RFID Wireless Employee Access System

COMPUTER ENGINEERING SENIOR PROJECT
UNIVERSITY OF UTAH
FALL 2008

Final Report

by
Fetah Basic
Kenneth Dean

– Table of Contents –

High Level Description

RFID Wireless Employee Access System is an application that utilizes RF readers along with RF active tags in tracking and logging of hourly employees. In our RDID WEA system we accomplish this by using two RFID readers one on each side of the door. They are always reading the active RFID tags and as the tags move to, from and between the readers we determine from the strength of the signal the readers are receiving what the system should do. Lock, unlock the door and clock in or clock out the employee. Along with this we control a mechanical locking mechanism (lock on the door) via wireless commands to allow employees access to the work site. Using a Micro controller placed near the door lock we control a solenoid that can push or pull the lock lever. We also have installed a magnetic sensor that senses if the door is open or closed besides from just knowing when the door is open or closed we use this knowledge to lock the door as soon as it is open and not to unlock the door redundantly if it is already open. Along side this we have connected two LCD displays one on each side of the door to communicate and clearly display the actions being taken by the RFID WEA system. So now imagine carrying a small key fob tag on you and as you walk up to the entrance of your workplace the door unlocks automatically and as you walk in you are also clocked in automatically. And the same occurs as you leave your work place. On top of which everything is being clearly displayed on two LCD displays and at the same time everything is being tracked on the back end application that is controlling the entire system.

This would be used to substitute manual clock punching in and out. Employees will have small RF tags that will be read by RF readers and perform the hour logging automatically. The tags will also allow employees access into the work place or specific area inside the workplace by reading the tags at the entrance and sending commands to the locking mechanism. As

employees approach a door the reader will read the tag access a database to verify authorized entry to that particular door. Upon authorization the door will be unlocked to allow access. This of course can be logged as well so the employer keeps track of who is entering or leaving the workplace and the time that the transaction occurs.

User Interface Description

There are two user interface categories in the RFID WEA system administrative managerial interface someone who has setup the system and wants to monitor it. And then there is the interface between the person carrying the tag and the RFID WEA system. Below is a typical walk through of what happens when a person approaches the door and ultimately clocks in. Walk through will show both interfaces.

Initial App Setup & Configuration

An administrator first configures the parameters of the system (see below Figure 1). This includes configuring hardware setup such as what reader is hooked up to what serial port. But it also includes system parameters such as when or at what signal strength threshold should the system recognize a tag as a candidate for a lock/unlock transaction. And after the signal strength threshold is met for how long should it be met, how many receive samples? These are some of the configurations available to the administrator of the system. Now the administrator can start the main application (Central Processing System see Figure 2) by hitting the Start button.

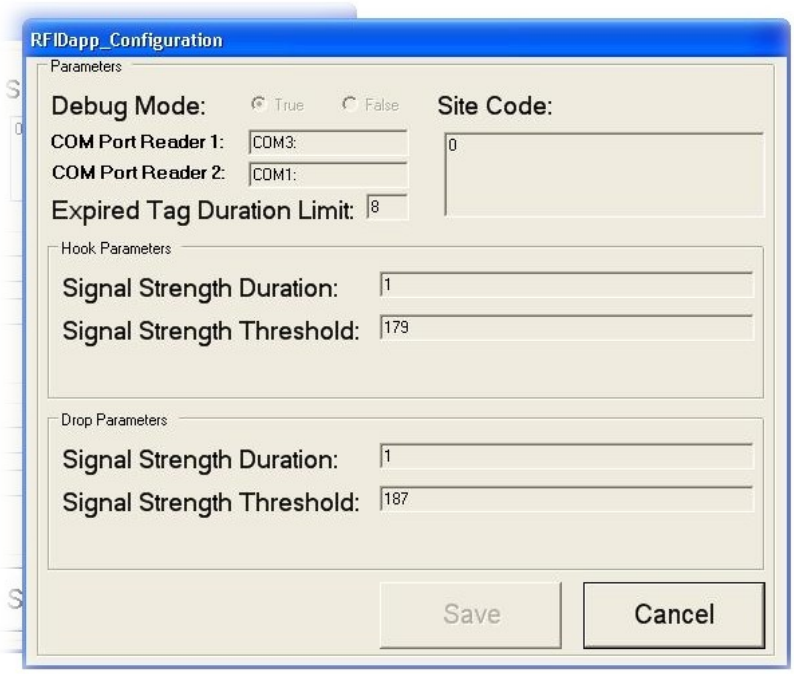


Figure 1. Configuration Screen

Unlock Transaction

Now that the main application is started the system can be used for its purpose. An employee approaches the entrance door as they come in range of the first reader and they meet the criteria of signal strength and duration, the system will authorize that tag/employee - if they exist in the system database - access into the building by unlocking the door. It will also print a message indicating that the action has taken place. As well all of this will be tracked and visible to the administrator on the back end, the main GUI (Figure 2).

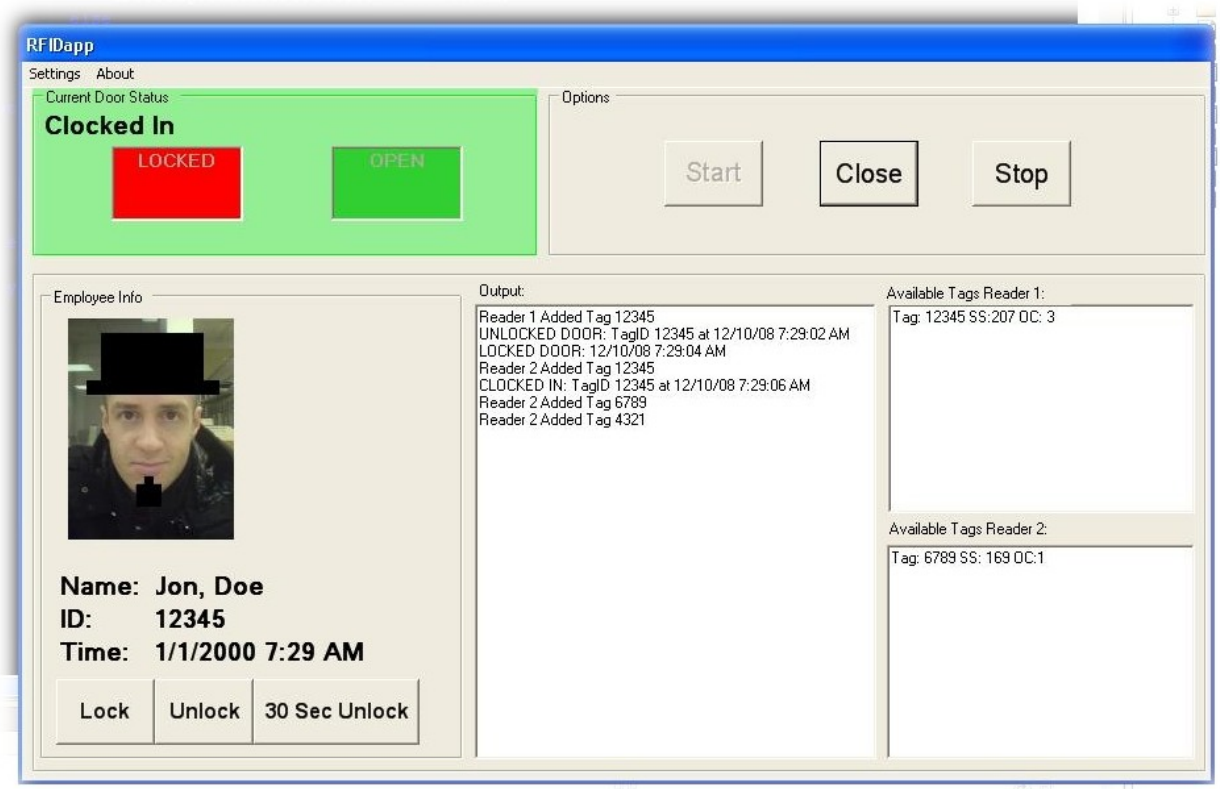


Figure 2. Main Application GUI

Clock In Transaction

Once the employee has been granted access into the building by unlocking the door now as he enters the building on the other side the second reader will pick up the tag similarly like the first outside reader did. Again if the criteria of signal strength and duration have been met the system will Clock In the person with that particular tag ID. As you can see in our figure 2 illustration of the Central Processor's GUI the employee Jon Doe has been marked as Clocked In by our system and the administrator could see that.

Similar transactions take place when the person is leaving the work places except in the reverse order.

