
this.gSensors.Name = "gSensors";
this.gSensors.RightToLeft = System.Windows.Forms.RightToLeft.Yes;
this.gSensors.Size = new System.Drawing.Size(224, 537);
this.gSensors.TabIndex = 295;
this.gSensors.TabStop = false;
this.gSensors.Text = "Sensors";
//
// cmSensorMenu
//
this.cmSensorMenu.Items.AddRange(new System.Windows.Forms.ToolStripItem[]{
    this.displayToolStripMenuItem,
    this.outputPacketDataToFileToolStripMenuItem});
this.cmSensorMenu.Name = "cmConnectionMenu";
this.cmSensorMenu.Size = new System.Drawing.Size(180, 48);
//
// displayToolStripMenuItem
//
this.displayToolStripMenuItem.DropDownItems.AddRange(new
System.Windows.Forms.ToolStripItem[]{
    this.startToolStripMenuItem1,
    this.stopToolStripMenuItem,
    this.clearToolStripMenuItem});
this.displayToolStripMenuItem.Name = "displayToolStripMenuItem";
this.displayToolStripMenuItem.Size = new System.Drawing.Size(179, 22);
this.displayToolStripMenuItem.Text = "Display";
//
// startToolStripMenuItem1
//
this.startToolStripMenuItem1.Name = "startToolStripMenuItem1";
this.startToolStripMenuItem1.Size = new System.Drawing.Size(123, 22);
this.startToolStripMenuItem1.Text = "Start";
//
// stopToolStripMenuItem
//
this.stopToolStripMenuItem.Name = "stopToolStripMenuItem";
this.stopToolStripMenuItem.Size = new System.Drawing.Size(123, 22);
this.stopToolStripMenuItem.Text = "Stop";
//
// clearToolStripMenuItem
//
this.clearToolStripMenuItem.Name = "clearToolStripMenuItem";
this.clearToolStripMenuItem.Size = new System.Drawing.Size(123, 22);
this.clearToolStripMenuItem.Text = "Clear";
this.clearToolStripMenuItem.Click += new
System.EventHandler(this.clearToolStripMenuItem_Click);
//
// outputPacketDataToFileToolStripMenuItem
//
this.outputPacketDataToFileToolStripMenuItem.DropDownItems.AddRange(new
System.Windows.Forms.ToolStripItem[]{
    this.startToolStripMenuItem2,
    this.stopToolStripMenuItem1});
this.outputPacketDataToFileToolStripMenuItem.Name = "outputPacketDataToFileToolStripMenuItem";
this.outputPacketDataToFileToolStripMenuItem.Size = new System.Drawing.Size(179, 22);
this.outputPacketDataToFileToolStripMenuItem.Text = "Output to File";
//
// startToolStripMenuItem2
//
this.startToolStripMenuItem2.Name = "startToolStripMenuItem2";
this.startToolStripMenuItem2.Size = new System.Drawing.Size(122, 22);
this.startToolStripMenuItem2.Text = "Start";
//
// stopToolStripMenuItem1
//
this.stopToolStripMenuItem1.Name = "stopToolStripMenuItem1";
this.stopToolStripMenuItem1.Size = new System.Drawing.Size(122, 22);
this.stopToolStripMenuItem1.Text = "Stop";
//
// lblChargeStateRaw
//
this.lblChargeStateRaw.Anchor =
this.lblChargeStateRaw.Location = new System.Drawing.Point(162, 309);
this.lblChargeStateRaw.Name = "lblChargeStateRaw";
this.lblChargeStateRaw.Size = new System.Drawing.Size(53, 19);
this.lblChargeStateRaw.TabIndex = 230;
//
// lblButtonsRaw
//
this.lblButtonsRaw.Anchor =
this.lblButtonsRaw.Location = new System.Drawing.Point(162, 201);
this.lblButtonsRaw.Name = "lblButtonsRaw";
this.lblButtonsRaw.Size = new System.Drawing.Size(53, 19);
this.lblButtonsRaw.TabIndex = 229;
//
// lblBumps_WheelDrops
//
this.lblBumps_WheelDrops.Anchor =
this.lblBumps_WheelDrops.Location = new System.Drawing.Point(7, 48);
this.lblBumps_WheelDrops.Name = "lblBumps_WheelDrops";
this.lblBumps_WheelDrops.Size = new System.Drawing.Size(210, 43);
this.lblBumps_WheelDrops.TabIndex = 228;
//
// label6
//
this.label6.Anchor =
this.label6.AutoSize = true;
this.label6.Location = new System.Drawing.Point(1, 511);
this.label6.Name = "label6";
this.label6.Size = new System.Drawing.Size(62, 17);
this.label6.TabIndex = 227;
this.label6.Text = "Capacity";

//
// label5
//
this.label5.Anchor =
this.label5.AutoSize = true;
this.label5.Location = new System.Drawing.Point(9, 490);
this.label5.Name = "label5";
this.label5.Size = new System.Drawing.Size(54, 17);
this.label5.TabIndex = 226;
this.label5.Text = "Charge";

//
// label4
//
this.label4.Anchor =
this.label4.AutoSize = true;
this.label4.Location = new System.Drawing.Point(19, 469);
this.label4.Name = "label4";
this.label4.Size = new System.Drawing.Size(44, 17);
this.label4.TabIndex = 225;
this.label4.Text = "Temp";

//
// label2
//
this.label2.Anchor =
this.label2.AutoSize = true;
this.label2.Location = new System.Drawing.Point(8, 451);
this.label2.Name = "label2";
this.label2.Size = new System.Drawing.Size(55, 17);
this.label2.TabIndex = 224;
this.label2.Text = "Current";

//
// label1
//
this.label1.Anchor =
this.label1.AutoSize = true;
this.label1.Location = new System.Drawing.Point(7, 429);
this.label1.Name = "label1";
this.label1.Size = new System.Drawing.Size(56, 17);
this.label1.TabIndex = 223;
this.label1.Text = "Voltage";
// label76
//
this.label76.AutoSize = true;
this.label76.Location = new System.Drawing.Point(104, 201);
this.label76.Name = "label76";
this.label76.Size = new System.Drawing.Size(56, 17);
this.label76.TabIndex = 106;
this.label76.Text = "Buttons";
//
// lblButtons
//
this.lblButtons.Location = new System.Drawing.Point(7, 223);
this.lblButtons.Name = "lblButtons";
this.lblButtons.Size = new System.Drawing.Size(208, 39);
this.lblButtons.TabIndex = 105;
//
// label68
//
this.label68.AutoSize = true;
this.label68.Location = new System.Drawing.Point(81, 179);
this.label68.Name = "label68";
this.label68.Size = new System.Drawing.Size(79, 17);
this.label68.TabIndex = 98;
this.label68.Text = "Virtual Wall";
//
// lblVirtual_Wall
//
this.lblVirtual_Wall.Location = new System.Drawing.Point(163, 179);
this.lblVirtual_Wall.Name = "lblVirtual_Wall";
this.lblVirtual_Wall.Size = new System.Drawing.Size(52, 18);
this.lblVirtual_Wall.TabIndex = 97;
//
// label70
//
this.label70.AutoSize = true;
this.label70.Location = new System.Drawing.Point(163, 179);
this.label70.Name = "label70";
this.label70.Size = new System.Drawing.Size(52, 18);
this.label70.TabIndex = 97;
this.label70.AutoSize = true;
this.label70.Location = new System.Drawing.Point(92, 158);
this.label70.Name = "label70";
this.label70.Size = new System.Drawing.Size(68, 17);
this.label70.TabIndex = 96;
this.label70.Text = "Cliff Right";
//
// lblCliff_Right
//
this.lblCliff_Right.Location = new System.Drawing.Point(163, 158);
this.lblCliff_Right.Name = "lblCliff_Right";
this.lblCliff_Right.Size = new System.Drawing.Size(52, 18);
this.lblCliff_Right.TabIndex = 95;
//
// label72
//
this.label72.AutoSize = true;
this.label72.Location = new System.Drawing.Point(55, 138);
this.label72.Name = "label72";
this.label72.Size = new System.Drawing.Size(105, 17);
this.label72.TabIndex = 94;
this.label72.Text = "Cliff Front Right";
//
// lblCliff_Front_Right
//
this.lblCliff_Front_Right.Location = new System.Drawing.Point(163, 138);
this.lblCliff_Front_Right.Name = "lblCliff_Front_Right";
this.lblCliff_Front_Right.Size = new System.Drawing.Size(52, 18);
this.lblCliff_Front_Right.TabIndex = 93;
//
// label45
//
this.label45.AutoSize = true;
this.label45.Location = new System.Drawing.Point(64, 117);
this.label45.Name = "label45";
this.label45.Size = new System.Drawing.Size(96, 17);
this.label45.TabIndex = 92;
this.label45.Text = "Cliff Front Left";

//
// lblCliff_Front_Left
//
this.lblCliff_Front_Left.Location = new System.Drawing.Point(163, 117);
this.lblCliff_Front_Left.Name = "lblCliff_Front_Left";
this.lblCliff_Front_Left.Size = new System.Drawing.Size(52, 18);
this.lblCliff_Front_Left.TabIndex = 91;

//
// label41
//
this.label41.AutoSize = true;
this.label41.Location = new System.Drawing.Point(100, 96);
this.label41.Name = "label41";
this.label41.Size = new System.Drawing.Size(63, 17);
this.label41.TabIndex = 90;
this.label41.Text = "Cliff Left ";

//
// lblCliff_Left
//
this.lblCliff_Left.Location = new System.Drawing.Point(163, 96);
this.lblCliff_Left.Name = "lblCliff_Left";
this.lblCliff_Left.Size = new System.Drawing.Size(52, 18);
this.lblCliff_Left.TabIndex = 89;

//
// label37
//
this.label37.AutoSize = true;
this.label37.Location = new System.Drawing.Point(8, 170);
this.label37.Name = "label37";
this.label37.Size = new System.Drawing.Size(35, 17);
this.label37.TabIndex = 88;
this.label37.Text = "Wall";

//
// lblWall
this.lblWall.Anchor = 
this.lblWall.Location = new System.Drawing.Point(8, 191);
this.lblWall.Name = "lblWall";
this.lblWall.Size = new System.Drawing.Size(52, 18);
this.lblWall.TabIndex = 87;

// // label35 //
this.label35.Anchor = 
this.label35.AutoSize = true;
this.label35.Location = new System.Drawing.Point(7, 22);
this.label35.Name = "label35";
this.label35.Size = new System.Drawing.Size(133, 17);
this.label35.TabIndex = 86;
this.label35.Text = "Bumps/WheelDrops";

// // lblBumps_WheelDropsRaw //
this.lblBumps_WheelDropsRaw.Anchor = 
this.lblBumps_WheelDropsRaw.Location = new System.Drawing.Point(163, 25);
this.lblBumps_WheelDropsRaw.Name = "lblBumps_WheelDropsRaw";
this.lblBumps_WheelDropsRaw.Size = new System.Drawing.Size(53, 19);
this.lblBumps_WheelDropsRaw.TabIndex = 85;

// // lblTempF //
this.lblTempF.Anchor = 
this.lblTempF.Location = new System.Drawing.Point(164, 470);
this.lblTempF.Name = "lblTempF";
this.lblTempF.Size = new System.Drawing.Size(53, 18);
this.lblTempF.TabIndex = 82;

// // label3 //
this.label3.Anchor = 
this.label3.AutoSize = true;
this.label3.Location = new System.Drawing.Point(138, 469);
this.label3.Name = "label3";
this.label3.Size = new System.Drawing.Size(26, 17);
this.label3.TabIndex = 81;
this.label3.Text = "(F)";

//
// lblChargeState
//
this.lblChargeState.Location = new System.Drawing.Point(7, 331);
this.lblChargeState.Name = "lblChargeState";
this.lblChargeState.Size = new System.Drawing.Size(208, 18);
this.lblChargeState.TabIndex = 68;

//
// label38
//
this.label38.AutoSize = true;
this.label38.Location = new System.Drawing.Point(69, 309);
this.label38.Name = "label38";
this.label38.Size = new System.Drawing.Size(91, 17);
this.label38.TabIndex = 67;
this.label38.Text = "Charge State";

//
// lblVoltage
//
this.lblVoltage.Location = new System.Drawing.Point(69, 429);
this.lblVoltage.Name = "lblVoltage";
this.lblVoltage.Size = new System.Drawing.Size(67, 18);
this.lblVoltage.TabIndex = 70;

//
// lblCurrent
//
this.lblCurrent.Location = new System.Drawing.Point(69, 450);
this.lblCurrent.Name = "lblCurrent";
this.lblCurrent.Size = new System.Drawing.Size(67, 18);
this.lblCurrent.TabIndex = 72;

//
// lblTemp

// // lblCharge //
//

// // lblCapacity //
//

// // label32 //
//

// // lblDistance_Traveled //
//
```csharp
this.label78.AutoSize = true;
this.label78.Location = new System.Drawing.Point(49, 41);
this.label78.Name = "label78";
this.label78.Size = new System.Drawing.Size(125, 17);
this.label78.TabIndex = 104;
this.label78.Text = "Dirt Detector Right";
//
// lblDirt_Detector_Right
//
this.lblDirt_Detector_Right.Location = new System.Drawing.Point(180, 41);
this.lblDirt_Detector_Right.Name = "lblDirt_Detector_Right";
this.lblDirt_Detector_Right.Size = new System.Drawing.Size(37, 18);
this.lblDirt_Detector_Right.TabIndex = 103;
//
// label80
//
this.label80.AutoSize = true;
this.label80.Location = new System.Drawing.Point(58, 20);
this.label80.Name = "label80";
this.label80.Size = new System.Drawing.Size(116, 17);
this.label80.TabIndex = 102;
this.label80.Text = "Dirt Detector Left";
//
// lblDirt_Detector_Left
//
this.lblDirt_Detector_Left.Location = new System.Drawing.Point(180, 20);
this.lblDirt_Detector_Left.Name = "lblDirt_Detector_Left";
this.lblDirt_Detector_Left.Size = new System.Drawing.Size(37, 18);
this.lblDirt_Detector_Left.TabIndex = 101;
//
// lblBytesRCVD
//
this.lblBytesRCVD.Location = new System.Drawing.Point(114, 4);
this.lblBytesRCVD.Name = "lblBytesRCVD";
this.lblBytesRCVD.Size = new System.Drawing.Size(123, 17);
this.lblBytesRCVD.TabIndex = 222;
this.lblBytesRCVD.Text = "Bytes RCVD";
```
this.lblBytesRCVD.TextAlign = System.Drawing.ContentAlignment.MiddleRight;
//
// lblSensorParse
//
this.lblSensorParse.Location = new System.Drawing.Point(114, 3);
this.lblSensorParse.Name = "lblSensorParse";
this.lblSensorParse.Size = new System.Drawing.Size(127, 24);
this.lblSensorParse.TabIndex = 307;
this.lblSensorParse.TextAlign = System.Drawing.ContentAlignment.MiddleLeft;
//
// tabControl1
//
this.tabControl1.Controls.Add(this.tabDisplay);
this.tabControl1.Controls.Add(this.tabConfig);
this.tabControl1.Location = new System.Drawing.Point(2, 26);
this.tabControl1.Name = "tabControl1";
this.tabControl1.SelectedIndex = 0;
this.tabControl1.Size = new System.Drawing.Size(243, 662);
this.tabControl1.TabIndex = 296;
//
// tabDisplay
//
this.tabDisplay.Controls.Add(this.groupBox1);
this.tabDisplay.Controls.Add(this.gSensors);
this.tabDisplay.Location = new System.Drawing.Point(4, 25);
this.tabDisplay.Name = "tabDisplay";
this.tabDisplay.Padding = new System.Windows.Forms.Padding(3);
this.tabDisplay.Size = new System.Drawing.Size(235, 633);
this.tabDisplay.TabIndex = 0;
this.tabDisplay.Text = "Display";
this.tabDisplay.UseVisualStyleBackColor = true;
//
// groupBox1
//
this.groupBox1.BackColor = System.Drawing.Color.Transparent;
this.groupBox1.ContextMenuStrip = this.cmSensorMenu;
this.groupBox1.Controls.Add(this.lblDirt_Detector_Right);
this.groupBox1.Controls.Add(this.label80);
this.groupBox1.Controls.Add(this.label78);
this.groupBox1.Controls.Add(this.lblDirt_Detector_Left);
this.groupBox1.ForeColor = System.Drawing.SystemColors.ControlText;
this.groupBox1.Location = new System.Drawing.Point(4, 539);
this.groupBox1.Name = "groupBox1";
this.groupBox1.RightToLeft = System.Windows.Forms.RightToLeft.Yes;
this.groupBox1.Size = new System.Drawing.Size(226, 87);
this.groupBox1.TabIndex = 296;
this.groupBox1.TabStop = false;
this.groupBox1.Text = "Roomba Only";
// lblMotorOvercurrentsRaw
//
this.lblMotorOvercurrentsRaw.Anchor =
this.lblMotorOvercurrentsRaw.Location = new System.Drawing.Point(177, 355);
this.lblMotorOvercurrentsRaw.Name = "lblMotorOvercurrentsRaw";
this.lblMotorOvercurrentsRaw.Size = new System.Drawing.Size(37, 18);
this.lblMotorOvercurrentsRaw.TabIndex = 107;
/
// label8
//
this.label8.Anchor =
this.label8.AutoSize = true;
this.label8.Location = new System.Drawing.Point(2, 353);
this.label8.Name = "label8";
this.label8.Size = new System.Drawing.Size(131, 17);
this.label8.TabIndex = 106;
this.label8.Text = "Motor Overcurrents";
/
// lblMotorOvercurrents
//
this.lblMotorOvercurrents.Anchor =
this.lblMotorOvercurrents.Location = new System.Drawing.Point(5, 377);
this.lblMotorOvercurrents.Name = "lblMotorOvercurrents";
this.lblMotorOvercurrents.Size = new System.Drawing.Size(209, 47);
this.lblMotorOvercurrents.TabIndex = 105;
this.lblMotorOvercurrents.TextAlign = System.Drawing.ContentAlignment.MiddleCenter;
/
// tabConfig
//
this.tabConfig.Controls.Add(this.chkPersist);
this.tabConfig.Controls.Add(this.chkAccurateSensors);
this.tabConfig.Controls.Add(this.udFormDisplay);
this.tabConfig.Controls.Add(this.label7);
this.tabConfig.Location = new System.Drawing.Point(4, 25);
this.tabConfig.Name = "tabConfig";
this.tabConfig.Padding = new System.Windows.Forms.Padding(3);
this.tabConfig.Size = new System.Drawing.Size(235, 655);
this.tabConfig.TabIndex = 1;
this.tabConfig.Text = "Config";
this.tabConfig.UseVisualStyleBackColor = true;
/
// chkPersist
//
this.chkPersist.Checked = true;
this.chkPersist.Location = new System.Drawing.Point(5, 158);
this.chkPersist.Name = "chkPersist";
this.chkPersist.Size = new System.Drawing.SizeF(227, 63);
this.chkPersist.TabIndex = 303;
this.chkPersist.Text = "Persist Last Sensor Reading (Do not clear if no data recieved)";
this.chkPersist.UseVisualStyleBackColor = true;
//
// chkAccurateSensors
//
this.chkAccurateSensors.Location = new System.Drawing.Point(5, 34);
this.chkAccurateSensors.Name = "chkAccurateSensors";
this.chkAccurateSensors.Size = new System.Drawing.SizeF(227, 115);
this.chkAccurateSensors.TabIndex = 302;
this.chkAccurateSensors.Text = "Accurate sensor Display (Means indicators will flash since some polls are dropped" +
" or never receieved. This is technically the most accurate display of sensor data" +
"");
this.chkAccurateSensors.UseVisualStyleBackColor = true;
//
// udFormDisplay
//
this.udFormDisplay.Location = new System.Drawing.Point(155, 6);
this.udFormDisplay.Maximum = new decimal(new int[] {32768,
0,
0,
0,
0});
this.udFormDisplay.Name = "udFormDisplay";
this.udFormDisplay.Size = new System.Drawing.SizeF(46, 22);
this.udFormDisplay.TabIndex = 301;
this.udFormDisplay.Value = new decimal(new int[] {25,
0,
0,
0});
//
// label7
//
this.label7.Anchor =
this.label7.AutoSize = true;
this.label7.Location = new System.Drawing.Point(14, 8);
this.label7.Name = "label7";
this.label7.Size = new System.Drawing.SizeF(135, 17);
this.label7.TabIndex = 300;
this.label7.Text = "Form Display Speed";
// frmSensors
//
this.AutoScaleDimensions = new System.Drawing.SizeF(8F, 16F);
this.ClientSize = new System.Drawing.Size(248, 691);
this.ContextMenuStrip = this.cmSensorMenu;
this.Controls.Add(this.lblBytesRCVD);
this.Controls.Add(this.lblSensorParse);
this.Controls.Add(this.tabControl1);
this.FormBorderStyle = System.Windows.Forms.FormBorderStyle.FixedToolWindow;
this.Name = "frmSensors";
this.Text = "Sensors";
this.FormClosing += new System.Windows.Forms.FormClosingEventHandler(this.frmSensors_FormClosing);
this.Controls.SetChildIndex(this.tabControl1, 0);
this.Controls.SetChildIndex(this.lblSensorParse, 0);
this.Controls.SetChildIndex(this.lblBytesRCVD, 0);
gSensors.ResumeLayout(false);
gSensors.PerformLayout();
cmSensorMenu.ResumeLayout(false);
tabControl1.ResumeLayout(false);
tabDisplay.ResumeLayout(false);
groupBox1.ResumeLayout(false);
tabConfig.ResumeLayout(false);
((System.ComponentModel.ISupportInitialize)(udFormDisplay)).EndInit();
tabConfig.PerformLayout();
this.ResumeLayout(false);
this.PerformLayout();
}

#endregion

private System.Windows.Forms.GroupBox gSensors;
internal System.Windows.Forms.Label lblBytesRCVD;
private System.Windows.Forms.Label label76;
private System.Windows.Forms.Label lblButtons;
private System.Windows.Forms.Label label78;
private System.Windows.Forms.Label lblDirt_Detector_Right;
private System.Windows.Forms.Label label80;
private System.Windows.Forms.Label lblDirt_Detector_Left;
private System.Windows.Forms.Label label68;
private System.Windows.Forms.Label lblVirtual_Wall;
private System.Windows.Forms.Label label70;
private System.Windows.Forms.Label lblCliff_Right;
private System.Windows.Forms.Label label72;
private System.Windows.Forms.Label lblCliff_Front_Right;
private System.Windows.Forms.Label label45;
private System.Windows.Forms.Label lblCliff_Front_Left;
private System.Windows.Forms.Label label41;
private System.Windows.Forms.Label lblCliff_Left;
private System.Windows.Forms.Label label37;
private System.Windows.Forms.Label lblWall;
private System.Windows.Forms.Label label35;
private System.Windows.Forms.Label lblBumps_WheelDropsRaw;
private System.Windows.Forms.Label label36;
private System.Windows.Forms.Label lblTempF;
private System.Windows.Forms.Label label3;
private System.Windows.Forms.Label lblChargeState;
private System.Windows.Forms.Label label38;
private System.Windows.Forms.Label lblVoltage;
private System.Windows.Forms.Label lblCurrent;
private System.Windows.Forms.Label lblTemp;
private System.Windows.Forms.Label lblCharge;
private System.Windows.Forms.Label lblCapacity;
private System.Windows.Forms.Label label29;
private System.Windows.Forms.Label lblRemote;
private System.Windows.Forms.Label label32;
private System.Windows.Forms.Label lblDistance_Traveled;
private System.Windows.Forms.Label label34;
private System.Windows.Forms.Label label1;
private System.Windows.Forms.Label label6;
private System.Windows.Forms.Label label5;
private System.Windows.Forms.Label label4;
private System.Windows.Forms.Label label2;
private System.Windows.Forms.Label lblSensorParse;
private System.Windows.Forms.TabControl tabControl1;
private System.Windows.Forms.TabPage tabDisplay;
private System.Windows.Forms.TabPage tabConfig;
private System.Windows.Forms.ToolStripMenuItem displayToolStripMenuItem;
private System.Windows.Forms.ToolStripMenuItem startToolStripMenuItem1;
private System.Windows.Forms.ToolStripMenuItem stopToolStripMenuItem;
private System.Windows.Forms.ToolStripMenuItem clearToolStripMenuItem;
private System.Windows.Forms.ToolStripMenuItem outputPacketDataToFileToolStripMenuItem;
private System.Windows.Forms.ToolStripMenuItem startToolStripMenuItem2;
private System.Windows.Forms.ToolStripMenuItem stopToolStripMenuItem1;
private System.Windows.Forms.CheckBox chkAccurateSensors;
private System.Windows.Forms.CheckBox chkPersist;
private System.Windows.Forms.GroupBox groupBox1;
private System.Windows.Forms.Label label8;
private System.Windows.Forms.Label lblMotorOvercurrents;
private System.Windows.Forms.Label lblBumps_WheelDrops;
private System.Windows.Forms.Label label39;
private System.Windows.Forms.Label lblButtonsRaw;
}
using System;
using System.IO.Ports;
using System.Reflection;
using System.Collections.Generic;
using Kevin_Logging;

namespace RoombaSCI
{
    public class RoombaException : System.ApplicationException
    {
        public RoombaException(string message) : base(message)
        {
        }
    }

    /// <summary>
    /// This class is is a representation of the various tasks that Roomba can carry out as specified in the Roomba SCI
    /// </summary>
    ///
    /// [Serializable]
    public class Roomba
    {
        #region Constants

        private const char c_cOpen = '[';
        private const char c_cClose = ']';

        private const string c_sBytesToRead = "Bytes to Read: ";
        private const string c_sBytesRetrieved = " Sensor Byte(s) Retrieved from IO: ";

        private const string c_sExecuteAction = "Execute Action: ";

        private const string c_sParsingData = "Parsing Sensor Data";
        private const string c_sIgnored = "Not enough data to parse, Ignored.";

        private const string c_sRoomba = "Roomba";
        private const string c_sExecuteActionFail = "Execute Action Fail: ";
        private const string c_sPacketData = "Packet Data";
        private const string c_sIO_DataRecieved = "IO Data Received";
        private const string c_sIO_PinChanged = "IO PinChanged";

        #endregion

        #region Events

        public delegate void Roomba_IO_Handler(byte[] RecievedBytes);
        public delegate void Roomba_Text_IO_Handler(string sRecievedText);

        #endregion
    }
}
public delegate void Roomba_Motor_Action_Handler(byte bSend_Motor_Byte, bool bSendResult);

public event Roomba_IO_Handler IO_Handler;
public event Roomba_Text_IO_Handler Text_IO_Handler;

//public event Roomba_Motor_Action_Handler Before_Motor_Action_Send;
//public event Roomba_Motor_Action_Handler After_Motor_Action_Send;

#region Events

public Roomba()
{
}

public Roomba(SerialPort IO, double Sensor_Polling_Interval, string sLogPath)
{
    try
    {
        this.LogPath = sLogPath;

        Log.This("Initializing Sensors", c_sRoomba, this.LogSCICommands);

        this.Sensors = new Sensors(Sensor_Polling_Interval);
        this.IO_Buffer = new List<byte>();

        Log.This("Initialize Complete", c_sRoomba, this.LogSCICommands);
    }
    catch (Exception ex)
    {
        throw new RoombaException("Error during Roomba Initialize: " + ex.Message);
    }
}

#endregion Events

#region Event Handlers

public void IO_DataReceived(object sender, System.IO.Ports.SerialDataReceivedEventArgs e)
{
    Log.This(c_sIO_DataRecieved, c_sRoomba, this.LogIO);

    if (!this.Do_Not_Parse_RCV) { this.Parse_Sensor_Data(); };
}

#endregion Event Handlers
    public void IO_PinChanged(object sender, System.IO.Ports.SerialPinChangedEventArgs e)
    {
        Log.This(c_sIO_PinChanged, c_sRoomba, this.LogIO);
        this.Sensors.IsCurrent = false;
    }

    #endregion

    #region Properties

    private SerialPort p_spIO = null;

    /// <summary>
    ///
    /// </summary>
    public SerialPort IO
    {
        get
        {
            return p_spIO;
        }
        set
        {
            p_spIO = value;

            if (this.IO != null)
            {
                this.IO.DataReceived -= new
                System.IO.Ports.SerialDataReceivedEventHandler(this.IO_DataReceived);
                this.IO.PinChanged -= new SerialPinChangedEventHandler(this.IO_PinChanged);
                Log.This("Serial Port Set: " + p_spIO.PortName, c_sRoomba, this.LogIO);
                this.IO.DataReceived += new
                System.IO.Ports.SerialDataReceivedEventHandler(this.IO_DataReceived);
                this.IO.PinChanged += new SerialPinChangedEventHandler(this.IO_PinChanged);
            }
        }
    }

    private Velocity p_vVelocity;

    public Velocity Velocity
    {
        get
        {
            return (this.p_vVelocity);
        }
    }

    #endregion
private Radius p_rRadius;

public Radius Radius
{
    get
    {
        return (this.p_rRadius);
    }
}

private Sensors p_sSensors;

public Sensors Sensors
{
    get
    {
        return p_sSensors;
    }
    set
    {
        p_sSensors = value;
        Log.This("Sensors Set: ", c_sRoomba, this.LogSCICommands);
    }
}

private Macro p_mMacro = new Macro();

public Macro Macro
{
    get
    {
        return p_mMacro;
    }
    set
    {
        p_mMacro = value;
    }
}

private byte p_bcBaud_Rate;

public byte Baud_Rate
{
    get

{ return (this.p_bcBaud_Rate);
}

private byte p_byMode;

public byte Mode 
{
  
  get
  {
    return p_byMode;
  }
  
  set
  {
    try
    {
      bool bSuccess = this.SetMode(value); //item logged here
      if (bSuccess) {p_byMode = value; ];
    }
    catch (Exception ex)
    {
      throw new RoombaException("An Error has occurred while setting mode: " +
      ex.Message);
    }
  }

  
}

private string p_sLogPath = ""

public string LogPath 
{

  get
  {
    return (this.p_sLogPath);
  }

  set
  {
    this.p_sLogPath = value;
  }

}

private string p_sErrorText = "";
public string ErrorText
{
    get
    {
        return (this.p_sErrorText);
    }
    set
    {
        this.p_sErrorText = value;
    }
}

public bool p_bDebugMode = true; //default ON

public bool DebugMode
{
    get
    {
        return (this.p_bDebugMode);
    }
    set
    {
        this.p_bDebugMode = value;
    }
}

public bool p_bAutomatic_Polling = false;

public bool Automatic_Polling
{
    get
    {
        return (this.p_bAutomatic_Polling);
    }
    set
    {
        this.p_bAutomatic_Polling = value;
    }
}

public bool p_bLogIO = false; //default OFF

public bool LogIO
{
    get
```csharp
public bool p_bLogPacketData = false; //default OFF

public bool LogPacketData
{
    get
    {
        return (this.p_bLogPacketData);
    }
    set
    {
        this.p_bLogPacketData = value;
    }
}

public bool p_bDo_Not_Parse_RCV = false;

public bool Do_Not_Parse_RCV
{
    get
    {
        return (this.p_bDo_Not_Parse_RCV);
    }
    set
    {
        this.p_bDo_Not_Parse_RCV = value;
    }
}

private bool p_bLogSCICommands;
public bool LogSCICommands
{
    get
    {
        return (this.p_bLogSCICommands);
    }
    set
    {
        this.p_bLogSCICommands = value;
    }
}
```
private List<byte> p_bIO_Buffer;
public List<byte> IO_Buffer
{
    get
    {
        return (this.p_bIO_Buffer);
    }
    set
    {
        this.p_bIO_Buffer = value;
    }
}

private bool p_bNew_State;
public bool New_State
{
    get
    {
        return (this.p_bNew_State);
    }
    set
    {
        this.p_bNew_State = value;
    }
}

private byte p_byCurrent_Mode;
public byte Current_Mode
{
    get
    {
        return (this.p_byCurrent_Mode);
    }
    set
    {
        this.p_byCurrent_Mode = value;
    }
}

#region Functions

#region IO

public List<string> GetPorts()
{

List<string> lAvailablePorts = new List<string>();
string sPortsAvailable = "";

try
{
    lAvailablePorts.AddRange(SerialPort.GetPortNames());

    foreach (string sPort in lAvailablePorts)
    {
        sPortsAvailable += " " + sPort;
    }
}
catch
{

}
return lAvailablePorts;

#endregion

public bool Wake()
{
    this.IO.RtsEnable = true;
    for (int i = 0; i < 600; i++)
    {
    }
    this.IO.RtsEnable = false;
    this.Macro.SetAction("WAKE");
    return true;
}

public bool Spin(Radius rAngle)
{
    this.ErrorText = "";
    bool bSuccess = false;
    byte[] byRadius = BitConverter.GetBytes(rAngle.ToInt);
    byte[] b = new byte[5];
b[0] = OpCode.Drive;
b[1] = byRadius[0];
b[2] = byRadius[1];
b[3] = 0;
b[4] = 0;

string sDebugSend = "[" + b[0].ToString() + "]" + b[1].ToString() + "]" + b[2].ToString() + "]" + b[3].ToString() + "]";

try
{
    this.IO.RtsEnable = false; // otherwise, roomba will ignore you
    this.IO.Write(b, 0, b.Length);
    this.p_rRadius = rAngle;

    bSuccess = true;
    Log.This("Spin Action Executed: ", c_sRoomba, this.LogSCICommands);
    this.Macro.SetAction("SPIN " + rAngle as string);
}
catch (Exception ex)
{
    Log.This("Spin Fail: " + ex.Message, c_sRoomba, this.LogSCICommands);
}

this.Macro.SetAction("SPIN " + rAngle);

return bSuccess;
}
public bool SpinLeft()
{
    return this.Spin(-1);
}

public bool SpinRight()
{
    return this.Spin(-2);
}

public bool SetMode(byte bySCI_Mode)
{
    this.ErrorText = "";

    bool bSuccess = false;

    try
    {
        this.IO.RtsEnable = false;

        switch (bySCI_Mode)
        {
        case SCI_Mode.Off:
bSuccess = this.Execute(OpCode.Power);
this.p_byMode = SCI_Mode.Off;
Log.This("Mode Set to Off", c_sRoomba, this.LogSCICommands);
this.Macro.SetAction("SETMODE\tOff");
this.Current_Mode = SCI_Mode.Off;
break;

case SCI_Mode.Passive:
bSuccess = this.Execute(OpCode.Start);
this.p_byMode = SCI_Mode.Passive;
Log.This("Mode Set to Passive", c_sRoomba, this.LogSCICommands);
this.Current_Mode = SCI_Mode.Passive;
System.Threading.Thread.Sleep(20);

bSuccess &= this.Execute(OpCode.Enable_User_Control);
Log.This("Enable User Control Set", c_sRoomba, this.LogSCICommands);
this.Macro.SetAction("SETMODE\tPassive");
break;

case SCI_Mode.Safe:
bSuccess = this.Execute(OpCode.Safe_Mode);
this.p_byMode = SCI_Mode.Safe;
Log.This("Mode Set to Safe", c_sRoomba, this.LogSCICommands);
this.Macro.SetAction("SETMODE\tSafe");
this.Current_Mode = SCI_Mode.Safe;
break;

case SCI_Mode.Full:
bSuccess = this.Execute(OpCode.Full_Mode);
this.p_byMode = SCI_Mode.Full;
Log.This("Mode Set to Full", c_sRoomba, this.LogSCICommands);
this.Macro.SetAction("SETMODE\tFull");
this.Current_Mode = SCI_Mode.Full;
break;
}

if (bSuccess) { this.p_byMode = bySCI_Mode; }

System.Threading.Thread.Sleep(20);