Design Description

In this section we broke up the design into sub-sections of the major components of the over all system. Design of each major component will be described in a top-down fashion. Any schematics, pictures, source code, etc will be referenced as necessary to help clarify things. Source code and diagrams are in the various appendices at the end of this document (also see Table of Contents)

RFID Readers / Tags

This hardware component was used as the basis for our system. The RFID readers received the data from the RFID tags, which gets fed to our Central Processing system. RFID readers used were two L-Series RX-300 WaveTrend (Figure 3). Plugs into any serial port it has a female DB9 connector use RS232 protocol with 57600 baud rate on outgoing data. No external power was needed all the power is provided by the PC communication port. The antenna is internal. We plug in two of these readers into two separate serial ports. Communication via the serial port was setup with following parameters:

-Baudrate: 57600
-ByteSize: 8
-Parity: None
-StopBits: 1
-Binary: True
-DtrControl: Enable
-RtsControl: Enable

Figure 3. L-Series RX-300 RFID Reader

RFID tags used were compatible L-Series TG800-IH WaveTrend (Figure 4). Internal battery powers these active tags. Which have a long life expectancy of around 7 years. They transmit a periodic Radio Frequency signal with encoded information. The transmitted Tag data includes Customer Site Code (CSC), Tag ID, RSSI (Signal Strength), Tag Age Counter Value, Movement Alarm and Tamper Alarm status. The readers receive this data. The tags can be factory programmed or programmed with the help of a special tool to transmit at a certain interval or to have certain identification, etc. Life expectancy of the battery depends on the frequency of the tag transmittals. Our tags are transmitting once every 1.5 seconds.

Figure 4. L-Series TG800-IH RFID Tag

Tag data is continuously received by the reader in raw binary format. The readers are setup in auto-polling mode so we continuously receive response packet data (Figure 5). Total packet is 39 byte 7 of which is part of the header and 32 part of the data payload. It is the 32 bytes of Data that holds the information we needed. Figure 6 shows the breakdown of the 32 bytes of Data. Component section Central Processor will further describe how this data was obtained and used.

Response (Tag Packet)

0x55	Data Length	Network ID	Receiver ID	Node ID	0x06	Data	Checksum

Figure 5. Get Tag Response Packet

Byte	Function / Value
1	!
2	*
3	*
4	Interval
5	Reed Switch Counter
6	Firmware version
7	B
8	C
9	Movement switch counter
10	Age byte MSB
11	Age byte
12	Age byte
13	Age byte LSB
14	Site code MSB
15	Site code
16	Site code LSB
17	Tag ID MSB
18	Tag ID
19	Tag ID
20	Tag ID LSB
21	Type of tag flag
22	Reader ID
23	RSSI signal strength
24	Checksum
25	ZUM (reserved)
26	Alarm byte
27	Node ID
28	Network ID
29	Reader Set RSSI Value
30	Firmware Version
31	LF
32	CR

Figure 6. Tag Response Packet Data Field

Zigbee USB Wireless Transmitter

Zigbee USB Wireless transmitter was used to communicate wirelessly with our Micro controller, which was part of our door lock device. It used ZigBee protocol based on the IEEE 802.15.4 standard for wireless personal area networks.

Communication with the Zigbee device was done using serial communication. It was used as a communication bridge between our Central Process and the Micro controller. We used it to send commands and receive data so to control the actions of the Micro controller and the Door Lock Device itself. The list of functions or commands that we built in the Micro controller to support are listed below:

The micro controller for the door has the following functionality, shown in Table 1, with each function called by sending a control byte to the required com port. Control codes 0 through 5 all expect four lines of twenty bytes to be sent after the control code. These bytes are displayed on the selected LCD screen. Failure to provide these bytes may result in undefined behavior.

Table 1. Micro controller command codes.

Control code:	**Function:**
0:*	**Opens the door and starts a count down timer is started and the door is relocked at the completion of the countdown. The LCD displays are cleared and turned off. Messages are displayed on both LCD screens.**
1:*	**Opens the door and displays a message on LCD unit 1.**
2:*	**Opens the door and displays a message on LCD unit 2.**
3:*	**Locks the door, clears both LCD screens, and turns both LCD units off.**
4:*	**Displays user message to LCD unit 1.**
5:*	**Displays user message to LCD unit 2.**
6:	**Returns a byte indicating the status of the door. 0 indicates the door is closed, 1 indicates the door is open.**
7:	**Unlocks the door.**
8:	**Locks the door.**
9:	**Clears the screen of LCD unit 1.**
10:	**Clears the screen of LCD unit 2.**
11:	**Turns off the back light of LCD unit 1.**
12:	**Turns off the back light of LCD unit 2.**
13:	**Turns on the back light of LCD unit 1.**
14:	**Turns on the back light of LCD unit 2.**

* These functions must be supplied four lines of twenty bytes after the control code.

Component section Central Processor will further describe how this was used.

Central Processor

Central Processor component is the one that tied the entire system together. This incorporated the RFID readers and the Zigbee. This is also where all the logic is for the system to do all the transactions, grant access, clock in / clock out, track the transactions, etc. Central Processor is also where the main GUI is (Figure 2).

The Central Processor was done in .NET using C# as a windows application. Done as a multi-class multi-thread application to handle all the parallel tasks of two RFID readers, Zigbee Communications class and main app window. It also uses COM objects to expose undying windows APIs for serial port communications.

The application loops constantly reading tag data on the two RFID readers independently. It processes the data in the local thread and then in critical sections it updates global members which get processed in a global state machine to determine action of the system. Communication with the Micro controller through the Zigbee is done asynchronously from within the Central Processor. Taking a closer look at one thread and one method, receiveFromReader1() see Appendix B, that receives and process data from one RFID reader. It firsts opens the COM port to the reader after which it resets the reader for receiving tag data. In a continuous loop it reads raw binary tag data into a local buffer. It locates the beginning of the data packet by finding the header. The data packet is then parsed out into separate object that get processed locally. Information such as Tag ID, Site Code and RSSI Signal Strength is retrieved. Then checking against set criteria and previous state of received tag data it updates global objects, occurrence

count, and RSSI. Once the criteria have been met the transaction will be initiated. Reading of current state of the door lock device is performed and appropriate action is taken by utilizing the Zigbee communications class. A command to unlock the door is sent along with a message to be written to one of the LCD screens.

The Central Processor is constantly processing on all fronts and updating its' state. The key is multi-threading and safe reliable inter process communication with no inter process dependencies.

.

Microcontroller

The micro controller is a wireless ARMITE from Coridium corporation using an ARM7 CPU running at 60 MHz with 32 K flash and 8 K SRAM memory. There are 24 TTL compatible digital I/O ports, eight 10 bit A/D converters with a 100 KHz sample rate, and 8 hardware PWM channels. One I/O port is used to trigger a solenoid driver to operate the door lock. One port is used to receive the data from the doors' magnetic reed switch. Fourteen port are connected to the LCD screens with eight of these ports used as a data bus connected to both LCD screens. Data read/write, function/display, and back light controls consuming the remaining six ports three to each screen.

Data received from the ZigBee is processed through a serial port and sent to the ARM7 CPU. Software controls extract the command, perform the command function, and send display data to the LCD screens. The micro controller command codes are listed in Table 1. The micro controller source code is list ed in appendix C. The micro controller schematic is shown in figure 7.

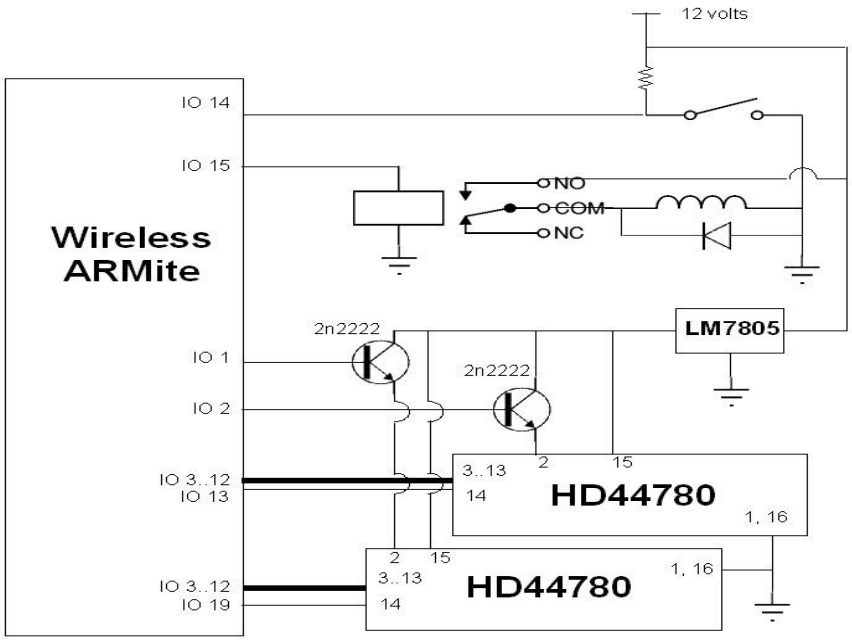


Figure 7. Micro controller schematic.

Door Lock Device

The door lock device consist of a SMT-2551L24AA solenoid connected directly to the door bolt. The bolt is retracted by energizing the solenoid and extended by a spring when the solenoid power is turned off.

Lessons Learned

Upon completion of the project it became obvious that the RFID readers should have been incorporated into the micro controller assembly. This would have made the door completely autonomous and this design would allow for multiple doors to be installed at a single workplace. Our design had the RFID reader connected to the central processor that limited the location of the door next to the central processor or would have required long lengths of wire to place the readers by the door.

Conclusions

All in all, this project went well and it was a success most of all. It was a great experience working as a team and seeing a yearlong project come to fruition. Some things could have gone better but didn't because we did not foresee it either do to poor planning or short sightedness. But that could have been because of the tight schedule and relatively short time we had to complete it all.