}

catch (Exception ex)
{
  Log.This("Set Mode Fail: " + ex.Message, c_sRoomba, this.LogSCICommands);
}

return bSuccess;

}

public void EnforceSafeMode(Sensors rsCurrentSensorPoll)
if (rsCurrentSensorPoll.Packet.WheelDrop.Left ||
    rsCurrentSensorPoll.Packet.WheelDrop.Caster ||
    rsCurrentSensorPoll.Packet.WheelDrop.Right ||
    rsCurrentSensorPoll.Packet.Cliff.Left ||
    rsCurrentSensorPoll.Packet.Cliff.FrontLeft ||
    rsCurrentSensorPoll.Packet.Cliff.FrontRight ||
{
    this.SetMode(SCI_Mode.Passive);
}

#region Actions

public bool Change_Baud_Rate(byte bps)
{
    this.ErrorText = "";

    bool bSuccess = false;

    try
    {
        Log.This("Baud Rate Change: " + bps.ToString(), c_sRoomba, this.LogSCICommands);
        this.Execute(OpCode.Baud, bps);
        p_bcBaud_Rate = bps;
        this.Macro.SetAction("BAUD RATE " + bps.ToString());

        bSuccess = true;
    }
    catch (Exception ex)
    {
        Log.This("Baud Rate Change Fail: " + ex.Message, c_sRoomba, this.LogSCICommands);
    }

    return bSuccess;
}

public bool Motor_Action(byte bSend_Motor_Byte)
{
    this.ErrorText = "";

    bool bSuccess = false;

    try
    {
byte bSend = bSend_Motor_Byte;  //bExistingSettings +
this.IO.RtsEnable = false; //otherwise, roomba will ignore you
Log.This("Motor Action Requested: ", c_sRoomba, this.LogSCICommands);
bSuccess = this.Execute(OpCode.Motors, bSend);
this.SendMotorBytesToMacro(bSend_Motor_Byte);
}
catch (Exception ex)
{
    Log.This("Motor Action Fail: "+ ex.Message, c_sRoomba, this.LogSCICommands);
}
return bSuccess;
}

public void Drive(byte[] driveAction)
{


    try
    {
        this.IO.RtsEnable = false;
        this.IO.Write(driveAction, 0, driveAction.Length);
        this.Macro.SetAction("DRIVE\t" + sDebugSend);
        Log.This("Drive Action Success "+ sDebugSend, c_sRoomba, this.LogSCICommands);
    }
    catch (Exception ex)
    {
        Log.This("Drive Action Fail: "+ ex.Message + " Debug: "+ sDebugSend, c_sRoomba, this.LogSCICommands);
    }
}

public bool Drive(Velocity vSpeed, Radius rAngle, byte bySCI_Mode)
{
    bool bDriveSuccess;
    this.Mode = bySCI_Mode;
    bDriveSuccess = this.Drive(vSpeed, rAngle);
    return bDriveSuccess;
}

public bool Drive(Velocity vSpeed, Radius rAngle)
{
this.ErrorText = "";

bool bSafe = this.Mode == SCI_Mode.Safe;
bool bFull = this.Mode == SCI_Mode.Full;
bool bError = (!bSafe) & (!bFull);

//Push back at the user
if (bError)
{
}

//Sample from the SCI Spec:
//to drive in reverse at a velocity of -200 mm/s while turning at a radius of 500mm, you
would send the serial byte sequence
//[137] [255] [56] [1] [244]
//[137] [Velocity High Byte] [Velocity Low Byte] [Radius High Byte] [Radius Low Byte]

//divide up vSpeed & rAngle into 2 bytes ea
int num = rAngle.ToInt;
byte byAngleHi = (byte)(num >> 8);
byte byAngleLo = (byte)(num & 255);

num = vSpeed.ToInt;
byte bySpeedHi = (byte)(num >> 8);
byte bySpeedLo = (byte)(num & 255);

bool bSuccess = false;

List<byte> lSend = new List<byte>();
lSend.Add(OpCode.Drive);
lSend.Add(bySpeedHi);
lSend.Add(bySpeedLo);
lSend.Add(byAngleHi);
lSend.Add(byAngleLo);

string sDebugSend = "[" + lSend[0].ToString() + "]" + lSend[1].ToString() + "]" + lSend[2].ToString() + "]" + lSend[3].ToString() + "]" + lSend[4].ToString() + "]";

try
{
  Log.This("Drive Action: " + sDebugSend, e_sRoomba, this.LogSCICommands);

  this.IO.RtsEnable = false;
  this.IO.Write(lSend.ToArray(), 0, lSend.Count);
  this.Macro.SetAction("DRIVE\t" + sDebugSend);

  this.p_vVelocity = vSpeed;
  this.p_rRadius = rAngle;

  bSuccess = true;
Log.This("Drive Action Success " + bSuccess.ToString() + " Velocity: " + this.p_vVelocity.ToString() + " Radius: " + this.p_rRadius.ToString(), c_sRoomba, this.LogSCICommands);

} catch (Exception ex)
{
    Log.This("Drive Action Fail: " + ex.Message, c_sRoomba, this.LogSCICommands);
}

return bSuccess;

}

public bool SetLED(byte bSetting)
{
    this.ErrorText = ""
;

    bool bSuccess = false;

    try
    {

        byte[] b = new byte[4];
        b[0] = OpCode.LEDs;
        b[1] = bSetting;
        b[2] = 0;
        b[3] = 0;

        this.IO.RtsEnable = false; //otherwise, roomba will ignore you
        this.IO.Write(b, 0, b.Length);

        bSuccess = true;
        Log.This("Set LED Success: ", c_sRoomba, this.LogSCICommands);
        this.Macro.SetAction("SET_LED " + b.ToString());
    }
    catch (Exception ex)
    {
        Log.This("Set LED Fail: " + ex.Message, c_sRoomba, this.LogSCICommands);
    }

    return bSuccess;

}

public bool SetLEDs(byte bLED_Bits, byte bPWR_Color, byte bPWR_Intensity)
{
    this.ErrorText = ""
;

    bool bSuccess = false;

    List<byte> b = new List<byte>();


try
{
    b.Add(OpCodes.LEDs);
    b.Add(bLED_Bits);
    b.Add(bPWR_Color);
    b.Add(bPWR_Intensity);

    //write something here to enforce the values
    this.IO.RtsEnable = false; //otherwise, roomba will ignore you
    this.IO.Write(b.ToArray(), 0, b.Count);

    bSuccess = true;
    Log.This("Set LEDs Success", c_sRoomba, this.LogSCICommands);
    this.Macro.SetAction("SET_LED " + b.ToString());
}
catch (Exception ex)
{
    Log.This("Set LEDs Fail: " + ex.Message, c_sRoomba, this.LogSCICommands);
}
return bSuccess;

#endregion

#endregion

#region Supporting Functions

public bool Parse_Sensor_Data()
{
    bool bSuccess = false;
    bool bState_Changed = false;

    try
    {
        byte[] byString;

        this.Get_Packet(out byString); //out bylRecievedSensorData,

        this.LogSensorData(this.IO_Buffer); //bylRecievedSensorData
        this.Set_Packet(this.IO_Buffer, out bSuccess, out bState_Changed, byString);//
    }
    catch (Exception ex)
    {
        Log.This("Parse Sensor Data Fail: " + ex.Message, c_sRoomba, this.LogSCICommands);
    }
    return bSuccess;
}

#endregion
return bSuccess;

private void LogSensorData(List<byte> bylRecievedSensorData)
{
    string sDebug = null;

    foreach (byte byCurrent in bylRecievedSensorData)
    {
        sDebug += c_cOpen + byCurrent.ToString() + c_cClose;
    }

    Log.This(c_sPacketData + " " + sDebug, c_sRoomba, this.LogPacketData);
}

private void Get_Packet(out byte[] byString) //out List<byte> bylRecievedSensorData,
{
    int iBytesToRead = this.IO.BytesToRead;
    int iBytesLeft = iBytesToRead;

    if (this.LogIO)
    {
        Log.This(c_sBytesToRead + iBytesToRead.ToString(), c_sRoomba, this.LogIO);
    }

    byte byCurrent;

    int iCount = 0;

    this.Sensors.Previous_Bytes = this.Sensors.Raw_Bytes = new List<byte>();

    this.Sensors.Raw_Bytes = new List<byte>();

    while (iBytesLeft > 0)
    {
        byCurrent = (byte)this.IO.ReadByte();
        iCount++;
        this.IO_Buffer.Add(byCurrent);
        iBytesLeft = this.IO.BytesToRead;
    }

    if (this.LogIO)
    {
Log.This(this.IO_Buffer.Count.ToString() + c_sBytesRetrieved + " bytes left: " + iBytesLeft.ToString() + " iCount: " + iCount.ToString(), c_sRoomba, this.LogIO);//bylRecievedSensorData

byString = this.IO_Buffer.ToArray(); //bylRecievedSensorData

private void Set_Packet(List<byte> bylRecievedSensorData, out bool bSuccess, out bool bState_Changed, byte[] byString)//,
{
    bSuccess = false;

    bState_Changed = !EqualByteArrays(this.Sensors.Previous_Bytes, bylRecievedSensorData); //
    this.New_State = bState_Changed;

    if (bState_Changed)
    {
        this.Sensors.Raw_Bytes = bylRecievedSensorData;
        if (bylRecievedSensorData.Count >= Packet.Full) //
        {
            Log.This(c_sParsingData, c_sRoomba, this.LogIO);
            bSuccess = this.Sensors.Parse();
            this.EnforceSafeMode(this.Sensors);

            this.IO_Buffer = new List<byte>();
            Log.This(" - Resetting Buffer - " + c_sBytesRetrieved, c_sRoomba, this.LogIO);
        }
        else
        {
            Log.This(c_sIgnored, c_sRoomba, this.LogIO);
        }
    }
    else
    {
        Log.This(c_sIgnored, c_sRoomba, this.LogIO);
    }

    if (IO_Handler != null)
    {
        Log.This("Broadcast", c_sRoomba, this.LogIO);
        this.IO_Handler(byString);
        Log.This("After Broadcast", c_sRoomba, this.LogIO);
    }
    else
    {
        if (bylRecievedSensorData.Count >= Packet.Full) //
        {
            this.Sensors.LastUpdated = DateTime.Now;
        }

        Log.This("Roomba State unchanged.", c_sRoomba, this.LogIO);
    }
}
public static bool EqualByteArrays(List<byte> data1, List<byte> data2)
{
    // If both are null, they're equal
    if (data1 == null && data2 == null) { return true; }
    
    // If either but not both are null, they're not equal
    if (data1 == null || data2 == null) { return false; }
    
    if (data1.Count != data2.Count) { return false; }
    for (int i = 0; i < data1.Count; i++)
    {
        if (data1[i] != data2[i]) { return false; }
    }
    return true;
}

public bool ReadExisting()
{
    int iBytesToRead = this.IO.BytesToRead;
    int iBytesLeft = iBytesToRead;
    string sRoombaText = "";
    sRoombaText = this.IO.ReadExisting();
    Text_IO_Handler(sRoombaText); //For Roomba's Text Responses
    iBytesLeft = this.IO.BytesToRead;

    return true;
}

public bool Execute(byte bOpCode, byte bSendByte)
{
    this.ErrorText = "";
    bool bSuccess = false;
    byte[] b = new byte[2];
    b[0] = bOpCode;
    b[1] = bSendByte;
    string sDebugSend = c_cOpen + b[0].ToString() + "]" + b[1].ToString() + c_cClose;
    try
    {
        Log.This(c_sExecuteAction + sDebugSend, c_sRoomba, this.LogIO);
        this.IO.Write(b, 0, b.Length);
bSuccess = true; //Command sent without error

} catch (Exception ex)
{
    Log.This(c_sExecuteActionFail + ex.Message, c_sRoomba, this.LogIO);
}

return bSuccess;

public bool Execute(byte bOpCode)
{
    this.ErrorText = "";

    bool bSuccess = false;
    byte[] b = new byte[1];
b[0] = bOpCode;

    string sDebugSend = c_cOpen + b[0].ToString() + c_cClose;

    try
    {
        Log.This(c_sExecuteAction + sDebugSend, c_sRoomba, this.LogIO);
        this.IO.Write(b, 0, b.Length);

        bSuccess = true;
    }
    catch (Exception ex)
    {
        Log.This("Execute Action Fail: " + ex.Message, c_sRoomba, this.LogIO);
    }

    return bSuccess;
}

public bool Execute(List<byte> sendBytes) //Untested as of 3.3.07
{
    this.ErrorText = "";

    bool bSuccess = false;

    byte[] b = sendBytes.ToArray();

    try
    {
        this.IO.RtsEnable = false; //otherwise, roomba will ignore you
        this.IO.Write(b, 0, b.Length);

        bSuccess = true;
        Log.This(c_sExecuteAction, c_sRoomba, this.LogSCICommands);
catch (Exception ex)
{
    Log.This(c_sExecuteAction + " fail ~ " + ex.Message, c_sRoomba, this.LogSCICommands);
}

return bSuccess;

protected void SendMotorBytesToMacro(byte bSend_Motor_Byte)
{
    string byteText = "";

    List<byte> motorBytes = new List<byte>();
    motorBytes.Add(bSend_Motor_Byte);
    this.Macro.SetAction("MOTORS\t" + bSend_Motor_Byte.ToString());
}

namespace RoombaSCI
{
    /// <summary>
    /// The commands in which to control Roomba. Each command consists of a one byte code. Some
    /// commands must also be followed by data bytes<br/>
    /// Roomba will not respond to any commands while asleep.
    /// </summary>

    public static class OpCode
    {
        /// Starts the SCI. The Start command must be sent before any other SCI Commands. <i>This
        /// command puts Roomba in <b>passive</b> mode.</i><br/>
        /// Data Bytes: 0<br/>
        /// OpCode = 128<br/>
        public const byte Start = 128;
    }
}
/// <summary>
/// Sets the baud rate (in bps) at which SCI commands and data are sent. The default baud rate at power up is 57600 bps.<br/>
/// Once the baud rate is changed, it will persist until Roomba is power cycled by removing the battery or the battery is dead.<br/>
/// You must wait 100ms after sending this command before sending additional commands at the new baud rate.<br/>
/// <i>Roomba must be in &lt;b&gt;passive&lt;/b&gt;, &lt;b&gt;safe&lt;/b&gt;, or &lt;b&gt;full&lt;/b&gt; mode to accept this command</i><br/>
/// Data Bytes: 1 (0 - 11)<br/>
/// OpCode = 129<br/>
/// </summary>
public const byte Baud = 129;

/// <summary>
/// Enables User Control of Roomba. This command must be sent after the &lt;i&gt;start&lt;/i&gt; command and before any control commands are sent.<br/>
/// &lt;i&gt;Roomba must be in &lt;b&gt;passive&lt;/b&gt; mode to accept this command&lt;/i&gt;&lt;br&gt;&lt;/br&gt;
/// Data Bytes: 0<br/>
/// OpCode = 130<br/>
/// </summary>
public const byte Enable_User_Control = 130;

/// <summary>
/// Enables a restricted mode of operation for Roomba. Roomba will react to all sensor input (such as cliff and bumper sensors), as appropriate.<br/>
/// &lt;i&gt;This command puts Roomba into &lt;b&gt;safe&lt;/b&gt; mode&lt;/i&gt;&lt;br&gt;&lt;/br&gt;
/// Roomba must be in &lt;b&gt;full&lt;/b&gt; mode to accept this command&lt;br&gt;&lt;/br&gt;
/// Data Bytes: 0&lt;br&gt;&lt;/br&gt;
/// OpCode = 131&lt;br&gt;&lt;/br&gt;
/// </summary>
public const byte Safe_Mode = 131;