Acknowledgments

- We like to thank DriverTech, LLC for providing the RFID equipment and serial cables.
- WaveTrend RFID Reader users manual. Used for configuration and protocol.
- Helpful MSDN Online Library's and the support found for COM APIs and DCB COM port object.
- And finally endless online forums regarding general C# and .NET help.

Appendix A - Bill of Materials

Part	Part #	Supplier	Cost
24VDC Solenoid	SMT-2551L24AA	Jameco	$11.99
Solenoid Driver	DRV101FKTWT	Digi-Key	$7.70
Wireless ARMmite	AM-WL	Coridium Corporation	$39.95
XBee Zigbee wireless	XB24-AWI-001	Digi-Key	$19.00
RFID Reader(2)	(2)L-RX300	Wavetrend	Free
Cables	PC serial port cable	N/A	Free
Cables	USB to Serial Cable	N/A	Free
Tags(2)	(2) L-TG800-IH	Wavetrend	Free
RFID Reader Analyzer Software	N/A	Wavetrend	Free

Appendix B – Central Processor Source (RFID & USB Zigbee)

RFIDMain.cs

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
using System.IO;
using System.Net;
using System.Web;
using System.Text;
using System.Threading;
using System.Runtime.InteropServices;

namespace RFIDapp
{
      /// <summary>
      /// Summary description for Form1.
      /// </summary>
      public class RFIDmain : System.Windows.Forms.Form
    {
        private IContainer components;

            private const int MAXDWORD = unchecked((int)0xffffffff);
            private const uint FILE_FLAG_OVERLAPPED = 0x40000000;
            private const uint FILE_ATTRIBUTE_NORMAL = 0x00000080;
            private const uint GENERIC_WRITE = 0x40000000;
            private const uint GENERIC_READ = 0x80000000;
            private const uint OPEN_EXISTING = 3;
            //private const int SITE_CODE = 4278612;
            private const int CBR_57600 = 57600;
            private const int CBR_115200 = 115200;
            private const byte ONESTOPBIT = 0;
            private const byte PARITY_NONE = 0;
            private const byte RSSI_THRESHOLD = 175;
            //private const byte TAG_REMOVE_TIME = 30;
            public static byte TAG_REMOVE_TIME = 30;
            private const byte OPTION_NORMAL = 0;
            private const byte OPTION_GRACE = 1;
            private const byte ERROR_ALREADY_EXISTS = 183;
        private System.IntPtr handleReader1;
        private System.IntPtr handleReader2;

            private delegate void OutputCallback(string print, bool bDebug);
            private delegate void OutputCallback1();
        private delegate void OutputCallback2();
            private delegate void LabelCallback(byte option);
            private delegate void LabelStrongTagCallback();
        private delegate void LabelPrintCallback(Label label, string print);
```

```
        private delegate void TextBoxPrintCallback(TextBox tb, string print,
Color color);

        public static Hashtable Employees = new Hashtable();
            private ArrayList tagsReader1 = new ArrayList();
        private ArrayList tagsReader2 = new ArrayList();
            private int strongestTagCounter = 0;
            private ManualResetEvent mre = null;
            private Thread threadMonitorTags = null;
        private Thread threadReader1 = null;
        private Thread threadReader2 = null;
        private Thread threadTagLogic = null;

            private Hashtable hsTagReader1 = new Hashtable();
        private Hashtable hsTagReader2 = new Hashtable();

            public static bool bDebug = false;
            public static string ComPortR1 = "COM1:";
        public static string ComPortR2 = "COM2:";
            public static bool bHook = false;
            public static int strongestTag = 0;
            public static short hookWaitTime = 60;
            public static short dropWaitTime = 10;
            public static short ssHookThreshold = 0;
            public static short ssDropThreshold = 0;

            public static string SITE_CODE = "0";
            static uint baudRate = 57600;
        //static byte readerType = 0; // 0=L-SERIES, 1=W-SERIES
            private System.Windows.Forms.Button btnStop;
            private System.Windows.Forms.Button btnClose;
        private System.Windows.Forms.Button btnStart;
            private System.Windows.Forms.RichTextBox txOutput;
        private System.Windows.Forms.Label label1;
            private System.Windows.Forms.Label label5;
            private System.Windows.Forms.Label lbStrongTag;
            private System.Windows.Forms.RichTextBox txOutput1; // default is
57600
            public static string eventTimestamp = "";
            public static string hookTimeStamp = "";
            public static string dropTimeStamp = "";
            public static string logFileName = "RFID_log.txt";

            public static double oH = -1.0;
            private System.Windows.Forms.GroupBox groupBox1;
        private System.Windows.Forms.GroupBox gbProgress;
            private System.Windows.Forms.GroupBox groupBox3;
            private System.Windows.Forms.GroupBox groupBox4;
            private System.Windows.Forms.Label label3;
            private System.Windows.Forms.Label label6;
            private System.Windows.Forms.Label label8;
            private System.Windows.Forms.Label label10;
            private System.Windows.Forms.Label lbState;
            private System.Windows.Forms.Label label12;
            private System.Windows.Forms.MainMenu mainMenu1;
            private System.Windows.Forms.MenuItem menuItem1;
            private System.Windows.Forms.MenuItem menuItemConfig;
```

```
    public static double oW = -1.0;
    private System.Windows.Forms.MenuItem menuItem2;
    private System.Windows.Forms.MenuItem menuItem3;
public static Configuration CconfigForm = null;
private PictureBox pbPicID;
private Label label7;
private Label lbTime;
private Label lbID;
private Label lbName;
private Label label11;
private Label label9;
private Button btnUnlockTimer;
private Button btnUnlock;
private Button btnLock;
private MenuItem menuItem4;
private TextBox txLockStatus;
private Label lbStatus;
private TextBox txDoorStatus;
private RichTextBox txOutput2;
private Label label14;
private Label label2;
private GroupBox gbStatus;
private Label lbStrongCounter;

private ComPort DoorLock = new ComPort();
public static DateTime doorUnlockedAt = new DateTime();

    [DllImport("user32.dll")]
    public static extern int SetWindowText
          (
          IntPtr hwnd,
          string str
          );

    public RFIDmain()
    {
          if (bDebug)
                InitializeComponent();

          SetWindowText(this.Handle, "RFIDapp");
    }

    /// <summary>
    /// Clean up any resources being used.
    /// </summary>
    protected override void Dispose( bool disposing )
    {
          if( disposing )
          {
                if (components != null)
                {
                      components.Dispose();
                }
          }
          base.Dispose( disposing );
    }
```

```
            #region Windows Form Designer generated code
            /// <summary>
            /// Required method for Designer support - do not modify
            /// the contents of this method with the code editor.
            /// </summary>
            private void InitializeComponent()
            {
            this.components = new System.ComponentModel.Container();
            this.btnStop = new System.Windows.Forms.Button();
            this.btnClose = new System.Windows.Forms.Button();
            this.btnStart = new System.Windows.Forms.Button();
            this.txOutput = new System.Windows.Forms.RichTextBox();
            this.label1 = new System.Windows.Forms.Label();
            this.label5 = new System.Windows.Forms.Label();
            this.lbStrongTag = new System.Windows.Forms.Label();
            this.txOutput1 = new System.Windows.Forms.RichTextBox();
            this.groupBox1 = new System.Windows.Forms.GroupBox();
            this.gbProgress = new System.Windows.Forms.GroupBox();
            this.txDoorStatus = new System.Windows.Forms.TextBox();
            this.txLockStatus = new System.Windows.Forms.TextBox();
            this.lbStatus = new System.Windows.Forms.Label();
            this.lbState = new System.Windows.Forms.Label();
            this.groupBox3 = new System.Windows.Forms.GroupBox();
            this.gbStatus = new System.Windows.Forms.GroupBox();
            this.pbPicID = new System.Windows.Forms.PictureBox();
            this.label7 = new System.Windows.Forms.Label();
            this.label9 = new System.Windows.Forms.Label();
            this.label11 = new System.Windows.Forms.Label();
            this.lbName = new System.Windows.Forms.Label();
            this.btnUnlockTimer = new System.Windows.Forms.Button();
            this.lbID = new System.Windows.Forms.Label();
            this.btnUnlock = new System.Windows.Forms.Button();
            this.lbTime = new System.Windows.Forms.Label();
            this.btnLock = new System.Windows.Forms.Button();
            this.lbStrongCounter = new System.Windows.Forms.Label();
            this.label14 = new System.Windows.Forms.Label();
            this.txOutput2 = new System.Windows.Forms.RichTextBox();
            this.label2 = new System.Windows.Forms.Label();
            this.mainMenu1 = new
System.Windows.Forms.MainMenu(this.components);
            this.menuItem1 = new System.Windows.Forms.MenuItem();
            this.menuItemConfig = new System.Windows.Forms.MenuItem();
            this.menuItem4 = new System.Windows.Forms.MenuItem();
            this.menuItem2 = new System.Windows.Forms.MenuItem();
            this.menuItem3 = new System.Windows.Forms.MenuItem();
            this.groupBox1.SuspendLayout();
            this.gbProgress.SuspendLayout();
            this.groupBox3.SuspendLayout();
            this.gbStatus.SuspendLayout();
            ((System.ComponentModel.ISupportInitialize)
(this.pbPicID)).BeginInit();
            this.SuspendLayout();
            //
            // btnStop
            //
```

```
            this.btnStop.Font = new System.Drawing.Font("Microsoft Sans
Serif", 15F, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, ((byte)(0)));
            this.btnStop.Location = new System.Drawing.Point(322, 40);
            this.btnStop.Name = "btnStop";
            this.btnStop.Size = new System.Drawing.Size(75, 50);
            this.btnStop.TabIndex = 5;
            this.btnStop.Text = "Stop";
            this.btnStop.Click += new System.EventHandler(this.btnStop_Click);
            //
            // btnClose
            //
            this.btnClose.DialogResult =
System.Windows.Forms.DialogResult.Cancel;
            this.btnClose.Font = new System.Drawing.Font("Microsoft Sans
Serif", 15F, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, ((byte)(0)));
            this.btnClose.Location = new System.Drawing.Point(206, 40);
            this.btnClose.Name = "btnClose";
            this.btnClose.Size = new System.Drawing.Size(75, 50);
            this.btnClose.TabIndex = 4;
            this.btnClose.Text = "Close";
            this.btnClose.Click += new
System.EventHandler(this.btnClose_Click);
            //
            // btnStart
            //
            this.btnStart.DialogResult =
System.Windows.Forms.DialogResult.Cancel;
            this.btnStart.Font = new System.Drawing.Font("Arial", 15F,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point, ((byte)
(0)));
            this.btnStart.Location = new System.Drawing.Point(88, 40);
            this.btnStart.Name = "btnStart";
            this.btnStart.Size = new System.Drawing.Size(75, 50);
            this.btnStart.TabIndex = 2;
            this.btnStart.Text = "Start";
            this.btnStart.Click += new
System.EventHandler(this.btnStart_Click);
            //
            // txOutput
            //
            this.txOutput.Location = new System.Drawing.Point(336, 28);
            this.txOutput.Name = "txOutput";
            this.txOutput.Size = new System.Drawing.Size(305, 346);
            this.txOutput.TabIndex = 1;
            this.txOutput.Text = "";
            //
            // label1
            //
            this.label1.Location = new System.Drawing.Point(337, 11);
            this.label1.Name = "label1";
            this.label1.Size = new System.Drawing.Size(100, 16);
            this.label1.TabIndex = 5;
            this.label1.Text = "Output:";
            //
            // label5
```

```
            //
            this.label5.Enabled = false;
            this.label5.Font = new System.Drawing.Font("Microsoft Sans
Serif", 14F, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, ((byte)(0)));
            this.label5.ForeColor = System.Drawing.Color.MediumBlue;
            this.label5.Location = new System.Drawing.Point(393, 77);
            this.label5.Name = "label5";
            this.label5.Size = new System.Drawing.Size(136, 24);
            this.label5.TabIndex = 9;
            this.label5.Text = "Strongest Tag:";
            this.label5.Visible = false;
            //
            // lbStrongTag
            //
            this.lbStrongTag.Enabled = false;
            this.lbStrongTag.Font = new System.Drawing.Font("Microsoft Sans
Serif", 14F, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, ((byte)(0)));
            this.lbStrongTag.ForeColor = System.Drawing.Color.MediumBlue;
            this.lbStrongTag.Location = new System.Drawing.Point(529, 77);
            this.lbStrongTag.Name = "lbStrongTag";
            this.lbStrongTag.Size = new System.Drawing.Size(144, 24);
            this.lbStrongTag.TabIndex = 10;
            this.lbStrongTag.Text = "00000000";
            this.lbStrongTag.Visible = false;
            //
            // txOutput1
            //
            this.txOutput1.Location = new System.Drawing.Point(650, 28);
            this.txOutput1.Name = "txOutput1";
            this.txOutput1.Size = new System.Drawing.Size(233, 159);
            this.txOutput1.TabIndex = 3;
            this.txOutput1.Text = "";
            //
            // groupBox1
            //
            this.groupBox1.Controls.Add(this.btnStop);
            this.groupBox1.Controls.Add(this.btnClose);
            this.groupBox1.Controls.Add(this.btnStart);
            this.groupBox1.Location = new System.Drawing.Point(400, 0);
            this.groupBox1.Name = "groupBox1";
            this.groupBox1.Size = new System.Drawing.Size(500, 128);
            this.groupBox1.TabIndex = 11;
            this.groupBox1.TabStop = false;
            this.groupBox1.Text = "Options";
            //
            // gbProgress
            //
            this.gbProgress.Controls.Add(this.txDoorStatus);
            this.gbProgress.Controls.Add(this.txLockStatus);
            this.gbProgress.Controls.Add(this.lbStatus);
            this.gbProgress.Location = new System.Drawing.Point(8, 0);
            this.gbProgress.Name = "gbProgress";
            this.gbProgress.Size = new System.Drawing.Size(384, 128);
            this.gbProgress.TabIndex = 12;
            this.gbProgress.TabStop = false;
```

```
            this.gbProgress.Text = "Current Door Status";
            //
            // txDoorStatus
            //
            this.txDoorStatus.BackColor = System.Drawing.Color.Red;
            this.txDoorStatus.Enabled = false;
            this.txDoorStatus.Font = new System.Drawing.Font("Microsoft Sans
Serif", 9.75F, System.Drawing.FontStyle.Bold,
System.Drawing.GraphicsUnit.Point, ((byte)(0)));
            this.txDoorStatus.ForeColor = System.Drawing.SystemColors.Window;
            this.txDoorStatus.Location = new System.Drawing.Point(227, 44);
            this.txDoorStatus.Multiline = true;
            this.txDoorStatus.Name = "txDoorStatus";
            this.txDoorStatus.Size = new System.Drawing.Size(100, 57);
            this.txDoorStatus.TabIndex = 21;
            this.txDoorStatus.Text = "CLOSED";
            this.txDoorStatus.TextAlign =
System.Windows.Forms.HorizontalAlignment.Center;
            //
            // txLockStatus
            //
            this.txLockStatus.BackColor = System.Drawing.Color.Red;
            this.txLockStatus.Enabled = false;
            this.txLockStatus.Font = new System.Drawing.Font("Microsoft Sans
Serif", 9.75F, System.Drawing.FontStyle.Bold,
System.Drawing.GraphicsUnit.Point, ((byte)(0)));
            this.txLockStatus.ForeColor = System.Drawing.SystemColors.Window;
            this.txLockStatus.Location = new System.Drawing.Point(60, 44);
            this.txLockStatus.Multiline = true;
            this.txLockStatus.Name = "txLockStatus";
            this.txLockStatus.Size = new System.Drawing.Size(100, 57);
            this.txLockStatus.TabIndex = 20;
            this.txLockStatus.Text = "LOCKED";
            this.txLockStatus.TextAlign =
System.Windows.Forms.HorizontalAlignment.Center;
            //
            // lbStatus
            //
            this.lbStatus.Font = new System.Drawing.Font("Microsoft Sans
Serif", 14.25F, System.Drawing.FontStyle.Bold,
System.Drawing.GraphicsUnit.Point, ((byte)(0)));
            this.lbStatus.Location = new System.Drawing.Point(6, 16);
            this.lbStatus.Name = "lbStatus";
            this.lbStatus.Size = new System.Drawing.Size(128, 25);
            this.lbStatus.TabIndex = 19;
            this.lbStatus.Text = "Door Status";
            //
            // lbState
            //
            this.lbState.Enabled = false;
            this.lbState.Font = new System.Drawing.Font("Microsoft Sans
Serif", 14F, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, ((byte)(0)));
            this.lbState.ForeColor = System.Drawing.Color.Black;
            this.lbState.Location = new System.Drawing.Point(321, 157);
            this.lbState.Name = "lbState";
            this.lbState.Size = new System.Drawing.Size(368, 24);
```

```
            this.lbState.TabIndex = 12;
            this.lbState.Text = "Un-Hooked";
            this.lbState.Visible = false;
            //
            // groupBox3
            //
            this.groupBox3.Controls.Add(this.lbState);
            this.groupBox3.Controls.Add(this.gbStatus);
            this.groupBox3.Controls.Add(this.lbStrongCounter);
            this.groupBox3.Controls.Add(this.label14);
            this.groupBox3.Controls.Add(this.lbStrongTag);
            this.groupBox3.Controls.Add(this.txOutput2);
            this.groupBox3.Controls.Add(this.label5);
            this.groupBox3.Controls.Add(this.label1);
            this.groupBox3.Controls.Add(this.txOutput);
            this.groupBox3.Controls.Add(this.label2);
            this.groupBox3.Controls.Add(this.txOutput1);
            this.groupBox3.Location = new System.Drawing.Point(8, 136);
            this.groupBox3.Name = "groupBox3";
            this.groupBox3.Size = new System.Drawing.Size(892, 384);
            this.groupBox3.TabIndex = 13;
            this.groupBox3.TabStop = false;
            //
            // gbStatus
            //
            this.gbStatus.Controls.Add(this.pbPicID);
            this.gbStatus.Controls.Add(this.label7);
            this.gbStatus.Controls.Add(this.label9);
            this.gbStatus.Controls.Add(this.label11);
            this.gbStatus.Controls.Add(this.lbName);
            this.gbStatus.Controls.Add(this.btnUnlockTimer);
            this.gbStatus.Controls.Add(this.lbID);
            this.gbStatus.Controls.Add(this.btnUnlock);
            this.gbStatus.Controls.Add(this.lbTime);
            this.gbStatus.Controls.Add(this.btnLock);
            this.gbStatus.Location = new System.Drawing.Point(6, 16);
            this.gbStatus.Name = "gbStatus";
            this.gbStatus.Size = new System.Drawing.Size(321, 358);
            this.gbStatus.TabIndex = 24;
            this.gbStatus.TabStop = false;
            this.gbStatus.Text = "Employee Info";
            //
            // pbPicID
            //
            this.pbPicID.Location = new System.Drawing.Point(15, 23);
            this.pbPicID.Name = "pbPicID";
            this.pbPicID.Size = new System.Drawing.Size(139, 169);
            this.pbPicID.SizeMode =
System.Windows.Forms.PictureBoxSizeMode.Zoom;
            this.pbPicID.TabIndex = 9;
            this.pbPicID.TabStop = false;
            //
            // label7
            //
            this.label7.Font = new System.Drawing.Font("Microsoft Sans
Serif", 14.25F, System.Drawing.FontStyle.Bold,
System.Drawing.GraphicsUnit.Point, ((byte)(0)));
```