/// <summary>
/// Enables unrestricted control of Roomba.<br/>
/// &lt;i&gt;This command puts Roomba into &lt;b&gt;full&lt;/b&gt; mode&lt;/i&gt;&lt;br&gt;&lt;/br&gt;
/// Roomba must be in &lt;b&gt;safe&lt;/b&gt; mode to accept this command&lt;br&gt;&lt;/br&gt;
/// Data Bytes: 0&lt;br&gt;&lt;/br&gt;
/// OpCode = 132&lt;br&gt;&lt;/br&gt;
/// </summary>
public const byte Full_Mode = 132;

/// <summary>
/// Puts Roomba to sleep, the same as the normal "power" button press. To wake, use the &lt;b&gt;wake&lt;/b&gt; function, or set Roomba.IO.RtsEnable Off, then on for &gt;500 MS<br/>
/// &lt;i&gt;Roomba must be in &lt;b&gt;passive&lt;/b&gt; mode&lt;/i&gt;&lt;br&gt;&lt;/br&gt;
/// Data Bytes: 0&lt;br&gt;&lt;/br&gt;
/// </summary>
public const byte Power = 133;

public const byte Spot = 134;

public const byte Clean = 135;

public const byte Max = 136;

/// Controls Roomba's drive wheels. This command takes 4 data bytes.<br>
/// These data bytes are interpreted as two 16 bit values using two's-complement.<br>
/// The first two bytes specify the average velocity of the drive wheels in millimeters per second (mm/s) <br>
/// The next 2 bytes specify the radius, in millimeters, in which Roomba should turn. The longer<br>
/// radii make Roomba drive straighter; shorter radii make Roomba turn more.<br>
/// A drive command with a positive velocity and a positive radius will make Roomba drive forward while<br>
/// turning toward the left.<br>
/// A negative radius will make Roomba turn toward the right.<br>
/// Special cases for the radius make Roomba turn in place or drive straight.<br>
/// Note: The robot system and its environment impose restrictions that may prevent the robot from<br>
/// accurately carrying out some drive commands. For example, it may not be possible to drive in a large arc<br>
/// with a large radius of curvature.<br>
/// Data Bytes: 4<br>
/// Data Bytes 1 and 2: Velocity (-500 - 500 mm/s)<br>
/// Data Bytes 3 and 4: Radius (-2000 - 2000 mm)
/// Special cases for Radius: 32768 = straight, -1 = turn in place clockwise, 1 = Turn in place counterclockwise
/// C# usage example:<br/>
/// To drive in reverse at a velocity of -200mm/s while turning at a radius of 500mm:<br/>
/// myRoombaObj.Drive(-200, 500);<br/>
/// <br/>
/// OpCode = 137<br/>
/// <summary><br/>
/// public const byte Drive = 137;<br/>
/// </summary><br/>
/// Controls Roomba's cleaning motors.<br/>
/// This OpCode is used in conjunction with the RoombaSCI.Motor class<br/>
/// <br/>
/// OpCode = 138<br/>
/// <summary><br/>
/// public const byte Motors = 138;<br/>
/// </summary><br/>
/// Controls Roomba's LEDs. This OpCode is used in conjunction with Roomba's LED class<br/>
/// <br/>
/// OpCode = 139<br/>
/// <summary><br/>
/// public const byte LEDs = 139;<br/>
/// </summary><br/>
/// Specifies a song to be played later. This opcode currently has minimal support by RoombaSCI (as of 5/24/06)<br/>
/// OpCode = 140<br/>
/// Stay Tuned!<br/>
/// <summary><br/>
/// public const byte Song = 140;<br/>
/// </summary><br/>
/// Plays one of 16 Songs. This opcode currently has minimal support by RoombaSCI (as of 5/24/06)<br/>
/// OpCode = 141<br/>
/// Stay Tuned!<br/>
public const byte Play = 141;

/// <summary>
/// Requests a packet of Sensor Data Bytes. The user can select one of 4 different Sensor packets.
/// This Command is used in conjunction with the <b>Sensors</b> class
/// This command does not change Roomba's mode<br/>
/// Roomba must be in <b>passive</b>, <b>safe</b> or <b>full</b> mode to accept this command<br/>
/// OpCode = 142<br/>
/// </summary>

public const byte Sensors = 142;

/// <summary>
/// Turns on Roomba's force-seeking dock mode, which causes the robot to immediately attempt to
dock during its cleaning cycle.<br/>
/// if it encounters the docking beams from its home base. (note, however, that if the robot was not
active in a clean, spot, or max cycle<br/>
/// it will not attempt to execute the docking.) Normally, the robot attempts to dock only if the
cleaning cycle has completed or the battery<br/>
/// is nearing depletion. The command can be sent anytime, but the mode will be cancelled if the
robot turns off, begins charging, or is<br/>
/// commanded to <b>safe</b> or <b>full</b> mode.<br/>
/// OpCode = 143<br/>
/// </summary>

public const byte Force_Seeking_Dock = 143;

public static class Sensor_Code
{
    public const byte Bump_Right = 1;
    public const byte Bump_Left = 2;
    public const byte Bump_Both = 3;
    public const byte WheelDrop_Right = 4;
    public const byte WheelDrop_Left = 8;
    public const byte WheelDrop_Caster = 16;
    public const byte OverCurrent_Side_Brush = 1;
    public const byte OverCurrent_Vacuum = 2;
    public const byte OverCurrent_Main_Brush = 4;
    public const byte OverCurrent_Drive_Right = 8;
    public const byte OverCurrent_Drive_Left = 16;
    public const byte Buttons_Max = 1;
    public const byte Buttons_Clean = 2;
    public const byte Buttons_Spot = 4;
    public const byte Buttons_Power = 8;
}

/// <summary>
/// Property bag that encapsulates the sensor packet sent by Roomba when polled.
/// </summary>
public class Sensor_Packet
{

    #region "Packet Subset 1 (10 bytes)"

    //Range: 0-31
    private Bump_Sensor p_bpBump = new Bump_Sensor(); // Bump + WheelDrops = 1 byte, unsigned

    /// <summary>
    /// Roomba SCI sensor packet subset 1<br>
    /// 1 byte, unsigned<br>
    /// Range  0-1<br>
    /// <i>Property bag</i><br>
    /// When any of this property's members are <b>true</b>, denotes that a bump has occurred at the appropriate sensor on Roomba's front bumper
    /// </summary>
    public Bump_Sensor Bump
    {
        get
        {
            return p_bpBump;
        }
        set
        {
            p_bpBump = value;
        }
    }

    private WheelDrop p_wWheelDrop = new WheelDrop();
    public WheelDrop WheelDrop
    {
        get
        {
            return p_wWheelDrop;
        }
        set
        {
            p_wWheelDrop = value;
        }
    }

    //Range: 0-1
    private bool p_bWall; //1 byte, unsigned

    public bool Wall
    {
        get
        {
            return p_bWall;
        }
    }
}
```csharp
set
{
    p_bWall = value;
}

//Range: 0-1 on each Cliff Sensor
private Cliff_Sensor p_csCliff = new Cliff_Sensor(); //1 byte, unsigned

public Cliff_Sensor Cliff
{
    get
    {
        return p_csCliff;
    }
    set
    {
        p_csCliff = value;
    }
}

//Range: 0-1
private bool p_bVirtual_Wall; //1 byte, unsigned

public bool Virtual_Wall
{
    get
    {
        return p_bVirtual_Wall;
    }
    set
    {
        p_bVirtual_Wall = value;
    }
}

//Range: 0-31
private OverCurrent p_oOverCurrent = new OverCurrent(); //Range: 0-31

public OverCurrent OverCurrent
{
    get
    {
        return p_oOverCurrent;
    }
    set
    {
        p_oOverCurrent = value;
    }
}
```
private Dirt_Detector p_ddDirt_Detector = new Dirt_Detector(); //1 byte, unsigned

public Dirt_Detector Dirt_Detector
{
    get
    {
        return p_ddDirt_Detector;
    }
    set
    {
        p_ddDirt_Detector = value;
    }
}

private ushort p_uRemote_Control_Command; //1 byte, unsigned

public ushort Remote_Control_Command
{
    get
    {
        return p_uRemote_Control_Command;
    }
    set
    {
        p_uRemote_Control_Command = value;
    }
}

private Buttons p_buButtons = new Buttons();  //1 byte, unsigned

public Buttons Buttons
{
    get
    {
        return p_buButtons;
    }
    set
    {
        p_buButtons = value;
    }
}

private short p_uiDistance; //2 bytes, signed

public short Distance
```csharp
{  
    get  
    {  
        return p_uiDistance;  
    }  
    set  
    {  
        p_uiDistance = value;  
    }  
}  

//Range -32768 - 32767  
private short p_uiAngle;  //2 bytes, signed  
public short Angle  
{  
    get  
    {  
        return p_uiAngle;  
    }  
    set  
    {  
        p_uiAngle = value;  
    }  
}  

#endregion  
#region "Packet Subset 3 (10 Bytes)"

//Range 0-5  
private byte p_byCharging_State;  //1 byte, unsigned  
public byte Charging_State  
{  
    get  
    {  
        return p_byCharging_State;  
    }  
    set  
    {  
        p_byCharging_State = value;  
    }  
}  

//Range 0 - 65535  
private ushort p_usVoltage;  //2 bytes, unsigned  
public ushort Voltage  
{  
    get  
    {  
        return p_usVoltage;  
    }  
    set  
    {  
        p_usVoltage = value;  
    }  
}  
```
private short p_shCurrent; //2 bytes, signed
public short Current
{
    get
    {
        return p_shCurrent;
    }
    set
    {
        p_shCurrent = value;
    }
}

private byte p_byTemperature; //1 byte, signed
public byte Temperature
{
    get
    {
        return p_byTemperature;
    }
    set
    {
        p_byTemperature = value;
    }
}

private UInt16 p_uiCharge; //2 bytes, unsigned
public UInt16 Charge
{
    get
    {
        return p_uiCharge;
    }
    set
    {
        p_uiCharge = value;
    }
}

private UInt16 p_uiCapacity; //2 bytes, unsigned
public UInt16 Capacity
{
get
{
    return p_uiCapacity;
}
set
{
    p_uiCapacity = value;
}

#endregion

/// <summary>
/// These constants represent Roomba's 4 Operating Modes.
/// </summary>
public static class SCI_Mode
{
    public const byte Off = 0;
    public const byte Passive = 1;
    public const byte Safe = 2;
    public const byte Full = 3;
}

public static class Motor
{
    public static class Vacuum
    {
        public const byte On = 2;
        public const byte Off = 0;
    }
    public static class Main_Brush
    {
        public const byte On = 4;
        public const byte Off = 0;
    }
    public static class Side_Brush
    {
        public const byte On = 1;
        public const byte Off = 0;
    }
}

public static class LED
{
    public static class Dirt_Detect
    {
        public const byte On = 1;
        public const byte Off = 0;
    }
public static class Max
{
    public const byte On = 2;
    public const byte Off = 0;
}

public static class Clean
{
    public const byte On = 4;
    public const byte Off = 0;
}

public static class Spot
{
    public const byte On = 8;
    public const byte Off = 0;
}

public static class Status
{
    public const byte Red = 16;
    public const byte Green = 32;
    public const byte BrightRed = 48;
    public const byte Off = 0;
}

public static class BaudCode
{
    public const byte x300 = 0;
    public const byte x600 = 1;
    public const byte x1200 = 2;
    public const byte x2400 = 3;
    public const byte x4800 = 4;
    public const byte x9600 = 5;
    public const byte x14400 = 6;
    public const byte x19200 = 7;
    public const byte x28800 = 8;
    public const byte x38400 = 9;
    public const byte x57600 = 10;
    public const byte x115200 = 11;
}

    /// <summary>
    /// This structure is used to read and set Roomba's velocity. This structure is designed to be used as a variable.
    /// This structure also serves to keep any assigned variables within the limits of the SCI spec
    /// The limits are: -500mm/s - 500 mm/s
    /// Velocity x = 250; //if the programmer sets this to a value > 500, then x will automatically set itself to 500
    /// Radius y = -400;
    /// this.CurrentRoomba.Drive(x, y);
/// <example>
public struct Velocity
{
    private readonly int m_iValue;

    public const int Maximum_Forward = 500;
    public const int Maximum_Reverse = -500;

    public static implicit operator Velocity(int iValue)
    {
        return new Velocity(iValue);
    }

    public Velocity(int iSpeed)
    {
        if (iSpeed > 500) { iSpeed = 500; }
        if (iSpeed < -500) { iSpeed = -500; }

        this.m_iValue = iSpeed;
    }

    public int ToInt
    {
        get
        {
            return this.m_iValue;
        }
    }
}

public struct Radius
{
    private readonly int m_iValue;
    public const int Straight = 32768;
    public const int Maximum_Right = -2000;
    public const int Maximum_Left = 2000;

    public static implicit operator Radius(int iValue)
    {
        return new Radius(iValue);
    }

    public Radius(int iAngle)
    {
        if ((iAngle > 2000) & (iAngle != 32768)) { iAngle = 2000; }
        if (iAngle < -2000) { iAngle = -2000; }

        this.m_iValue = iAngle;
    }
}
public int ToInt
{
    get
    {
        return this.m_iValue;
    }
}

public static class Packet
{
    public const byte Full = 26;
}

#region Property Objects

/// <summary>
/// Roomba SCI sensor packet subset 1<br/>
/// 1 byte, unsigned<br/>
/// Range 0-1<br/>
/// Property bag<br/>
/// When any of the class members are true, denotes that a cliff is present at the appropriate sensor on the bottom of Roomba's front bumper
/// </summary>
public class Cliff_Sensor
{
    /// <summary>
    /// When true, denotes that a cliff is present at the leftmost cliff sensor on the bottom of Roomba's front bumper
    /// </summary>
    public bool Left;

    /// <summary>
    /// When true, denotes that a cliff is present at the front left cliff sensor on the bottom of Roomba's front bumper
    /// </summary>
    public bool FrontLeft;

    /// <summary>
    /// When true, denotes that a cliff is present at the rightmost cliff sensor on the bottom of Roomba's front bumper
    /// </summary>
    public bool Right;

    /// <summary>
    /// When true, denotes that a cliff is present at the front right cliff sensor on the bottom of Roomba's front bumper
    /// </summary>
    public bool FrontRight;
}
/// <summary>
/// Roomba SCI sensor packet subset 1
/// 1 byte, unsigned
/// Range 0-1
/// Property bag
/// When any of the class members are true, denotes that a bump has occurred at the appropriate sensor on Roomba's front bumper
/// </summary>
public class Bump_Sensor
{
    /// <summary>
    /// When true, denotes that a bump has occurred at the left bump sensor on Roomba's front bumper
    /// </summary>
    public bool Left;

    /// <summary>
    /// When true, denotes that a bump has occurred at the right bump sensor on Roomba's front bumper
    /// </summary>
    public bool Right;

    /// <summary>
    /// When true, denotes that both bump sensors have been activated
    /// </summary>
    public bool Both;
}

/// <summary>
/// Roomba SCI sensor packet subset 1
/// 1 byte, unsigned
/// Range 0-1
/// Property bag
/// When any of the class members are true, denotes that one of Roomba's wheels has dropped, or a combination thereof
/// </summary>
public class WheelDrop
{
    /// <summary>
    /// When true, denotes that no wheel drops are occurring
    /// </summary>
    public bool None;

    /// <summary>
    /// When true, denotes that Roomba's left wheel has dropped
    /// </summary>
    public bool Left;

    /// <summary>
    /// When true, denotes that Roomba's right wheel has dropped
    /// </summary>
    public bool Right;

    /// <summary>
    /// When true, denotes that both wheels have dropped
    /// </summary>
    public bool Both;
}
/// When `<b>true</b>`, denotes that Roomba's `<b>right</b>` wheel has dropped
/// </summary>

        public bool Right;

/// When `<b>true</b>`, denotes that Roomba's `<b>center</b>` wheel/caster has dropped
/// </summary>
        public bool Caster;
    }

/// Roomba SCI sensor packet subset 1<br/>
/// 1 byte, unsigned (for each detector)<br/>
/// Range 0-255<br/>
/// <i>Property bag</i><br/>
/// For each member, a value of 0 indicates that no dirt is detected. Higher values indicate higher
/// levels of dirt detected. (255 Max)
/// </summary>

    public class Dirt_Detector
    {
        /// For the `<b>left</b>` detector, value of 0 indicates that no dirt is detected. Higher values indicate
        /// higher levels of dirt detected.
        /// </summary>
        public UInt16 Left;

        /// For the `<b>right</b>` detector, value of 0 indicates that no dirt is detected. Higher values
        /// indicate higher levels of dirt detected.
        /// </summary>
        public UInt16 Right;
    }

/// Roomba SCI sensor packet subset 2<br/>
/// 1 byte, unsigned<br/>
/// Range 0-31<br/>
/// <i>Property bag</i><br/>
/// When any of the class members are `<b>true</b>`, denotes an overcurrent state in one of Roomba's
/// 5 motors.
/// </summary>

    public class OverCurrent
    {
        /// When `<b>true</b>`, denotes an overcurrent state in Roomba's `<b>left wheel</b>` motor.
        /// </summary>
        public bool Left_Wheel;