```
            this.label7.Location = new System.Drawing.Point(12, 214);
            this.label7.Name = "label7";
            this.label7.Size = new System.Drawing.Size(76, 25);
            this.label7.TabIndex = 11;
            this.label7.Text = "Name:";
            //
            // label9
            //
            this.label9.Font = new System.Drawing.Font("Microsoft Sans
Serif", 14.25F, System.Drawing.FontStyle.Bold,
System.Drawing.GraphicsUnit.Point, ((byte)(0)));
            this.label9.Location = new System.Drawing.Point(12, 239);
            this.label9.Name = "label9";
            this.label9.Size = new System.Drawing.Size(76, 25);
            this.label9.TabIndex = 12;
            this.label9.Text = "ID:";
            //
            // label11
            //
            this.label11.Font = new System.Drawing.Font("Microsoft Sans
Serif", 14.25F, System.Drawing.FontStyle.Bold,
System.Drawing.GraphicsUnit.Point, ((byte)(0)));
            this.label11.Location = new System.Drawing.Point(12, 264);
            this.label11.Name = "label11";
            this.label11.Size = new System.Drawing.Size(76, 25);
            this.label11.TabIndex = 13;
            this.label11.Text = "Time:";
            //
            // lbName
            //
            this.lbName.Font = new System.Drawing.Font("Microsoft Sans
Serif", 14.25F, System.Drawing.FontStyle.Bold,
System.Drawing.GraphicsUnit.Point, ((byte)(0)));
            this.lbName.Location = new System.Drawing.Point(83, 214);
            this.lbName.Name = "lbName";
            this.lbName.Size = new System.Drawing.Size(228, 25);
            this.lbName.TabIndex = 14;
            this.lbName.Text = "Jon, Doe";
            //
            // btnUnlockTimer
            //
            this.btnUnlockTimer.DialogResult =
System.Windows.Forms.DialogResult.Cancel;
            this.btnUnlockTimer.Font = new System.Drawing.Font("Arial", 12F,
System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point, ((byte)
(0)));
            this.btnUnlockTimer.Location = new System.Drawing.Point(162, 297);
            this.btnUnlockTimer.Name = "btnUnlockTimer";
            this.btnUnlockTimer.Size = new System.Drawing.Size(126, 50);
            this.btnUnlockTimer.TabIndex = 18;
            this.btnUnlockTimer.Text = "30 Sec Unlock";
            this.btnUnlockTimer.Click += new
System.EventHandler(this.btnUnlockTimer_Click);
            //
            // lbID
            //
```

```
            this.lbID.Font = new System.Drawing.Font("Microsoft Sans Serif",
14.25F, System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point,
((byte)(0)));
            this.lbID.Location = new System.Drawing.Point(83, 239);
            this.lbID.Name = "lbID";
            this.lbID.Size = new System.Drawing.Size(228, 25);
            this.lbID.TabIndex = 15;
            this.lbID.Text = "12345";
            //
            // btnUnlock
            //
            this.btnUnlock.DialogResult =
System.Windows.Forms.DialogResult.Cancel;
            this.btnUnlock.Font = new System.Drawing.Font("Arial", 12F,
System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point, ((byte)
(0)));
            this.btnUnlock.Location = new System.Drawing.Point(87, 297);
            this.btnUnlock.Name = "btnUnlock";
            this.btnUnlock.Size = new System.Drawing.Size(75, 50);
            this.btnUnlock.TabIndex = 17;
            this.btnUnlock.Text = "Unlock";
            this.btnUnlock.Click += new
System.EventHandler(this.button2_Click);
            //
            // lbTime
            //
            this.lbTime.Font = new System.Drawing.Font("Microsoft Sans
Serif", 14.25F, System.Drawing.FontStyle.Bold,
System.Drawing.GraphicsUnit.Point, ((byte)(0)));
            this.lbTime.Location = new System.Drawing.Point(83, 264);
            this.lbTime.Name = "lbTime";
            this.lbTime.Size = new System.Drawing.Size(228, 25);
            this.lbTime.TabIndex = 16;
            this.lbTime.Text = "1/1/2000";
            //
            // btnLock
            //
            this.btnLock.DialogResult =
System.Windows.Forms.DialogResult.Cancel;
            this.btnLock.Font = new System.Drawing.Font("Arial", 12F,
System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point, ((byte)
(0)));
            this.btnLock.Location = new System.Drawing.Point(12, 297);
            this.btnLock.Name = "btnLock";
            this.btnLock.Size = new System.Drawing.Size(75, 50);
            this.btnLock.TabIndex = 6;
            this.btnLock.Text = "Lock";
            this.btnLock.Click += new System.EventHandler(this.button1_Click);
            //
            // lbStrongCounter
            //
            this.lbStrongCounter.Enabled = false;
            this.lbStrongCounter.Font = new System.Drawing.Font("Microsoft
Sans Serif", 28F, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, ((byte)(0)));
            this.lbStrongCounter.ForeColor = System.Drawing.Color.DarkGreen;
```

```
            this.lbStrongCounter.Location = new System.Drawing.Point(449,
109);
            this.lbStrongCounter.Name = "lbStrongCounter";
            this.lbStrongCounter.Size = new System.Drawing.Size(120, 40);
            this.lbStrongCounter.TabIndex = 6;
            this.lbStrongCounter.Text = "00:00";
            this.lbStrongCounter.Visible = false;
            //
            // label14
            //
            this.label14.Location = new System.Drawing.Point(649, 192);
            this.label14.Name = "label14";
            this.label14.Size = new System.Drawing.Size(140, 16);
            this.label14.TabIndex = 23;
            this.label14.Text = "Available Tags Reader 2:";
            //
            // txOutput2
            //
            this.txOutput2.Location = new System.Drawing.Point(649, 211);
            this.txOutput2.Name = "txOutput2";
            this.txOutput2.Size = new System.Drawing.Size(234, 163);
            this.txOutput2.TabIndex = 22;
            this.txOutput2.Text = "";
            //
            // label2
            //
            this.label2.Location = new System.Drawing.Point(647, 13);
            this.label2.Name = "label2";
            this.label2.Size = new System.Drawing.Size(142, 16);
            this.label2.TabIndex = 8;
            this.label2.Text = "Available Tags Reader 1:";
            //
            // mainMenu1
            //
            this.mainMenu1.MenuItems.AddRange(new
System.Windows.Forms.MenuItem[] {
            this.menuItem1,
            this.menuItem2});
            //
            // menuItem1
            //
            this.menuItem1.Index = 0;
            this.menuItem1.MenuItems.AddRange(new
System.Windows.Forms.MenuItem[] {
            this.menuItemConfig,
            this.menuItem4});
            this.menuItem1.Text = "Settings";
            //
            // menuItemConfig
            //
            this.menuItemConfig.Index = 0;
            this.menuItemConfig.Text = "Configurations";
            this.menuItemConfig.Click += new
System.EventHandler(this.menuItem2_Click);
            //
            // menuItem4
            //
```

```
            this.menuItem4.Index = 1;
            this.menuItem4.Text = "Managment";
            //
            // menuItem2
            //
            this.menuItem2.Index = 1;
            this.menuItem2.MenuItems.AddRange(new
System.Windows.Forms.MenuItem[] {
            this.menuItem3});
            this.menuItem2.Text = "About";
            //
            // menuItem3
            //
            this.menuItem3.Index = 0;
            this.menuItem3.Text = "Version 1.2";
            //
            // RFIDmain
            //
            this.AutoScaleBaseSize = new System.Drawing.Size(5, 13);
            this.CancelButton = this.btnClose;
            this.ClientSize = new System.Drawing.Size(903, 526);
            this.ControlBox = false;
            this.Controls.Add(this.groupBox3);
            this.Controls.Add(this.gbProgress);
            this.Controls.Add(this.groupBox1);
            this.FormBorderStyle =
System.Windows.Forms.FormBorderStyle.FixedSingle;
            this.Menu = this.mainMenu1;
            this.Name = "RFIDmain";
            this.Text = "RFIDapp";
            this.groupBox1.ResumeLayout(false);
            this.gbProgress.ResumeLayout(false);
            this.gbProgress.PerformLayout();
            this.groupBox3.ResumeLayout(false);
            this.gbStatus.ResumeLayout(false);
            ((System.ComponentModel.ISupportInitialize)
(this.pbPicID)).EndInit();
            this.ResumeLayout(false);

            }
            #endregion

            /// <summary>
            /// The main entry point for the application.
            /// </summary>
            [STAThread]
            static void Main()
            {
                  try
                  {
                        CconfigForm = new Configuration();

                        //Get Configuration settings for the app from XML
config file
                        GetConfigurations();

                //Load list of active employees
```

```
                LoadEmployees();

                        IntPtr hInstance =
SerialComClass.CreateEvent(IntPtr.Zero, false, false,
"RFIDapp_INSTANCE_RUNNING");

                        if (Marshal.GetLastWin32Error() ==
ERROR_ALREADY_EXISTS)
                        {
                              SerialComClass.CloseHandle(hInstance);
                              hInstance = IntPtr.Zero;
                              if (RFIDmain.bDebug)
                                    MessageBox.Show("Instance already
exists!", "Error");
                              else
                              {
                                    StreamWriter sw =
File.AppendText(Application.StartupPath + "\\" + RFIDmain.logFileName);
                                    sw.WriteLine(DateTime.UtcNow.ToString() +
"\t" + "Main(): Error(s) Occured: " + "Instance already exists!" + "\r");
                                    sw.Flush();
                                    sw.Close();
                              }
                              return;
                        }

                        RFIDmain.bHook = false;
                        RFIDmain.strongestTag = 0;
                        StreamReader sr = null;
                        if (File.Exists(Application.StartupPath +
"\\LastState.txt"))
                        {
                              // Read the Last State file to set the initial
state of app
                              sr = File.OpenText(Application.StartupPath +
"\\LastState.txt");
                              if (sr.ReadLine().Split('=')
[1].ToUpper().Equals("HOOK"))
                                    RFIDmain.bHook = true;
                              else
                                    RFIDmain.bHook = false;

                              try{RFIDmain.strongestTag =
int.Parse(sr.ReadLine().Split('=')[1].ToUpper());}
                              catch {RFIDmain.strongestTag = 0;}
                              sr.Close();
                        }

                        if (RFIDmain.bDebug)
                              Application.Run(new RFIDmain());
                        else
                        {
                              RFIDmain rfidClass = new RFIDmain();
                              rfidClass.EntryPoint();
                        }
                  }
                  catch (Exception exc)
```

```
                {
                        StreamWriter sw =
File.AppendText(Application.StartupPath + "\\" + RFIDmain.logFileName);
                        sw.WriteLine(DateTime.UtcNow.ToString() + "\t" +
"Main(): Error(s) Occured: " + exc.Message + ", Last Error: " +
Marshal.GetLastWin32Error().ToString() + "\r");
                        sw.Flush();
                        sw.Close();
                }
            }

            public static void GetConfigurations()
            {
                  try
                  {
                // Read the Config XML for initial settings of the app
                string debug_mode = "FALSE", com_portR1 = "COM1:", com_portR2
= "COM2:", site_code = "0", expired_tag_duration = "30";
                Settings.GetProgramSettings(ref debug_mode, ref com_portR1,
ref com_portR2, ref site_code, ref expired_tag_duration);

                if (debug_mode.Trim().ToUpper().Equals("TRUE"))
                    RFIDmain.bDebug = true;
                else
                    RFIDmain.bDebug = false;

                RFIDmain.ComPortR1 = com_portR1.Trim().ToUpper();
                RFIDmain.ComPortR2 = com_portR2.Trim().ToUpper();
                        try{RFIDmain.SITE_CODE = site_code.Trim();}
                        catch {RFIDmain.SITE_CODE = "0";}
                        try{RFIDmain.TAG_REMOVE_TIME =
byte.Parse(expired_tag_duration.Trim());}
                        catch {RFIDmain.TAG_REMOVE_TIME = 30;}

                        // Read the Config XML for Hook Parameters
                        string hook_ss_duration = "60", hook_ss_threshold =
"0";
                        Settings.GetHookParams(ref hook_ss_duration, ref
hook_ss_threshold);

                        try{RFIDmain.hookWaitTime =
short.Parse(hook_ss_duration.Trim());}
                        catch {RFIDmain.hookWaitTime = 60;}
                        try{RFIDmain.ssHookThreshold =
short.Parse(hook_ss_threshold.Trim());}
                        catch {RFIDmain.ssHookThreshold = 0;}

                        // Read the Config XML for Drop Parameters
                        string drop_ss_duration = "60", drop_ss_threshold =
"0";
                        Settings.GetDropParams(ref drop_ss_duration, ref
drop_ss_threshold);

                        try{RFIDmain.dropWaitTime =
short.Parse(drop_ss_duration.Trim());}
                        catch {RFIDmain.dropWaitTime = 60;}
```

```
                        try{RFIDmain.ssDropThreshold =
short.Parse(drop_ss_threshold.Trim());}
                        catch {RFIDmain.ssDropThreshold = 0;}
                    }
                    catch (Exception exc)
                    {
                        StreamWriter sw =
File.AppendText(Application.StartupPath + "\\" + RFIDmain.logFileName);
                        sw.WriteLine(DateTime.UtcNow.ToString() + "\t" +
"GetConfigurations(): Error(s) Occured: " + exc.Message + ", Last Error: " +
Marshal.GetLastWin32Error().ToString() + "\r");
                        sw.Flush();
                        sw.Close();
                    }
            }

        public static void LoadEmployees()
        {
            try
            {
                // Read the Employess info from XML for initial settings of
the app
                Settings.GetEmployeeInfo(ref RFIDmain.Employees);
            }
            catch (Exception exc)
            {
                StreamWriter sw = File.AppendText(Application.StartupPath +
"\\" + RFIDmain.logFileName);
                sw.WriteLine(DateTime.UtcNow.ToString() + "\t" +
"LoadEmployees(): Error(s) Occured: " + exc.Message + ", Last Error: " +
Marshal.GetLastWin32Error().ToString() + "\r");
                sw.Flush();
                sw.Close();
            }
        }

            private bool Open(string FileName, ref IntPtr handle)
            {
                  try
                  {
                        if (handle == System.IntPtr.Zero && (int)handle > 0)
                              CloseCom(ref handle);

                        // open the existing file for reading
                        handle = SerialComClass.CreateFile(FileName,
GENERIC_READ|GENERIC_WRITE, 0, 0, OPEN_EXISTING, 0, 0);

                        //int e = Marshal.GetLastWin32Error();
                        if (handle != System.IntPtr.Zero && (int)handle > 0)
                        {
                              //bool s = SerialComClass.PurgeComm(handle,
0x0008|0x0004);

                              configComm(handle);

                              return true;
                        }
```

```
                        else
                        {
                              return false;
                        }
                  }
                  catch (Exception exc)
                  {
                        if ((int)handle > 0)
                              CloseCom(ref handle);
                        Output("Open: Error(s) Occured: " + exc.Message + ",
Last Error: " + Marshal.GetLastWin32Error().ToString(), true);
                        return false;
                  }
            }

            private unsafe int Read(byte[] buffer, int index, int count, uint
timeoutMilli, ref IntPtr handle)
            {
                  try
                  {
                        int n = 0;
                        fixed (byte* p = buffer)
                        {
                              int milli = (DateTime.Now.Minute*60*1000) +
(DateTime.Now.Second*1000) + (DateTime.Now.Millisecond);
                              while (n < 1)
                              {
                                    if (!SerialComClass.ReadFile(handle, p +
index, count, &n, 0))
                                    {
                                 Output("Read: Error(s) Occured: ReadFile failed,
handle: " + handle.ToString(), true);
//                                        Open(comPort);
                                          return 0;
                                    }
                                    if (Math.Abs(milli-
((DateTime.Now.Minute*60*1000) + (DateTime.Now.Second*1000) +
(DateTime.Now.Millisecond))) > timeoutMilli)
                                          break;
                              }
                        }
                        int e = Marshal.GetLastWin32Error();
                        return n;
                  }
                  catch (Exception exc)
                  {
                        Output("Read: Error(s) Occured: " + exc.Message + ",
" + Marshal.GetLastWin32Error().ToString(), true);
                        return 0;
                  }
            }

            private unsafe int Write(byte[] buffer, int bytesToWrite, ref
IntPtr handle)
            {
                  try
                  {
```

```
                        int n = 0;
                        fixed (byte* p = buffer)
                        {
                              if (!SerialComClass.WriteFile(handle, p,
bytesToWrite, &n, 0))
                              {
                        Output("Write: Error(s) Occured: WriteFile failed,
handle: " + handle.ToString(), true);
//                        Open(comPort);
                                    return 0;
                              }
                        }
                        int e = Marshal.GetLastWin32Error();
                        return n;
                  }
                  catch (Exception exc)
                  {
                        Output("Write: Error(s) Occured: " + exc.Message + ",
" + Marshal.GetLastWin32Error().ToString(), true);
                        return 0;
                  }
            }

            private bool CloseCom(ref IntPtr handle)
            {
                  try
                  {
                        bool result = false;
                        if (handle != System.IntPtr.Zero)
                              if (!SerialComClass.CloseHandle(handle))
                                    throw new Exception("CloseHandle returned
false!");