        /// When `<b>true</b>`, denotes an overcurrent state in Roomba's `<b>right wheel</b>` motor.
        /// </summary>
public bool Right_Wheel;

/// <summary>
/// When <b>true</b>, denotes an overcurrent state in Roomba's <b>main brush</b> motor.
/// </summary>
public bool Main_Brush;

/// <summary>
/// When <b>true</b>, denotes an overcurrent state in Roomba's <b>side brush</b> motor.
/// </summary>
public bool Side_Brush;

/// <summary>
/// When <b>true</b>, denotes an overcurrent state in Roomba's <b>vacuum</b> motor.
/// </summary>
public bool Vacuum;

public class Buttons
{
    /// <summary>
    /// When <b>true</b>, denotes that Roomba's <b>power</b> button has been pressed.
    /// </summary>
    public bool Power;

    /// <summary>
    /// When <b>true</b>, denotes that Roomba's <b>spot</b> button has been pressed.
    /// </summary>
    public bool Spot;

    /// <summary>
    /// When <b>true</b>, denotes that Roomba's <b>clean</b> button has been pressed.
    /// </summary>
    public bool Clean;

    /// <summary>
    /// When <b>true</b>, denotes that Roomba's <b>max</b> button has been pressed.
    /// </summary>
    public bool Max;
}

/// <summary>
/// Roomba SCI sensor packet subset 3</summary>
/// 1 byte, unsigned
/// Range 1-15
/// <i>Property bag</i>
/// When any of the class members are <b>true</b>, denotes a that one of Roomba's buttons have been pressed, or a combination thereof
/// </summary>
1 byte, unsigned
Range 0-5
Property bag
When any of the class members are true, denotes that Roomba is in the specified charging state.

public class Charging_State
{
    public const byte Not_Charging = 0;

    public const byte Charging_Recovery = 1;
    public const byte Charging = 2;
    public const byte Trickle_Charging = 3;
    public const byte Waiting = 4;
    public const byte Charging_Error = 5;
    public const byte Indeterminate = 6;
}

public class Charge_State_Description
{
    public const string Not_Charging = "Not Charging";
    public const string Charging_Recovery = "Charging Recovery";
    public const string Charging = "Charging";
    public const string Trickle_Charging = "Trickle Charging";
    public const string Waiting = "Waiting";
    public const string Charging_Error = "Charging Error";
    public const string Indeterminate = "Indeterminate";

    public const string Plugged_In = "Plugged In";
}

#endregion
using System;
using System.Collections;
using System.Collections.Generic;

namespace RoombaSCI
{
    public class SensorsException : System.ApplicationException
    {
        public SensorsException(string message) : base(message)
        {
        }
    }

    /// <summary>
    /// This structure is intended to be a representation of Roomba's SCI Sensor Packets. This structure is set up as a group of property bags that are loaded every time Roomba's Sensors are polled by <b>Sensors.Parse()</b>
    /// </summary>
    public class Sensors
    {
        /// <summary>
        /// </summary>
        /// <param name="Sensor_Polling_Interval"></param>
        public Sensors(double Sensor_Polling_Interval)
        {
            try
            {
                this.PollingInterval = Sensor_Polling_Interval;
            }
            catch (Exception ex)
            {
                throw new SensorsException("An Error has occurred while initializing Sensors " + ex.Message);
            }
        }

        #region Constants

        protected string c_sAngle = "Angle";
        protected string c_sBumpBoth = "Bump.Both";
        protected string c_sBumpNone = "Bump.None";
        protected string c_sBumpRight = "Bump.Right";
        protected string c_sBumpLeft = "Bump.Left";
        protected string c_sClean = "Buttons.Clean";

        #endregion
    }
}
protected string c_sMax = "Buttons.Max";
protected string c_sPower = "Buttons.Power";
protected string c_sSpot = "Buttons.Spot";
protected string c_sCapacity = "Capacity";
protected string c_sCharge = "Charge";
protected string c_sChargingState = "Charging_State";
protected string c_sCliffFrontLeft = "Cliff.FrontLeft";
protected string c_sCliffFrontRight = "Cliff.FrontRight";
protected string c_sCliffLeft = "Cliff.Left";
protected string c_sCliffRight = "Cliff.Right";
protected string c_sCurrent = "Current";
protected string c_sDirtDetectorLeft = "Dirt_Detector.Left";
protected string c_sDirtDetectorRight = "Dirt_Detector.Right";
protected string c_sDistance = "Distance";
protected string c_sOverCurrentLeft = "OverCurrent.Left_Wheel";
protected string c_sOverCurrentRight = "OverCurrent.Right_Wheel";
protected string c_sOverCurrentMAIN = "OverCurrent.Main_Brush";
protected string c_sOverCurrentSide = "OverCurrent.Side_Brush";
protected string c_sOverCurrentVacuum = "OverCurrent.Vacuum";
protected string c_sRemote = "Remote_Control_Command";
protected string c_sTemperature = "Temperature";
protected string c_sVirtualWall = "Virtual_Wall";
protected string c_sVoltage = "Voltage";
protected string c_sWall = "Wall";
protected string c_sWheelDropCaster = "WheelDrop.Caster";
protected string c_sWheelDropLeft = "WheelDrop.Left";
protected string c_sWheelDropLeftAndCaster = "WheelDrop.LeftAndCaster";
protected string c_sWheelDropLeftAndRight = "WheelDrop.LeftAndRight";
protected string c_sWheelDropNone = "WheelDrop.None";
protected string c_sWheelDropRight = "WheelDrop.Right";
protected string c_sWheelDropRightAndCaster = "WheelDrop.RightAndCaster";

#endregion
#region Local Variables

/// <summary>
/// This is the value of the remote control default value. Since the packets can jump around a little bit in the polling stream, this is used to sync things back up.
/// </summary>
private const byte c_byPacket_Sync = 255;

private byte m_byCapacity1 = 10;
private byte m_byCapacity2 = 142;

#endregion
#region Properties

private double p_dPollingInterval;

/// <summary>
/// The interval (in milliseconds) that Roomba's Sensors will be polled
/// </summary>
public double PollingInterval
{
    get
    {
        return p_dPollingInterval;
    }
    set
    {
        p_dPollingInterval = value;
    }
}

private double p_dPolling;
public double Polling
{
    get
    {
        return p_dPolling;
    }
    set
    {
        bool bSuccess = this.Set_Polling_Interval(value);
        if (bSuccess) { p_dPolling = value; };
    }
}

public List<byte> p_lBytes;

/// <summary>
/// Generic List of of the <b>raw</b> bytes that are returned from a sensor poll
/// </summary>
public List<byte> Raw_Bytes
{
    get
    {
        return p_lBytes;
    }
    set
    {
        p_lBytes = value;
    }
}

public List<byte> p_lprBytes;
/// <summary>
/// Generic List of of the <b>raw</b> bytes that are returned from a sensor poll
/// </summary>
public List<byte> Previous_Bytes
{
    get
    {
        return p_lprBytes;
    }
    set
    {
        p_lprBytes = value;
    }
}

public List<byte> p_lSync_Bytes;

/// <summary>
/// Generic List of of the bytes that are returned from a sensor poll that are assumed to be Roomba's sensor packet.
/// </summary>
public List<byte> Sync_Bytes
{
    get
    {
        return p_lSync_Bytes;
    }
    set
    {
        p_lSync_Bytes = value;
    }
}

private bool p_bStopPolling;

/// <summary>
/// When <b>true</b>, then polling of Roomba's sensors will cease
/// </summary>
public bool StopPolling
{
    get
    {
        return p_bStopPolling;
    }
    set
    {
    }
private bool p_bLock;

/// <summary>
/// Experimental property. If Lock = true, then Sensor polling is ignored. Lock will keep the
/// Sensors object from being repopulated
/// </summary>
public bool Lock
{
    get
    {
        return p_bLock;
    }
    set
    {
        p_bLock = value;
    }
}

private DateTime p_dtLastUpdated;

/// <summary>
/// </summary>
public DateTime LastUpdated
{
    get
    {
        return p_dtLastUpdated;
    }
    set
    {
        p_dtLastUpdated = value;
    }
}

private TimeSpan p_tsUpdateResponse;
public TimeSpan UpdateResponse
{
    get
    {
        return p_tsUpdateResponse;
    }
    set
    {
    }
}
private TimeSpan p_tsParseTime;

public TimeSpan ParseTime
{
    get
    {
        return p_tsParseTime;
    }
    set
    {
        p_tsParseTime = value;
    }
}

private bool p_bIsCurrent = false;

/// <summary>
/// Experimental property. Intended to reveal that the data sitting in the Sensors object is not old data. This property is set to True upon evaluation that this sensors object is being populated on a routine basis.
/// </summary>
public bool IsCurrent
{
    get
    {
        return p_bIsCurrent;
    }
    set
    {
        p_bIsCurrent = value;
    }
}

private int p_iIsCurrent_Threshold = 300; //default value

/// <summary>
/// .
/// .
/// </summary>
public int IsCurrent_Threshold
{
}
```csharp
private int p_iDefaultDataPoints = 200;
public int DefaultDataPoints
{
    get
    {
        return p_iDefaultDataPoints;
    }
    set
    {
        p_iDefaultDataPoints = value;
    }
}

private long p_iSensor_Parse = 200;
public long Sensor_Parse
{
    get
    {
        return p_iSensor_Parse;
    }
    set
    {
        p_iSensor_Parse = value;
    }
}

private bool p_bIsText = false;

/// <summary>
/// Experimental property. Set to True when it is determined that Roomba is returning Text (and not sensor packets) in response to sensor polling
/// <summary>
public bool IsText
{
```
get
{
    return p_bIsText;
}
set
{
    p_bIsText = value;
}
}

private bool p_sSensorText = false;

/// <summary>
/// Experimental property.
/// </summary>
public bool SensorText
{
    get
    {
        return p_sSensorText;
    }
    set
    {
        p_sSensorText = value;
    }
}

private Sensor_Packet p_pPacket = new Sensor_Packet();

/// <summary>
/// Property bag<br></br>
/// encopsulates the Sensor Packets Sent from Roomba in response to a sensor poll
/// </summary>
public Sensor_Packet Packet
{
    get
    {
        return p_pPacket;
    }
    set
    {
        p_pPacket = value;
    }
}
private bool p_bAutoExport;

/// <summary>
/// This is used by the Macro object, or anyone else wanting to refer to the Sensors structure this way.
/// </summary>
public bool AutoExport
{
    get
    {
        return p_bAutoExport;
    }
    set
    {
        p_bAutoExport = value;
    }
}

private Hashtable p_hHashtable;

/// <summary>
/// This is used by the Macro object, or anyone else wanting to refer to the Sensors structure this way.
/// </summary>
public Hashtable Hashtable
{
    get
    {
        return p_hHashtable;
    }
    set
    {
        p_hHashtable = value;
    }
}

#region Functions

/// <summary>
/// This is the function that divvies up Roomba's Raw bytes into this sensor structure
/// </summary>
/// <returns></returns>
public bool Parse()
{
    bool bSuccess = false;
    bool bBytesSyncd = false;
    int iPacketBytes = 0;
}
try
{
    m_wParseTime.Start();

    this.IsCurrent = false;

    bBytesSyncd = this.Sync_Raw_Bytes2();

    if (bBytesSyncd)
    {
        //What kind of packet do we have? Is it Text?
        iPacketBytes = this.Sync_Bytes.Count;

        //if we have a non-text sensor pull
        if (iPacketBytes > 6)
        {
            this.Parse_BumpAndWheelDrops();
            this.Parse_CliffSensors();
            this.Parse_OverCurrents();
            this.Parse_ChargingState();
            this.Parse_Voltage();
            this.Parse_Current();
            this.Parse_Temperature();
            this.Parse_Charge();
            this.Parse_Capacity();
            this.Parse_Wall();
            this.Parse_Virtual_Wall();
            this.Parse_Dirt_Detect();
            this.Parse_Distance();
            this.Parse_Angle();
            this.Parse.Buttons();
            this.LastUpdated = DateTime.Now;

            this.Set_IsCurrent();
            if (this.AutoExport) { this.Hashtable = this.ExportSensors(this.Packet); }
        }
    }
    else
    {
        this.IsCurrent = false;
    }

    bSuccess = true;
}
catch (Exception ex)
{
    string sError = ex.Message; //for debug purposes.
}
finally
```csharp
{ 
    m_wParseTime.Stop();
    TimeSpan tsElapsed = m_wParseTime.Elapsed;
    this.ParseTime = tsElapsed;
    this.Sensor_Parse = m_wParseTime.ElapsedTicks;
    m_wParseTime.Reset();
}

return bSuccess;
}

#region "Supporting Parse Functions"

#region Sync'ing bytes

private bool Sync_Raw_Bytes2()
{
    //Roomba sometimes includes some extra "undocumented bytes" at the start of the packet. I am not sure what they are, but one 
    //of my roombas does it on occasion. This code is my way of filtering it out. I use the 
    remote default of 255 as a way to deduce the start 
    //of packet. Failing that, I set out and count boolean entries in the packet to find out where 
    I am.
    
    bool bSuccess = false;
    int iStart_Of_Packet = 0;
    //Loop through Raw bytes
    bool bFoundIt;
    
    //what we are looking for is the remote control's default value of 255
    bFoundIt = this.Found_Remote_Marker(this.Raw_Bytes, ref iStart_Of_Packet); //The easy way..
    
    //If we didn't find it, then that is because some extra bytes got shoved in at the front of the 
    packet and we have > 26 bytes. (logic elsewhere keeps us from ever seeing less than 26 bytes)
    if (!bFoundIt)
    {
        //We are looking for 8 entries in a row that are Zero or One. This is the method that I use 
        //to locate where in the stream Roomba's packet is..
        bFoundIt = this.Found_8_Bools(this.Raw_Bytes, ref iStart_Of_Packet); //the harder way - which may also mean that someone is currently sending a remote command. this feature is not tested.
    }
    
    bool bCreate_Sync_Bytes = this.Create_Sync_Bytes(iStart_Of_Packet);
    
    bSuccess = bCreate_Sync_Bytes; //so what happens if they don't sync? I wouldn't want to log the volume of errors we would get
```
return bSuccess;
}

private bool Found_8_Bools(List<byte> Bytes_To_Search, ref int iStart_Of_Packet)
{
    bool bSuccess = false;
    List<bool> FoundBools = new List<bool>();
    bool bCurrentMatch = false;

    //Find 7 bools in Bytes_To_Search in which the previous bool examined is also a bool.
    Total = 8 Bools
    int iBoolCount = 0;
    int i = 1;

    try
    {
        //Set BoolCount + 1 if current item == 0 or 1 and previous == 0 or 1
        //I would prefer to use BinarySearch, but these must be consecutive
        for (i = 0; i < Bytes_To_Search.Count; i++) //skip the first byte. That will never be our
        first bool.
        {
            bCurrentMatch = (Bytes_To_Search[i] == 0) || (Bytes_To_Search[i] == 1);
            if (bCurrentMatch)
            {
                FoundBools.Add(bCurrentMatch);
            }
            else
            {
                //ok, so we did not find a bool. this could only mean 1 thing: We aren't in a string
                of bools no more..
                if (iBoolCount < 8)
                {
                    FoundBools.Clear();
                    FoundBools = new List<bool>();
                }
            }
            iBoolCount = FoundBools.Count;
            if (iBoolCount == 8) { break; }
        }

        bSuccess = (iBoolCount == 8);
        if (bSuccess) { iStart_Of_Packet = (i + 1) - (iBoolCount); }
    }
    catch
    {
    }
return bSuccess;
}
private bool Found_Remote_Marker(List<byte> Bytes_To_Search, ref int iStart_Of_Packet)
{
  //I'm not gonna spend all day looking for it. its either here or its not.
  Bytes_To_Search[11]'s range is 0-15.
  //Seeing a 255 there is a sure fire way of telling that those undocumented bytes are
  sneaking in..
  bool bFoundMarker = (Bytes_To_Search[10] == 255) & (Bytes_To_Search[11] < 16);
  if (bFoundMarker) { iStart_Of_Packet = 0; }
  return bFoundMarker;
}
private bool Get_Sync_Bytes(ref int iRemote_Default, ref int iCapacity1, ref int iCapacity2)
{
  bool bSuccess = false;
  try
  {
    iRemote_Default = this.Raw_Bytes.FindIndex(0, IsRemote_Sync);
    iCapacity1 = this.Raw_Bytes.FindIndex(0, IsCapacity1);
    iCapacity2 = this.Raw_Bytes.FindIndex(0, IsCapacity2);
    bSuccess = true;
  }
  catch
  {
  }
  return bSuccess;
}
public bool Create_Sync_Bytes(int iStart_Of_Packet)
{
  bool bSuccess = false;
  try
  {
    this.Sync_Bytes = new List<byte>();
    this.Sync_Bytes.Add(this.Raw_Bytes[iStart_Of_Packet]);
    this.Sync_Bytes.Add(this.Raw_Bytes[iStart_Of_Packet + 1]);
    this.Sync_Bytes.Add(this.Raw_Bytes[iStart_Of_Packet + 2]);
    this.Sync_Bytes.Add(this.Raw_Bytes[iStart_Of_Packet + 3]);
    this.Sync_Bytes.Add(this.Raw_Bytes[iStart_Of_Packet + 4]);
    this.Sync_Bytes.Add(this.Raw_Bytes[iStart_Of_Packet + 5]);
    this.Sync_Bytes.Add(this.Raw_Bytes[iStart_Of_Packet + 6]);
    this.Sync_Bytes.Add(this.Raw_Bytes[iStart_Of_Packet + 7]);
  }
this.Sync_Bytes.Add(this.Raw_Bytes[iStart_Of_Packet + 8]);
this.Sync_Bytes.Add(this.Raw_Bytes[iStart_Of_Packet + 9]);
this.Sync_Bytes.Add(this.Raw_Bytes[iStart_Of_Packet + 10]);
this.Sync_Bytes.Add(this.Raw_Bytes[iStart_Of_Packet + 11]);
this.Sync_Bytes.Add(this.Raw_Bytes[iStart_Of_Packet + 12]);
this.Sync_Bytes.Add(this.Raw_Bytes[iStart_Of_Packet + 13]);
this.Sync_Bytes.Add(this.Raw_Bytes[iStart_Of_Packet + 14]);
this.Sync_Bytes.Add(this.Raw_Bytes[iStart_Of_Packet + 15]);
this.Sync_Bytes.Add(this.Raw_Bytes[iStart_Of_Packet + 16]);
this.Sync_Bytes.Add(this.Raw_Bytes[iStart_Of_Packet + 17]);
this.Sync_Bytes.Add(this.Raw_Bytes[iStart_Of_Packet + 18]);
this.Sync_Bytes.Add(this.Raw_Bytes[iStart_Of_Packet + 19]);
this.Sync_Bytes.Add(this.Raw_Bytes[iStart_Of_Packet + 20]);
this.Sync_Bytes.Add(this.Raw_Bytes[iStart_Of_Packet + 21]);
this.Sync_Bytes.Add(this.Raw_Bytes[iStart_Of_Packet + 22]);
this.Sync_Bytes.Add(this.Raw_Bytes[iStart_Of_Packet + 23]);
this.Sync_Bytes.Add(this.Raw_Bytes[iStart_Of_Packet + 24]);
this.Sync_Bytes.Add(this.Raw_Bytes[iStart_Of_Packet + 25]);

bSuccess = true;
}

catch
{
}

return bSuccess;

// Temporary, till i find a cleaner way of doing this.
private bool IsRemote_Sync(byte b)
{
    if (b == Sensors.c_byPacket_Sync)
    {
        return true;
    }
    else
    {
        return false;
    }
}

private bool IsCapacity1(byte b)
{
    if (b == this.m_byCapacity1)
    {
        return true;
    }
    else
    {
        return false;
    }
}
private bool IsCapacity2(byte b)
{
    if (b == this.m_byCapacity2)
    {
        return true;
    }
    else
    {
        return false;
    }
}

#region
/// <summary>
/// </summary>
private void Parse_BumpAndWheelDrops()
{
    if (this.Sync_Bytes.Count > 0)
    {
        byte byBumpAndWheelDrops = this.Sync_Bytes[0];

        //Bumps
        //this.Packet.Bump.Left = (Convert.ToByte(byBumpAndWheelDrops &
        //Sensor_Code.Bump_Left) != 0);
        //this.Packet.Bump.Right = (Convert.ToByte(byBumpAndWheelDrops &
        //Sensor_Code.Bump_Right) != 0);

        switch (byBumpAndWheelDrops)
        {
                    this.Packet.Bump.Left = false; break;
                    this.Packet.Bump.Left = false; break;
                    this.Packet.Bump.Both = false; break;
                    this.Packet.Bump.Left = false; break;
        }

        //WheelDrops
        this.Packet.WheelDrop.Left = (Convert.ToByte(byBumpAndWheelDrops &
        Sensor_Code.WheelDrop_Left) != 0);
        this.Packet.WheelDrop.Right = (Convert.ToByte(byBumpAndWheelDrops &
        Sensor_Code.WheelDrop_Right) != 0);
        this.Packet.WheelDrop.Caster = (Convert.ToByte(byBumpAndWheelDrops &
        Sensor_Code.WheelDrop_Caster) != 0);

        //endregion

        /// <summary>
        /// </summary>

private void Parse_Wall()
{
    if (this.Sync_Bytes.Count > 1)
    {
        byte byWall = this.Sync_Bytes[1];

        this.Packet.Wall = (byWall == 1);
    }
    else
    {
    }
}

private void Parse_CliffSensors()
{
    try
    {

        if (this.Sync_Bytes.Count > 5)
        {
            byte byCliffLeft = this.Sync_Bytes[2];
            byte byCliffFrontLeft = this.Sync_Bytes[3];
            byte byCliffFrontRight = this.Sync_Bytes[4];
            byte byCliffRight = this.Sync_Bytes[5];

            this.Packet.Cliff.Left = (byCliffLeft == 1);
            this.Packet.Cliff.FrontLeft = (byCliffFrontLeft == 1);
            this.Packet.Cliff.FrontRight = (byCliffFrontRight == 1);
            this.Packet.Cliff.Right = (byCliffRight == 1);
        }
        else
        {
        }
    }

    catch (Exception ex)
    {
        string sError = ex.Message;
    }
}

private void Parse_OverCurrents()
{
    if (this.Sync_Bytes.Count > 6)
byte byOverCurrent = this.Sync_Bytes[7];

//OverCurrent
this.Packet.OverCurrent.Left_Wheel = (Convert.ToByte(byOverCurrent & Sensor_Code.OverCurrent_Drive_Left) != 0);
this.Packet.OverCurrent.Right_Wheel = (Convert.ToByte(byOverCurrent & Sensor_Code.OverCurrent_Drive_Right) != 0);
this.Packet.OverCurrent.Side_Brush = (Convert.ToByte(byOverCurrent & Sensor_Code.OverCurrent_Side_Brush) != 0);
this.Packet.OverCurrent.Main_Brush = (Convert.ToByte(byOverCurrent & Sensor_Code.OverCurrent_Main_Brush) != 0);
this.Packet.OverCurrent.Vacuum = (Convert.ToByte(byOverCurrent & Sensor_Code.OverCurrent_Vacuum) != 0);
}

private void Parse_Dirt_Detect()
{
    if (this.Sync_Bytes.Count > 7)
    {
        this.Packet.Dirt_Detector.Left = this.Sync_Bytes[8];
        this.Packet.Dirt_Detector.Right = this.Sync_Bytes[9];
    }
}

private void Parse_Virtual_Wall()
{
    if (this.Sync_Bytes.Count > 5)
    {
        this.Packet.Virtual_Wall = this.Sync_Bytes[6] != 0;
    }
}

private void Parse_Remote_Control()
{
    if (this.Sync_Bytes.Count > 9)
    {
        this.Packet.Remote_Control_Command = this.Sync_Bytes[10];
    }
}

private void Parse_Angle()
{
    if (this.Sync_Bytes.Count > 13)
    {
        short difference = (short)((Sync_Bytes[14] << 8) | Sync_Bytes[15]);

        if (difference > 360) difference -= 360; // Limit to +-360
        if (difference < -360) difference += 360;

        this.Packet.Angle = difference; // (short)((2 * difference) / 258); // radians
    }
}
private void Parse_Distance()
{
    if (this.Sync_Bytes.Count > 11)
    {
        try
        {
            byte[] byDistance = { this.Sync_Bytes[12], this.Sync_Bytes[13] };
            this.Packet.Distance = (short)((Sync_Bytes[12] << 8) | Sync_Bytes[13]);
        }
        catch (Exception ex)
        {
            //Throw error of some sort
        }
    }
}

private void Parse.Buttons()
{
    if (this.Sync_Bytes.Count > 10)
    {
        byte byButtons = this.Sync_Bytes[11];

        //Buttons
        this.Packet.Buttons.Power = (Convert.ToByte(byButtons & Sensor_Code.Buttons_Power) != 0);
        this.Packet.Buttons.Spot = (Convert.ToByte(byButtons & Sensor_Code.Buttons_Spot) != 0);
        this.Packet.Buttons.Clean = (Convert.ToByte(byButtons & Sensor_Code.Buttons_Clean) != 0);
        this.Packet.Buttons.Max = (Convert.ToByte(byButtons & Sensor_Code.Buttons_Max) != 0);
    }
}

private void Parse.ChargingState()
{
    if (this.Sync_Bytes.Count > 15)
    {
        byte byChargingState = this.Sync_Bytes[16];

        this.Packet.Charging_State = RoombaSCI.Charging_State.Indeterminate;
        switch (byChargingState)
        {
            case 0:
                this.Packet.Charging_State = RoombaSCI.Charging_State.Not_Charging;
                break;
            case 1:
                this.Packet.Charging_State = RoombaSCI.Charging_State.Charging_Recovery;
                break;
            case 2:
                this.Packet.Charging_State = RoombaSCI.Charging_State.Charging;
                break;
        }
    }
}
case 3:
    this.Packet.Charging_State = RoombaSCI.Charging_State.Trickle_Charging;
    break;
    case 4:
        this.Packet.Charging_State = RoombaSCI.Charging_State.Waiting;
        break;
    case 5:
        this.Packet.Charging_State = RoombaSCI.Charging_State.Charging_Error;
        break;
    }

    }

private void Parse_Voltage()
{
    if (this.Sync_Bytes.Count > 16)
        
        byte[] byVoltage = { this.Sync_Bytes[17], this.Sync_Bytes[18] };
        this.Packet.Voltage = BitConverter.ToUInt16(byVoltage, 0);
    }

private void Parse_Current()
{
    if (this.Sync_Bytes.Count > 18)
        
        byte[] byCurrent = { this.Sync_Bytes[19], this.Sync_Bytes[20] };
        this.Packet.Current = BitConverter.ToInt16(byCurrent, 0);
    }

private void Parse_Temperature()
{
    if (this.Sync_Bytes.Count > 20)
        
        this.Packet.Temperature = this.Sync_Bytes[21];
    }

private void Parse_Charge()
{
    if (this.Sync_Bytes.Count > 21)
        
        byte[] byCharge = { this.Sync_Bytes[22], this.Sync_Bytes[23] };
        this.Packet.Charge = BitConverter.ToUInt16(byCharge, 0);
    }

private void Parse_Capacity()
{
    if (this.Sync_Bytes.Count > 23)
        
        byte[] byCapacity = { this.Sync_Bytes[24], this.Sync_Bytes[25] };
        this.Packet.Capacity = BitConverter.ToUInt16(byCapacity, 0);
/// <summary>
/// 
/// </summary>
/// <param name="lParse_Into_Sensor_Structure"></param>
/// <returns></returns>
public bool Parse(List<byte> lParse_Into_Sensor_Structure)
{
    return false;
}

private void Set_IsCurrent()
{
    double iPollInterval = this.PollingInterval;

    DateTime dtNow = DateTime.Now;
    DateTime dtLastUpdated = this.LastUpdated;
    TimeSpan tsResult = dtNow - dtLastUpdated;
    TimeSpan tsCalculation = tsResult.Add(TimeSpan.FromMilliseconds(this.PollingInterval +
    this.PollingInterval));
    TimeSpan tsCalcResult = tsCalculation.Subtract(tsResult);

    this.UpdateResponse = tsCalcResult;

    double dResult = (dtNow - dtLastUpdated).TotalMilliseconds;

    this.IsCurrent = (dResult < Convert.ToDouble(this.IsCurrent_Threshold));
    if (this.IsCurrent)
    {
        this.LastUpdated = dtNow;
    }
}

/// <summary>
/// 
/// </summary>
/// <param name="interval"></param>
/// <returns></returns>
public bool Set_Polling_Interval(double interval)
{
    try
    {
        if (interval > 0)
        {
            //this.m_tPolling = new Timer(interval);
            this.PollingInterval = interval;
        }
    }
    catch (Exception e)
    {
        throw;
    }
}
/this.m_tPolling.Elapsed += new System.Timers.ElapsedEventHandler(this.m_tPolling_IntervalElapsed);

else
{
    this.PollingInterval = 0;
}

catch (Exception ex)
{
    throw new SensorsException("An Error has occurred setting polling interval " + ex.Message);
}
return true;

//Move this to Sensors Object..
public Hashtable ExportSensors(RoombaSCI.Sensor_Packet sensors)
{
    Hashtable hLookup = new Hashtable();

    hLookup.Add(c_sAngle, sensors.Angle);
    hLookup.Add(c_sBumpRight, sensors.Bump.Both);
    hLookup.Add(c_sBumpRight, sensors.Bump.Right);
    hLookup.Add(c_sBumpLeft, sensors.Bump.Left);
    hLookup.Add(c_sClean, sensors.Buttons.Clean);
    hLookup.Add(c_sMax, sensors.Buttons.Max);
    hLookup.Add(c_sPower, sensors.Buttons.Power);
    hLookup.Add(c_sSpot, sensors.Buttons.Spot);
    hLookup.Add(c_sCapacity, sensors.Capacity);
    hLookup.Add(c_sCharge, sensors.Charge);
    hLookup.Add(c_sChargingState, sensors.Charging_State);
    hLookup.Add(c_sCliffFrontLeft, sensors.Cliff.FrontLeft);
    hLookup.Add(c_sCliffFrontRight, sensors.Cliff.FrontRight);
    hLookup.Add(c_sCliffLeft, sensors.Cliff.Left);
    hLookup.Add(c_sCliffRight, sensors.Cliff.Right);
    hLookup.Add(c_sCurrent, sensors.Current);
    hLookup.Add(c_sDirtDetectorLeft, sensors.Dirt_Detector.Left);
    hLookup.Add(c_sDirtDetectorRight, sensors.Dirt_Detector.Right);
    hLookup.Add(c_sDistance, sensors.Distance);
    hLookup.Add(c_sOverCurrentLeft, sensors.OverCurrent.Left_Wheel);
    hLookup.Add(c_sOverCurrentRight, sensors.OverCurrent.Right_Wheel);
    hLookup.Add(c_sOverCurrentMAIN, sensors.OverCurrent.Main_Brush);
    hLookup.Add(c_sOverCurrentSide, sensors.OverCurrent.Side_Brush);
    hLookup.Add(c_sOverCurrentVacuum, sensors.OverCurrent.Vacuum);
    hLookup.Add(c_sRemote, sensors.Remote_Control_Command);
    hLookup.Add(c_sTemperature, sensors.Temperature);
    hLookup.Add(c_sVirtualWall, sensors.Virtual_Wall);
    hLookup.Add(c_sVoltage, sensors.Voltage);
    hLookup.Add(c_sWall, sensors.Wall);
    hLookup.Add(c_sWheelDropCaster, sensors.WheelDrop.Caster);
hLookup.Add(c_sWheelDropLeft, sensors.WheelDrop.Left);
hLookup.Add(c_sWheelDropNone, sensors.WheelDrop.None);
hLookup.Add(c_sWheelDropRight, sensors.WheelDrop.Right);

    //This is how you look up..
    //string x = hLookup["Angle"].ToString();

    return hLookup;
  }

#endregion


Roomba Poller

using System;
using System.IO;
using System.Text;
using System.IO.Ports;
using System.Diagnostics;
using System.Threading;
using System.Collections.Generic;
using Kevin_Logging;

namespace RoombaSCI
{

  [Serializable]
  public class Roomba_Poller: Roomba
  {
    //region Member Variables

    Stopwatch m_wPollTime = new Stopwatch();
    
#endregion

    #region Constants

    protected const string c_sRoomba_Poller = "Roomba_Poller";
    protected const string c_sSensorPoll = "Sensor Poll: ";
    protected const string c_sSettingIO = "Setting IO Port";

    #endregion

    public Roomba_Poller()
    {
    }
  }
}
public Roomba_Poller(SerialPort IO, double Sensor_Polling_Interval, string sLogPath) : base(IO, Sensor_Polling_Interval, sLogPath)
{
    this.AutoPollingCheck = new RoombaSCI.Timer();
    this.AutoPollingCheck.Period = 200;

    // Hook up the Elapsed event for the timer.
    this.AutoPollingCheck.Tick += new EventHandler(OnTimedEvent);
    this.AutoPollingCheck.Start();

    this.Macro = new RoombaSCI.Macro(this);

    //this.ConnectionTime = new RoombaSCI.Timer();
    //this.ConnectionTime.Period = 100; //default

    ///// Hook up the Elapsed event for the timer.
    ///this.ConnectionTime.Elapsed += new ElapsedEventHandler(OnTimedEvent);
    //this.ConnectionTime.Start();

}

public void HookUp()
{
    this.IO = new System.IO.Ports.SerialPort();

    if (this.IO != null)
    {
        // this.IO.DataReceived += new System.IO.Ports.SerialDataReceivedEventHandler(this.Roomba_DataReceived);
        // this.IO.ErrorReceived += new System.IO.Ports.SerialErrorReceivedEventHandler(this.Roomba_DataErrorRecieved);
    }
}