                        return result;
                  }
                  catch (Exception exc)
                  {
                        Output("CloseCom: Error(s) Occured: " + exc.Message +
", " + Marshal.GetLastWin32Error().ToString(), true);
                        return false;
                  }
            }

            public void EntryPoint ()
            {
                  try
                  {
                        //Re-read XML configurations
                        GetConfigurations();

                        mre = new ManualResetEvent(false);

                threadReader1 = new Thread(new
ThreadStart(receiveFromReader1));
                threadReader1.Start();
```

```
                threadReader2 = new Thread(new
ThreadStart(receiveFromReader2));
                threadReader2.Start();

                threadTagLogic = new Thread(new ThreadStart(RFIDTagLogic));
                threadTagLogic.Start();

                threadMonitorTags = new Thread(new ThreadStart(monitorTags));
                threadMonitorTags.Start();
                        //receiveFromReader();
                  }
                  catch (Exception exc)
                  {
                        Output("EntryPoint: Error(s) Occured: " + exc.Message
+ ", " + Marshal.GetLastWin32Error().ToString(), true);
                  }
            }

        private void receiveFromReader1()
        {
                  try
                  {
                tagAttr ta;

                        Output("Reader 1 started...", RFIDmain.bDebug);

                        //                      int inx = 0;
                        //                      int temp_inx = 0;
                        int bRead = 0;
                        int tagID = 0;
                        byte RSSI = 0;
                        int siteID = 0;
                        byte [] buf = new byte[1024];
                        string oneDataPacket = "";
                ArrayList dataPackets = null;
                int arrayIndx = 0;

                        if (RFIDmain.bDebug)
                        {
                    if (RFIDmain.bHook)
                        //this.lbState.Text = "Hooked";
                        LabelPrint(this.lbState, "Hooked");
                    else
                        //this.lbState.Text = "Un-Hooked";
                        LabelPrint(this.lbState, "Un-Hooked");
                        }

                //DoorLock.OpenPort();

                        if (!Open(RFIDmain.ComPortR1, ref handleReader1))
                              throw new Exception("Error opening com port
reader 1!");

                //Reset Network
                byte checksum = 0x00 ^ 0x00 ^ 0x00 ^ 0xFF ^ 0x00;
                byte[] command = new byte[7] { 0xaa, 0x00, 0x00, 0x00, 0xFF,
0x00, checksum };
```

```
                // Write command
                bRead = Write(command, 7, ref handleReader1);

                // Write command to reader to enable auto polling
                checksum = 0x00 ^ 0x00 ^ 0x00 ^ 0x01 ^ 0x01;
                command = new byte[7] { 0xaa, 0x00, 0x00, 0x00, 0x01, 0x01,
checksum };

                // Write command
                bRead = Write(command, 7, ref handleReader1);

                // Write command Set RSSI value to most sensetive 0 - 255
zero is most sensetive
                checksum = 0x01 ^ 0x00 ^ 0x00 ^ 0x01 ^ 0x07 ^ 0x00;
                command = new byte[8] { 0xaa, 0x01, 0x00, 0x00, 0x01, 0x07,
0x00, checksum };

                // Write command
                bRead = Write(command, 8, ref handleReader1);

                int indexStep = 38; //38 bytes for L series readers
                /*Get Tag Packet
                 * Response:
                 * 0x55*            *Data Length*           *NetworkID*
     *ReceiverID*      *NodeID*    *0x06*      *Data*      *Checksum*
                 * Total Length Header and Data is:
                 * L-SERIES = 38
                 * W-SERISS = 21
                */
                //if (RFIDmain.readerType == 1)
                //    indexStep = 21; //21 bytes for W series readers

                while (!(mre.WaitOne(750, true)))
                {
                    try
                    {
                        buf.Initialize();
                        //buf = new byte[1024];
                        bRead = Read(buf, 0, 1024, 0, ref handleReader1);
                        //                              inx = 0;
                        //                                    ulTag.Clear();
                        if (bRead > 0)
                        {
                            dataPackets = new ArrayList();
                            arrayIndx = 0;

                            while ((arrayIndx + indexStep) <= bRead)
                            {
                                //L-SERIES Format: Check for packet header
checks first 3 bytes could expand to check first 4,5,6,7 or 8
                                if (buf[arrayIndx] == 85 && buf[arrayIndx +
1] == 32 && buf[arrayIndx + 2] == 80 /*&& buf[arrayIndx+3] == 1 &&
buf[arrayIndx+4] == 2 && buf[arrayIndx+5] == 33 && buf[arrayIndx+5] == 42 &&
buf[arrayIndx+5] == 42*/)
                                {
```

```
                                    dataPackets.Add(Encoding.UTF7.GetString(b
uf, arrayIndx, indexStep));
                                    arrayIndx += indexStep;
                                }
                                else
                                {
                                    break;
                                }
                            }

                            if (dataPackets.Count > 0)
                            {
                                for (int i = 0; i < dataPackets.Count; i++)
                                {
                                    try
                                    {
                                        oneDataPacket =
(string)dataPackets[i];

                                        tagID = (((byte)oneDataPacket[24]) <<
0) | (((byte)oneDataPacket[23]) << 8) | (((byte)oneDataPacket[22]) << 16) |
(((byte)oneDataPacket[21]) << 24);
                                        RSSI = ((byte)oneDataPacket[27]);
                                        siteID = (((byte)oneDataPacket[20])
<< 0) | (((byte)oneDataPacket[19]) << 8) | (((byte)oneDataPacket[18]) << 16);
                                    }
                                    catch (Exception exc)
                                    {
                                        tagID = -1;
                                        Output("receiveFromReader1 Data
packet parsing: Error(s) Occured: " + exc.Message + ", Last Error: " +
Marshal.GetLastWin32Error().ToString(), true);
                                    }

                                    if (tagID > 0)
                                    {
                                        if (SITE_CODE.Trim().Equals("0") ||
SITE_CODE.Trim().Equals("") || SITE_CODE.IndexOf(siteID.ToString()) > -1)
                                        {
                                            ta = new tagAttr();
                                            ta.RSSI = RSSI;
                                            ta.timeStamp = DateTime.UtcNow;
                                            ta.occuranceCount = 1;

                                            lock (hsTagReader1)
                                            {
                                                if (RSSI >=
RFIDmain.ssHookThreshold)
                                                {
                                                    if
(hsTagReader1.ContainsKey(tagID))
                                                    {
                                                        //hsTagReader1[tagID]
= ta;
                                                        ((tagAttr)hsTagReader
1[tagID]).RSSI = ta.RSSI;
```

```
                                                        if (((TimeSpan)
(ta.timeStamp - ((tagAttr)hsTagReader1[tagID]).timeStamp)).TotalMilliseconds
< 2001)
                                                            ((tagAttr)hsTagRe
ader1[tagID]).occuranceCount++;
                                                        else
                                                            ((tagAttr)hsTagRe
ader1[tagID]).occuranceCount = 1;
                                                        ((tagAttr)hsTagReader
1[tagID]).timeStamp = ta.timeStamp;

                                                        if
(((tagAttr)hsTagReader1[tagID]).occuranceCount >= (RFIDmain.hookWaitTime))
                                                        {
                                                            if (RSSI >=
RFIDmain.ssDropThreshold)
                                                                TagHooked(tag
ID.ToString(), 1);
                                                            //DoorLock.OpenDo
or(2, "Feto");
                                                            if
(((employee)Employees[tagID.ToString()]).state == EmployeeState.clockedIn &&
(((tagAttr)hsTagReader2[tagID]) == null ||
((tagAttr)hsTagReader2[tagID]).RSSI < RSSI))
                                                            {
                                                                DoorLock.cloc
kedOut(((employee)Employees[tagID.ToString()]).firstName + "," +
((employee)Employees[tagID.ToString()]).lastName);
                                                                ((employee)Em
ployees[tagID.ToString()]).state = EmployeeState.clockedOut;
                                                                gbProgress.Ba
ckColor = Color.LightBlue;
                                                                LabelPrint(lb
Status, "Clocked Out");
                                                            }
                                                        }
                                                    }
                                                    else
                                                    {
                                                        hsTagReader1.Add(tagI
D, ta);
                                                        Output("TagID: " +
tagID + " has been Added", bDebug);
                                                    }
                                                }
                                                else
                                                {
                                                    if
(hsTagReader1.ContainsKey(tagID))
                                                    {
                                                        ((tagAttr)hsTagReader
1[tagID]).RSSI = ta.RSSI;
                                                    }
                                                }
                                            }
                                        }
                                    }
```

```
                                }
                            }
                        }
                    }
                    catch (Exception exc)
                    {
                        Output("receiveFromReader1 while loop: Error(s)
Occured: " + exc.Message + ", Last Error: " +
Marshal.GetLastWin32Error().ToString(), true);
                    }
                }
                  }
                  catch (Exception exc)
                  {
                Output("receiveFromReader1: Error(s) Occured: " + exc.Message
+ ", Last Error: " + Marshal.GetLastWin32Error().ToString(), true);
                  }
            }

        private void receiveFromReader2()
        {
            try
            {
                tagAttr ta;

                Output("Reader 2 started...", RFIDmain.bDebug);