#region Properties

protected Thread p_tPollThread = null;
public Thread PollThread
{
    get
    {
        return (this.p_tPollThread);
    }

    set
    {
        this.p_tPollThread = value;
    }
}

private RoombaSCI.Timer p_tAutoPollingCheck;
public RoombaSCI.Timer AutoPollingCheck
{
    get
    {
        return (this.p_tAutoPollingCheck);
    }
    set
    {
        this.p_tAutoPollingCheck = value;
    }
}

private Timer p_tConnectionTime;
public RoombaSCI.Timer ConnectionTime
{
    get
    {
        return (this.p_tConnectionTime);
    }
    set
    {
        this.p_tConnectionTime = value;
    }
}

private bool p_bPolling;
public bool Polling
{
    get
    {
        return (this.p_bPolling);
    }
    set
    {
        this.p_bPolling = value;
    }
}

private long p_lPollTicks;
public long PollTicks
{
    get
    {
        return (this.p_lPollTicks);
    }
    set
    {
        this.p_lPollTicks = value;
    }
}
private bool p_bPoller_LogPermission;
public bool Poller_LogPermission
{
   get
   {
      return (this.p_bPoller_LogPermission);
   }
   set
   {
      this.p_bPoller_LogPermission = value;
   }
}

#region Event Handlers
public void Roomba_DataReceived(object sender, SerialDataReceivedEventArgs e)
{
   //m_bRCV_Err = false;

   //Read existing here when not polling
   string m_sReturn = e.ToString() + DateTime.Now.ToString();
   //m_iBytesToRead = Program.UI.CurrentRoomba.IO.BytesToRead;
   //m_sReturn = Program.UI.CurrentRoomba.IO.ReadExisting();

   //MessageBox.Show(m_sReturn);

   //this.UpdateForm();

   //this.pSensorRCV.BackColor = Color.Green;
   //if (this.chkFlashStatus.Checked)
   //{  
   //   this.m_bTextRecieved = true;
   //   this.SetCommStatus();
   //}; //Feedback to Roomba to show that we have communication

   //g_sRCV  = this.CurrentRoomba.IO.ReadExisting();
   //int iBytesToRead = this.CurrentRoomba.IO.BytesToRead;

   //this.SetText();

   //Application.DoEvents();
}

public void Roomba_DataErrorRecieved(object sender, SerialErrorReceivedEventArgs e)
{
   string m_sReturn = this.IO.ReadExisting();
   //m_bRCV_Err = true;
}

private void OnTimedEvent(object sender, System.EventArgs e)
if (this.Automatic_Polling)
{
    if (this.PollThread == null)
    {
        this.Start_Automatic_Polling();
    }
    else
    {
        m_wPollTime.Stop();
        this.PollTicks = m_wPollTime.ElapsedMilliseconds;
        this.Polling = true;
        m_wPollTime.Reset();
        m_wPollTime.Start();
    }

    // this.AutoPollingCheck.Period = this.Config
}
else
{
    m_wPollTime.Stop();
    this.PollTicks = m_wPollTime.ElapsedTicks;
    this.Polling = false;
    if (this.PollThread != null) {this.Stop_Automatic_Polling();}
}

#endregion Functions

#region Polling

/// <summary>
/// Starts the Roomba class Automatic polling routine in a new thread. As of v1.01, this function currently polls all sensors
/// </summary>
public void Start_Automatic_Polling()
{
    //Yank off the taps into COMM Port
    //this.IO.DataReceived -= new System.IO.Ports.SerialDataReceivedEventHandler(this.Roomba_DataReceived);
    //this.IO.ErrorReceived -= new System.IO.Ports.SerialErrorReceivedEventHandler(this.Roomba_DataErrorRecieved);
    this.PollThread = new Thread(new ThreadStart(this.PollMe));

    System.IO.Ports.SerialDataReceivedEventHandler roomba_dataReceivedHandler = this.Roomba_DataReceived;
    System.IO.Ports.SerialErrorReceivedEventHandler roomba_dataErrorRecievedHandler = this.Roomba_DataErrorRecieved;

    this.IO.DataReceived -= roomba_dataReceivedHandler;
    this.IO.ErrorReceived -= roomba_dataErrorRecievedHandler;

    this.PollThread.Start();
}

#endregion
this.PollThread.Start();
this.Macro.SetAction("Start_Automatic_Polling");
}

/// <summary>
/// Stops the Roomba class Automatic polling routine (aborts thread)
/// </summary>
protected void Stop_Automatic_Polling()
{
    if (this.PollThread != null)
    {
        this.PollThread.Abort();
        this.Macro.SetAction("Stop_Automatic_Polling");
    }

    //If we are not polling, then we can set up to recieve Text from Roomba
    //if (this.IO != null)
    //{
    //    this.IO.DataReceived += new System.IO.Ports.SerialDataReceivedEventHandler(this.Roomba_DataReceived);
    //    this.IO.ErrorReceived += new System.IO.Ports.SerialErrorReceivedEventHandler(this.Roomba_DataErrorRecieved);
    //}
}

/// <summary>
/// supporting function for the Automatic polling routine
/// </summary>
private void PollMe()
{
    if (!this.Sensors.Lock)
    {
        do
        {
            this.Poll_Sensors(0);
            Thread.Sleep((int)this.Sensors.PollingInterval);
        } while (!this.Sensors.StopPolling);
    }
}

/// <summary>
///  The Command to Poll Roomba's sensors
/// </summary>
/// <param name="byPollType"></param>
/// <returns></returns>
public bool Poll_Sensors(byte byPollType)
{
    bool bSuccess = false;
    byte[] b = new byte[2];
b[0] = OpCode.Sensors;
b[1] = byPollType;

try
{
    this.IO.RtsEnable = true;
    this.Sensors.LastUpdated = new DateTime();

    if (this.Mode != SCI_Mode.Off)
    {
        Log.This(c_sSensorPoll, c_sRoomba_Poller, this.LogIO);
        this.IO.Write(b, 0, b.Length);
    }

    //For debugging purposes
    //int iBytesInBuffer = this.IO.BytesToRead;
    //this.LogThis("Bytes Found in Buffer: " + iBytesInBuffer.ToString());

    //if (iBytesInBuffer > 0)
    //{  
    //    string sRCV = this.IO.ReadExisting().ToString();
    //    //}

    bSuccess = true;
}
catch (Exception ex)
{
    Log.This("Sensor Poll Fail: " + ex.Message, c_sRoomba_Poller, this.LogIO); //This may not always be an error, someone could have shut Roomba Off
}

return bSuccess;
}

public void Start(string Port_Name, bool Start_Polling)
{
    Log.This(c_sSettingIO, c_sRoomba_Poller, this.LogIO);

    this.IO = new SerialPort();
    this.IO.PortName = Port_Name;
    this.IO.BaudRate = 57600;
    this.IO.DataBits = 8;
    this.IO.DtrEnable = false;
    this.IO.StopBits = StopBits.One;
    this.IO.Handshake = Handshake.None;
    this.IO.Parity = Parity.None;
    this.IO.RtsEnable = false;
    this.IO.Close();
    this.IO.Open();
Log.This("IO Port: " + Port_Name + " Open:", c_sRoomba_Poller, this.LogIO);

if (Start_Polling)
{
    GC.Collect();

    this.SetMode(SCI_Mode.Passive);

    if (this.PollThread != null)
    {
        try { this.PollThread.Abort(); } 
        catch { } 
        finally { this.PollThread = null; }
    }

    this.Automatic_Polling = true; //A timer will come along and check the property & will start //this.Start_Automatic_Polling();
    this.Macro.SetAction("Start");
}

#region License

/* Copyright (c) 2006 Leslie Sanford 
 *
* Permission is hereby granted, free of charge, to any person obtaining a copy
* of this software and associated documentation files (the "Software"), to
* deal in the Software without restriction, including without limitation the
* rights to use, copy, modify, merge, publish, distribute, sublicense, and/or
* sell copies of the Software, and to permit persons to whom the Software is
* furnished to do so, subject to the following conditions:
* 
* The above copyright notice and this permission notice shall be included in
* all copies or substantial portions of the Software.
* 
* THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
* IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
* FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
* AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
namespace RoombaSCI
{
    /// <summary>
    /// Defines constants for the multimedia Timer's event types.
    /// </summary>
    public enum TimerMode
    {
        /// <summary>
        /// Timer event occurs once.
        /// </summary>
        OneShot,

        /// <summary>
        /// Timer event occurs periodically.
        /// </summary>
        Periodic
    }

    /// <summary>
    /// Represents information about the multimedia Timer's capabilities.
    /// </summary>
    [StructLayout(LayoutKind.Sequential)]
    public struct TimerCaps
    {
        /// <summary>
        /// Minimum supported period in milliseconds.
        /// </summary>
    }
}
public int periodMin;

/// <summary>
/// Maximum supported period in milliseconds.
/// </summary>
public int periodMax;
}

/// <summary>
/// Represents the Windows multimedia timer.
/// http://www.codeproject.com/cs/miscctrl/lescsmultimediatimer.asp
/// </summary>
public sealed class Timer : IComponent
{
    #region Timer Members

    #region Delegates

    // Represents the method that is called by Windows when a timer event occurs.
    private delegate void TimeProc(int id, int msg, int user, int param1, int param2);

    // Represents methods that raise events.
    private delegate void EventRaiser(EventArgs e);

    #endregion

    #region Win32 Multimedia Timer Functions

    // Gets timer capabilities.
    [DllImport("winmm.dll")]
    private static extern int timeGetDevCaps(ref TimerCaps caps, int sizeOfTimerCaps);

    // Creates and starts the timer.
    [DllImport("winmm.dll")]
    private static extern int timeSetEvent(int delay, int resolution, TimeProc proc, int user, int mode);

    // Stops and destroys the timer.
    [DllImport("winmm.dll")]
    private static extern int timeKillEvent(int id);

    // Indicates that the operation was successful.
    private const int TIMERR_NOERROR = 0;

    #endregion

    #region Fields

    // Timer identifier.
    private int timerID;

    #endregion

// Timer mode.
private volatile TimerMode mode;

// Period between timer events in milliseconds.
private volatile int period;

// Timer resolution in milliseconds.
private volatile int resolution;

// Called by Windows when a timer periodic event occurs.
private TimeProc timeProcPeriodic;

// Called by Windows when a timer one shot event occurs.
private TimeProc timeProcOneShot;

// Represents the method that raises the Tick event.
private EventRaiser tickRaiser;

// Indicates whether or not the timer is running.
private bool running = false;

// Indicates whether or not the timer has been disposed.
private volatile bool disposed = false;

// The ISynchronizeInvoke object to use for marshaling events.
private ISynchronizeInvoke synchronizingObject = null;

// For implementing IComponent.
private ISite site = null;

// Multimedia timer capabilities.
private static TimerCaps caps;

#endregion

#region Events

/// <summary>
/// Occurs when the Timer has started;
/// </summary>
public event EventHandler Started;

/// <summary>
/// Occurs when the Timer has stopped;
/// </summary>
public event EventHandler Stopped;

/// <summary>
/// Occurs when the time period has elapsed.
/// </summary>
public event EventHandler Tick;

#endregion

#region Construction

/// <summary>
/// Initialize class.
/// </summary>
static Timer()
{
    // Get multimedia timer capabilities.
    timeGetDevCaps(ref caps, Marshal.SizeOf(caps));
}

/// <summary>
/// Initializes a new instance of the Timer class with the specified IContainer.
/// </summary>
/// <param name="container">
/// The IContainer to which the Timer will add itself.
/// </param>
public Timer(IContainer container)
{
    ///
    /// Required for Windows.Forms Class Composition Designer support
    ///
    container.Add(this);

    Initialize();
}

/// <summary>
/// Initializes a new instance of the Timer class.
/// </summary>
public Timer()
{
    Initialize();
}

~Timer()
{
    if (IsRunning)
    {
        // Stop and destroy timer.
        timeKillEvent(timerID);
    }
}

// Initialize timer with default values.
private void Initialize()
{

this.mode = TimerMode.Periodic;
this.period = Capabilities.periodMin;
this.resolution = 1;

running = false;

timeProcPeriodic = new TimeProc(TimerPeriodicEventCallback);
timeProcOneShot = new TimeProc(TimerOneShotEventCallback);
tickRaiser = new EventRaiser(OnTick);
}

#endregion

#region Methods

/// <summary>
/// Starts the timer.
/// </summary>
/// <exception cref="ObjectDisposedException">
/// The timer has already been disposed.
/// </exception>
/// <exception cref="TimerStartException">
/// The timer failed to start.
/// </exception>
public void Start()
{
    #region Require

    if (disposed)
    {
        throw new ObjectDisposedException("Timer");
    }

    #endregion

    #region Guard

    if (IsRunning)
    {
        return;
    }

    #endregion

    // If the periodic event callback should be used.
    if (Mode == TimerMode.Periodic)
    {
        // Create and start timer.
        timerID = timeSetEvent(Period, Resolution, timeProcPeriodic, 0, (int)Mode);
    }
    // Else the one shot event callback should be used.
else
{
    // Create and start timer.
    timerID = timeSetEvent(Period, Resolution, timeProcOneShot, 0, (int)Mode);
}

// If the timer was created successfully.
if (timerID != 0)
{
    running = true;

    if (SynchronizingObject != null && SynchronizingObject.InvokeRequired)
    {
        SynchronizingObject.BeginInvoke(
            new EventRaiser(OnStarted),
            new object[] { EventArgs.Empty });
    }
    else
    {
        OnStarted(EventArgs.Empty);
    }
    else
    {
        throw new TimerStartException("Unable to start multimedia Timer.");
    }
}

/// <summary>
/// Stops timer.
/// </summary>
/// <exception cref="ObjectDisposedException">If the timer has already been disposed.
public void Stop()
{
    #region Require

    if (disposed)
    {
        throw new ObjectDisposedException("Timer");
    }

    #endregion

    #region Guard

    // If running

    #endregion

#endregion

    // Stop and destroy timer.
    int result = timeKillEvent(timerID);
    Debug.Assert(result == TIMERR_NOERROR);

    running = false;

    if (SynchronizingObject != null && SynchronizingObject.InvokeRequired)
    {
        SynchronizingObject.BeginInvoke(
            new EventRaiser(OnStopped),
            new object[] { EventArgs.Empty });
    }
    else
    {
        OnStopped(EventArgs.Empty);
    }

#region Callbacks

    // Callback method called by the Win32 multimedia timer when a timer
    // periodic event occurs.
    private void TimerPeriodicEventCallback(int id, int msg, int user, int param1, int param2)
    {
        if (synchronizingObject != null)
        {
            synchronizingObject.BeginInvoke(tickRaiser, new object[] { EventArgs.Empty });
        }
        else
        {
            OnTick(EventArgs.Empty);
        }
    }

    // Callback method called by the Win32 multimedia timer when a timer
    // one shot event occurs.
    private void TimerOneShotEventCallback(int id, int msg, int user, int param1, int param2)
    {
        if (synchronizingObject != null)
        {
            synchronizingObject.BeginInvoke(tickRaiser, new object[] { EventArgs.Empty });
            Stop();
        }
        else
        {
            OnTick(EventArgs.Empty);
            Stop();
        }
    }

#endregion
#endregion

#region Event Raiser Methods

// Raises the Disposed event.
private void OnDisposed(EventArgs e)
{
    EventHandler handler = Disposed;

    if (handler != null)
    {
        handler(this, e);
    }
}

// Raises the Started event.
private void OnStarted(EventArgs e)
{
    EventHandler handler = Started;

    if (handler != null)
    {
        handler(this, e);
    }
}

// Raises the Stopped event.
private void OnStopped(EventArgs e)
{
    EventHandler handler = Stopped;

    if (handler != null)
    {
        handler(this, e);
    }
}

// Raises the Tick event.
private void OnTick(EventArgs e)
{
    EventHandler handler = Tick;

    if (handler != null)
    {
        handler(this, e);
    }
}
/// <summary>
/// Gets or sets the object used to marshal event-handler calls.
/// </summary>
public ISynchronizeInvoke SynchronizingObject
{
    get
    {
        #region Require
        if (disposed)
        {
            throw new ObjectDisposedException("Timer");
        }
        #endregion
        return synchronizingObject;
    }
    set
    {
        #region Require
        if (disposed)
        {
            throw new ObjectDisposedException("Timer");
        }
        #endregion
        synchronizingObject = value;
    }
}

/// <summary>
/// Gets or sets the time between Tick events.
/// </summary>
/// <exception cref="ObjectDisposedException">If the timer has already been disposed.</exception>
public int Period
{
    get
    {
        #region Require

    }
if (disposed)
{
    throw new ObjectDisposedException("Timer");
}
#region

return period;

#endregion

set
{
    #region Require

    if (disposed)
    {
        throw new ObjectDisposedException("Timer");
    }
    else if (value < Capabilities.periodMin || value > Capabilities.periodMax)
    {
        throw new ArgumentOutOfRangeException("Period", value,
            "Multimedia Timer period out of range.");
    }
    #endregion

    period = value;

    if (IsRunning)
    {
        Stop();
        Start();
    }
}

/// <summary>
/// Gets or sets the timer resolution.
/// </summary>
/// <exception cref="ObjectDisposedException">If the timer has already been disposed.</exception>
/// <remarks>
/// The resolution is in milliseconds. The resolution increases
/// with smaller values; a resolution of 0 indicates periodic events
/// should occur with the greatest possible accuracy. To reduce system
/// overhead, however, you should use the maximum value appropriate
/// for your application.
/// </remarks>
public int Resolution
{
    get
    {
{  
    #region Require
    
    if (disposed)
    {
    
        throw new ObjectDisposedException("Timer");
    }
    #endregion
    
    return resolution;
}

set
{
    #region Require

    if (disposed)
    {
        throw new ObjectDisposedException("Timer");
    }
    else if (value < 0)
    {
        throw new ArgumentOutOfRangeException("Resolution", value, "Multimedia timer resolution out of range.");
    }
    #endregion

    resolution = value;

    if (IsRunning)
    {
        Stop();
        Start();
    }
}

/// <summary>
/// Gets the timer mode.
/// </summary>
/// <exception cref="ObjectDisposedException">If the timer has already been disposed.
/// </exception>
public TimerMode Mode
{
    get
    {
        #region Require

        if (disposed)
{ 
    throw new ObjectDisposedException("Timer"); 
}

#region Require

if (disposed)
{
    throw new ObjectDisposedException("Timer");
}
#endregion

return mode;
}

set
{
    #region Require

    if (disposed)
    {
        throw new ObjectDisposedException("Timer");
    }
    #endregion

    mode = value;

    if (IsRunning)
    {
        Stop();
        Start();
    }
}

/// <summary>
/// Gets a value indicating whether the Timer is running.
/// </summary>
public bool IsRunning
{
    get
    {
        return running;
    }
}

/// <summary>
/// Gets the timer capabilities.
/// </summary>
public static TimerCaps Capabilities
{
    get
    {
        return caps;
    }
}
#endregion

#region IComponent Members

public event System.EventHandler Disposed;

public ISite Site
{
    get
    {
        return site;
    }
    set
    {
        site = value;
    }
}

#endregion

#region IDisposable Members

/// <summary>
/// Frees timer resources.
/// </summary>
/// <summary>
public void Dispose()
{
    #region Guard

    if (disposed)
    {
        return;
    }

    #endregion

    if (IsRunning)
    {
        Stop();
    }

disposed = true;

    OnDisposed(EventArgs.Empty);
}

#endregion
/**<summary>
**The exception that is thrown when a timer fails to start.
/**</summary>
public class TimerStartException : ApplicationException
{
  /// <summary>
  /// Initializes a new instance of the TimerStartException class.
  /// </summary>
  /// <param name="message">The error message that explains the reason for the exception.
  /// </param>
  public TimerStartException(string message)
  : base(message)
  {
  }
}
using System;
using System.IO;
using System.Xml.Serialization;
using System.IO.Ports;
using System.Drawing;
using System.Windows.Forms;
using System.Collections.Generic;
using System.Text;
using RoombaSCI;
using Kevin_Logging;

namespace roomba_term
{
    [Serializable]
    public class RoombaUI
    {
        #region Constants
        protected const string c_sLogStart = "Log Start\n\n";
        protected const string c_sRoombaUI = "RoombaUI";
        protected const string c_sStartRoomba = "Start Roomba";
        protected const string c_sMacroStart = "Macro Start";
        #endregion

        //Property Flash_Connection
        #region Properties

        private Config_Settings p_csConfig = null;
        public Config_Settings Config
        {
            get
            {
                return p_csConfig;
            }
            set
            {
                p_csConfig = value;
            }
        }

        private Statistics p_sStatistics = null;
        public Statistics Statistics
        {
            get
            {
                return p_sStatistics;
            }
            set
            {
                p_sStatistics = value;
            }
        }

        #endregion
    }
}
```csharp
get
{
    return p_sStatistics;
}
set
{
    p_sStatistics = value;
}

// Later, this will be a list of Roombas.
private Roomba_Poller p_rCurrentRoomba = null;
public Roomba_Poller CurrentRoomba
{
    get
    {
        return p_rCurrentRoomba;
    }
    set
    {
        p_rCurrentRoomba = value;
    }
}

private List<Line2D> p_lCharts = null;
public List<Line2D> Charts
{
    get
    {
        return p_lCharts;
    }
    set
    {
        p_lCharts = value;
    }
}

private bool p_bStarted = false;
public bool Started
{
    get
    {
        return p_bStarted;
    }
    set
    {
        p_bStarted = value;
    }
}

private bool p_bDebugMode = false;
```
public bool DebugMode
{
    get
    {
        return p_bDebugMode;
    }
    set
    {
        p_bDebugMode = value;
    }
}

private bool p_bOpen_For_Restart = false;
public bool Open_For_Restart
{
    get
    {
        return p_bOpen_For_Restart;
    }
    set
    {
        p_bOpen_For_Restart = value;
    }
}

private bool p_bSuspended = false;
public bool Suspended
{
    get
    {
        return p_bSuspended;
    }
    set
    {
        p_bSuspended = value;
    }
}

private string p_sLogPath;
public string LogPath
{
    get
    {
        return p_sLogPath;
    }
    set
    {
        p_sLogPath = value;
    }
}
private string p_sMacroPath;
public string MacroPath
{
    get
    {
        return p_sMacroPath;
    }
    set
    {
        p_sMacroPath = value;
    }
}

#region

public RoombaUI()
{
    this.Config = new Config_Settings();
    this.Statistics = new Statistics();
}

public void Start_Connection(bool setupLog, bool setupMacro, string appVersion)
{
    //string sCOMM_PORT;
    Log.This("Start Connection", c_sRoombaUI, this.Config.Log.RoombaUI);

    if (this.CurrentRoomba != null)
    {
        // sCOMM_PORT = //Whatever it was set at.
    }

    Log.This("Creating New Roomba Class", c_sRoombaUI, this.Config.Log.RoombaUI);

    //Creating the Roomba Object requires you to set the Log path
    this.CurrentRoomba = new Roomba_Poller(new SerialPort(), this.Config.Polling.Frequency,
                                             this.Config.Log.Path);

    if (setupLog) { this.Setup_Log(appVersion); }
    if (setupMacro) { this.Setup_Macro(appVersion); }

    this.CurrentRoomba.Macro.FilePath = Program.UI.MacroPath; //you can get away with not
    this.CurrentRoomba.Do_Not_Parse_RCV = false;
    this.CurrentRoomba.LogSCICommands = Program.UI.Config.Log.ILogSCICommands;
    this.CurrentRoomba.LogIO = Program.UI.Config.Log.Roomba_IO;

    Log.This(c_sStartRoomba, c_sRoombaUI, this.Config.Log.RoombaUI);
    this.CurrentRoomba.Start(this.Config.COMM.ConnectedTo, this.Config.Polling.Sensors);
if(this.CurrentRoomba != null)
{
    this.CurrentRoomba.Automatic_Polling = this.Config.Polling.Sensors; //Roomba Poller will pick this up & Start Roomba
}

this.Started = true;
}

public void Setup_Macro(string appVersion)
{
    Program.UI.CurrentRoomba.Macro = new Macro(Program.UI.CurrentRoomba);

    System.IO.StreamWriter swLogWriter = new System.IO.StreamWriter(Program.UI.MacroPath);
    swLogWriter.Write(c_sMacroStart + " " + appVersion + "\r\n\r\n");
    swLogWriter.Close();

    Log.This("MacroWriter Initialized.", this.GetType().ToString(), true);
}

public void Setup_Log(string appVersion)
{
    Log.Path = Program.UI.LogPath = Application.ExecutablePath + " Log " + Log.GetTimeStamp(false) + ".txt";

    System.IO.StreamWriter swLogWriter = new System.IO.StreamWriter(Program.UI.LogPath);
    swLogWriter.Write(c_sLogStart);
    swLogWriter.Close();

    Log.This("LogWriter Initialized.", this.GetType().ToString(), true);
}

public void Stop_Connection(bool bDestroyRoombaObj)
{
    Log.This("Stop Connection", c_sRoombaUI, this.Config.Log.RoombaUI);

    this.Started = false;

    if (this.CurrentRoomba != null)
    {
        if (this.CurrentRoomba.ConnectionTime != null)
        {
            this.CurrentRoomba.ConnectionTime.Stop();
        }

        this.CurrentRoomba.Automatic_Polling = false;
        this.CurrentRoomba.SetMode(SCI_Mode.Off);

        if (this.CurrentRoomba.IO != null)
        {
            this.CurrentRoomba.IO.Close();
        }
    }
this.CurrentRoomba.IO = null;
}

if (bDestroyRoombaObj)
{
    this.CurrentRoomba = null;
}
}

public void SetPictureBox(bool bSensor, PictureBox pBox)
{
    if (bSensor)
    {
        pBox.BackColor = Color.Green;
    }
    else
    {
        pBox.BackColor = Color.Transparent;
    }
}

private string GetTimeStamp(bool bSplit)
{
    string sTimeStamp = DateTime.Now.Year.ToString() +
                        DateTime.Now.Month.ToString() +
                        DateTime.Now.Day.ToString() +
                        DateTime.Now.Hour.ToString() +
                        DateTime.Now.Minute.ToString() +
                        DateTime.Now.Second.ToString() +
                        DateTime.Now.Millisecond.ToString();

    if (bSplit)
    {
        sTimeStamp = DateTime.Now.Month.ToString() + "/" +
                      DateTime.Now.Day.ToString() + "/" +
                      DateTime.Now.Year.ToString() + "   " +
                      DateTime.Now.Hour.ToString() + ":" +
                      DateTime.Now.Minute.ToString() + ":" +
                      DateTime.Now.Second.ToString() + "." +
                      DateTime.Now.Millisecond.ToString();
    }

    return sTimeStamp;
}

public void BorgMyMenu(frmMenu FormToBorgify)
{
    foreach (IntPtr pCurrent in Program.Menu_Cache)


```csharp
private frmGrabbedForm = (frmMenu)Form.FromHandle(pCurrent);

    foreach (ToolStripMenuItem tsTop in frmGrabbedForm.MainMenuStrip.Items)
    {
        if (tsTop.Name == "formToolStripMenuItem")
        {
            foreach (ToolStripMenuItem grabbedFormMenuItem in tsTop.DropDownItems)
            {
                foreach (ToolStripMenuItem originalFormToolItem in FormToBorgify.formToolStripMenuItem.DropDownItems)
                {
                    if (grabbedFormMenuItem.Name == originalFormToolItem.Name) {
                        grabbedFormMenuItem.Enabled = originalFormToolItem.Enabled; }
                }
            }
        }
    }

    public bool IsOpen(string Form_Name)
    {
        bool bIsOpen = false;

        foreach (IntPtr pCurrent in Program.Menu_Cache)
        {
            frmMenu frmGrabbedForm = (frmMenu)Form.FromHandle(pCurrent);
            if (frmGrabbedForm.Name == Form_Name)
            {
                bIsOpen = true;
                break;
            }
            frmGrabbedForm = null;
        }

        return bIsOpen;
    }

#region Forms to Open

    public void Open_Packet_Form(frmMenu frmMenu)
    {
        frmPacket Packet = new frmPacket();
        Program.Menu_Cache.Add(Packet.Handle);

        //Now, set all of this new form's menu control settings to be the same as mine.
        Program.UI.BorgMyMenu(frmMenu);
        Packet.Show();
    }

    public void Open_Graph_Form(frmMenu frmMenu)
    {
    }
```
open Config Form(frmMenu frmMenu, string sSelectedTab)
{
    frmConfig Config = new frmConfig();
    Program.Menu_Cache.Add(Config.Handle);

    //Now, set all of this new form's menu control settings to be the same as mine.
    Program.UI.BorgMyMenu(frmMenu);

    switch (sSelectedTab)
    {
        case "tabPolling":
            Config.tcConfig.SelectedTab = Config.tcConfig.SelectedTab = Config.tabPolling;
            break;

        case "tabCOMM":
            Config.tcConfig.SelectedTab = Config.tcConfig.SelectedTab = Config.tabCOMM;
            break;

        default:
            break;
    }

    Config.Show();
}

public void Open_Drive_Form(frmMenu frmMenu)
{
    frmDrive Drive = new frmDrive();
    Program.Menu_Cache.Add(Drive.Handle);

    //Now, set all of this new form's menu control settings to be the same as mine.
    Program.UI.BorgMyMenu(frmMenu);
    Drive.Show();
}

public void Open_Sensors_Form(frmMenu frmMenu)
{
    frmSensors Sensors = new frmSensors();
    Program.Menu_Cache.Add(Sensors.Handle);

    //Now, set all of this new form's menu control settings to be the same as mine.
    Program.UI.BorgMyMenu(frmMenu);
    Sensors.Show();
}

public void Open_Command_Form(frmMenu frmMenu)
{
frmCommand Command = new frmCommand();
Program.Menu_Cache.Add(Command.Handle);

//Now, set all of this new form's menu control settings to be the same as mine.
Program.UI.BorgMyMenu(frmMenu);
Command.Show();
}
public void Open_Macro_Form(frmMenu frmMenu)
{
    frmMacro Macro = new frmMacro();
    Program.Menu_Cache.Add(Macro.Handle);

    //Now, set all of this new form's menu control settings to be the same as mine.
    Program.UI.BorgMyMenu(frmMenu);
    Macro.Show();
}

#endregion

public void Check_Battery(Sensors sSensorPollToCheck)
{
}

public List<string> GetPorts()
{
    List<string> lAvailablePorts = new List<string>();
    string sPortsAvailable = "";

    try
    {
        lAvailablePorts.AddRange(SerialPort.GetPortNames());
        foreach (string sPort in lAvailablePorts)
        {
            sPortsAvailable += " " + sPort;
        }
    }
    catch
    {
        
    }

    return lAvailablePorts;
}

public void Suspend_Comm(bool Suspend)
{
    this.Suspended = Suspend;

    if (!this.Suspended)
{  
    this.Open_For_Restart = !Suspend;
}

public static void ShowFile(string sFilePath)
{
    bool bFileExists = File.Exists(sFilePath);

    if (bFileExists)
    {
        try
        {
            proc.EnableRaisingEvents = false;
            proc.StartInfo.FileName = sFilePath;
            proc.Start();
        }
        catch (Exception ex)
        {
            throw ex;
        }
    }
}
Appendix C: A Simple Agent

A simple agent as defined by Russell and Norvig (9) can be an object such as a clock. As agents, most clocks perform the right action regardless of outside events. The clocks have no percepts to adjust their functionality. This might be needed to adjust the time should the clock traverse a time zone. A cellular phone with the clock is an example of a clock with a percept. As the cell phone crosses the time zone, it receives a signal from the cell tower to adjust to the new time automatically. The cell phone might be considered an intelligent agent according to this definition.

The percepts that are received could be mapped to an action, but this might lead to an extensive list depending on the number of percept combinations. Having an autonomous agent or one that can react to experiences and built in knowledge and then adjust accordingly would make this a better agent. The agent is the combination of the architecture and program. The architecture would be the Roomba in this research as it makes the percepts from the sensors available to the program which will make decisions or react accordingly. The program is simply the implementation in C# which will help control the agent in conjunction with the architecture.

During the implementation of the research the Roomba progressed from a simple reflex agent which only reacted to the percepts to a goal based agent. For example, when a bumper was hit the Roomba simply stopped. This represented an action mapped to a percept. The goal based agent was tested in track 3 and 4. In these experiments the agent was given a goal. The virtual wall was used as the goal or stopping point. The Roombas traversed the test tracks until they found the virtual wall.
List of References


2. —. *iRobot. [Online]* iRobot, October 1, 2005. [Cited: April 17, 2007.]


   http://compreviews.about.com/od/glossary/g/SerialPort.htm.


6. **Tribelhorn, Ben and Dodds, Zachary.** Evaluating the Roomba: A low-cost, ubiquitous.
   Claremont, California, USA : s.n., 2006.

   [Online] Robo Dynamics, 1 1, 2006. [Cited: June 1, 2006.]


VITA

Donald Samuel McCune was born October 20, 1970 in Anderson, Indiana. He graduated from Amherst County High School in 1989. He joined the U.S. Army in 1991 as a Bradley Fighting Vehicle Mechanic. He was assigned to 4/7th Infantry in Aschaffenburg, Germany. After the 4/7th Infantry deactivated, he was reassigned to 4/3 Air Defense Artillery in Kitzengen, Germany before being selected for the Army Green to Gold program. He attended Virginia State University from 1994 to 1998 and graduated summa cum laude with a bachelors of Science in Electrical Engineering Technology. During college he also completed both the U.S. Army Airborne and Air Assault courses, and participated in Ranger Challenge for four consecutive years. He graduated as a Distinguished Military Graduate and was commissioned as a 2LT in 1998 into the Signal Corps. He finished the officer basic course in 1998 and was assigned as the Commo platoon leader / Battalion S-6 in 2/8th Infantry Battalion. He graduated the Signal Captains Career Course and CAS3 in 2001 and assigned as the Battalion S-1 in 307th Signal Battalion in Waegwan, South Korea. He next commanded Headquarters and Headquarters Company 307th Signal Battalion. He was then selected to command a 2nd company, the 229th, the only tactical satellite company of its kind in the U.S. Army. His following assignment was as Communications officer / Deputy J-6 for SOCKOR located in Seoul, South Korea. He was selected to teach Computer Science at West Point Military Academy and recently completed his Masters of Science in Computer Science at Virginia Commonwealth University in Richmond, Virginia.