                //                        int inx = 0;
                //                        int temp_inx = 0;
                int bRead = 0;
                int tagID = 0;
                byte RSSI = 0;
                int siteID = 0;
                byte[] buf = new byte[1024];
                string oneDataPacket = "";
                ArrayList dataPackets = null;
                int arrayIndx = 0;

                if (RFIDmain.bDebug)
                {
                    if (RFIDmain.bHook)
                        //this.lbState.Text = "Hooked";
                        LabelPrint(this.lbState, "Hooked");
                    else
                        //this.lbState.Text = "Un-Hooked";
                        LabelPrint(this.lbState, "Un-Hooked");
                }

                //DoorLock.OpenPort();

                if (!Open(RFIDmain.ComPortR2, ref handleReader2))
                    throw new Exception("Error opening com port reader 2!");

                //Reset Network
                byte checksum = 0x00 ^ 0x00 ^ 0x00 ^ 0xFF ^ 0x00;
                byte[] command = new byte[7] { 0xaa, 0x00, 0x00, 0x00, 0xFF,
0x00, checksum };
```

```
                // Write command
                bRead = Write(command, 7, ref handleReader2);

                // Write command to reader to enable auto polling
                checksum = 0x00 ^ 0x00 ^ 0x00 ^ 0x01 ^ 0x01;
                command = new byte[7] { 0xaa, 0x00, 0x00, 0x00, 0x01, 0x01,
checksum };

                // Write command
                bRead = Write(command, 7, ref handleReader2);

                // Write command Set RSSI value to most sensetive 0 - 255
zero is most sensetive
                checksum = 0x01 ^ 0x00 ^ 0x00 ^ 0x01 ^ 0x07 ^ 0x00;
                command = new byte[8] { 0xaa, 0x01, 0x00, 0x00, 0x01, 0x07,
0x00, checksum };

                // Write command
                bRead = Write(command, 8, ref handleReader2);

                int indexStep = 38; //38 bytes for L series readers
                /*Get Tag Packet
                 * Response:
                 * 0x55*            *Data Length*           *NetworkID*
     *ReceiverID*      *NodeID*    *0x06*      *Data*      *Checksum*
                 * Total Length Header and Data is:
                 * L-SERIES = 38
                 * W-SERISS = 21
                */
                //if (RFIDmain.readerType == 1)
                //    indexStep = 21; //21 bytes for W series readers

                while (!(mre.WaitOne(750, true)))
                {
                    try
                    {
                        buf.Initialize();
                        //buf = new byte[1024];
                        bRead = Read(buf, 0, 1024, 0, ref handleReader2);
                        //                              inx = 0;
  //                      ulTag.Clear();
                        if (bRead > 0)
                        {
                            dataPackets = new ArrayList();
                            arrayIndx = 0;

                            while ((arrayIndx + indexStep) <= bRead)
                            {
                                ////W-SERIES Format: Check for packet header
checks first 3 bytes could expand to check first 4,5, or 6
                                //if (buf[arrayIndx] == 85 && buf[arrayIndx +
1] == 14 && buf[arrayIndx + 2] == 0 /*&& buf[arrayIndx+3] == 0 &&
buf[arrayIndx+4] == 1 && buf[arrayIndx+5] == 6*/)
                                //{
                                //
dataPackets.Add(Encoding.UTF7.GetString(buf, arrayIndx, indexStep));
```

```
                                //    arrayIndx += indexStep;
                                //}
                                //L-SERIES Format: Check for packet header
checks first 3 bytes could expand to check first 4,5,6,7 or 8
                                if (buf[arrayIndx] == 85 && buf[arrayIndx +
1] == 32 && buf[arrayIndx + 2] == 80 /*&& buf[arrayIndx+3] == 1 &&
buf[arrayIndx+4] == 2 && buf[arrayIndx+5] == 33 && buf[arrayIndx+5] == 42 &&
buf[arrayIndx+5] == 42*/)
                                {
                                    dataPackets.Add(Encoding.UTF7.GetString(b
uf, arrayIndx, indexStep));
                                    arrayIndx += indexStep;
                                }
                                else
                                {
                                    break;
                                }
                            }

                            if (dataPackets.Count > 0)
                            {
                                for (int i = 0; i < dataPackets.Count; i++)
                                {
                                    try
                                    {
                                        oneDataPacket =
(string)dataPackets[i];

                                        //tagID = (((byte)oneDataPacket[14])
<< 0) | (((byte)oneDataPacket[13]) << 8) | (((byte)oneDataPacket[12]) << 16)
| (((byte)oneDataPacket[11]) << 24);
                                        //RSSI = ((byte)oneDataPacket[6]);
                                        //siteID = (((byte)oneDataPacket[10])
<< 0) | (((byte)oneDataPacket[9]) << 8) | (((byte)oneDataPacket[8]) << 16);
                                        tagID = (((byte)oneDataPacket[24]) <<
0) | (((byte)oneDataPacket[23]) << 8) | (((byte)oneDataPacket[22]) << 16) |
(((byte)oneDataPacket[21]) << 24);
                                        RSSI = ((byte)oneDataPacket[27]);
                                        siteID = (((byte)oneDataPacket[20])
<< 0) | (((byte)oneDataPacket[19]) << 8) | (((byte)oneDataPacket[18]) << 16);
                                    }
                                    catch (Exception exc)
                                    {
                                        tagID = -1;
                                        Output("receiveFromReader2 Data
packet parsing: Error(s) Occured: " + exc.Message + ", Last Error: " +
Marshal.GetLastWin32Error().ToString(), true);
                                    }

                                    if (tagID > 0)
                                    {
                                        if (SITE_CODE.Trim().Equals("0") ||
SITE_CODE.Trim().Equals("") || SITE_CODE.IndexOf(siteID.ToString()) > -1)
                                        {
                                            ta = new tagAttr();
                                            ta.RSSI = RSSI;
                                            ta.timeStamp = DateTime.UtcNow;
```

```
                                        ta.occuranceCount = 1;

                                        lock (hsTagReader2)
                                        {
                                            if (RSSI >=
RFIDmain.ssHookThreshold)
                                            {
                                                if
(hsTagReader2.ContainsKey(tagID))
                                                {
                                                    //hsTagReader2[tagID]
= ta;
                                                    ((tagAttr)hsTagReader
2[tagID]).RSSI = ta.RSSI;
                                                    if (((TimeSpan)
(ta.timeStamp - ((tagAttr)hsTagReader2[tagID]).timeStamp)).TotalMilliseconds
< 2001)
                                                        ((tagAttr)hsTagRe
ader2[tagID]).occuranceCount++;
                                                    else
                                                        ((tagAttr)hsTagRe
ader2[tagID]).occuranceCount = 1;
                                                    ((tagAttr)hsTagReader
2[tagID]).timeStamp = ta.timeStamp;

                                                    if
(((tagAttr)hsTagReader2[tagID]).occuranceCount >= (RFIDmain.hookWaitTime))
                                                    {
                                                        if (RSSI >=
RFIDmain.ssDropThreshold)
                                                            TagHooked(tag
ID.ToString(), 2);
                                                        //DoorLock.OpenDo
or(2, "Feto");
                                                        if
(((employee)Employees[tagID.ToString()]).state == EmployeeState.clockedOut &&
(((tagAttr)hsTagReader1[tagID]) == null ||
((tagAttr)hsTagReader1[tagID]).RSSI < RSSI))
                                                        {
                                                            DoorLock.cloc
kedIn(((employee)Employees[tagID.ToString()]).firstName + "," +
((employee)Employees[tagID.ToString()]).lastName);
                                                            ((employee)Em
ployees[tagID.ToString()]).state = EmployeeState.clockedIn;
                                                            gbProgress.Ba
ckColor = Color.LightGreen;
                                                            LabelPrint(lb
Status, "Clocked In");
                                                        }
                                                    }
                                                }
                                                else
                                                {
                                                    hsTagReader2.Add(tagI
D, ta);
                                                    Output("TagID: " +
tagID + " has been Added", bDebug);
```

```
                                                    }
                                                }
                                                else
                                                {
                                                    if
(hsTagReader2.ContainsKey(tagID))
                                                    {
                                                        ((tagAttr)hsTagReader
2[tagID]).RSSI = ta.RSSI;
                                                    }
                                                }
                                            }
                                        }
                                    }
                                }
                            }
                            catch (Exception exc)
                            {
                                Output("receiveFromReader2 while loop: Error(s)
Occured: " + exc.Message + ", Last Error: " +
Marshal.GetLastWin32Error().ToString(), true);
                            }
                        }
                    }
                    catch (Exception exc)
                    {
                        Output("receiveFromReader2: Error(s) Occured: " + exc.Message
+ ", Last Error: " + Marshal.GetLastWin32Error().ToString(), true);
                    }
                }

                private void RFIDTagLogic()
                {
                    ArrayList ulTag = new ArrayList();

                    tagAttr ta;

                    if (RFIDmain.bHook)
                    {
                        ta = new tagAttr();
                        ta.RSSI = 0;
                        ta.timeStamp = DateTime.UtcNow;

                        lock (hsTagReader2)
                        {
                            if (hsTagReader2.ContainsKey(RFIDmain.strongestTag))
                                hsTagReader2[RFIDmain.strongestTag] = ta;
                            else
                            {
                                hsTagReader2.Add(RFIDmain.strongestTag, ta);
                                Output("TagID: " + RFIDmain.strongestTag + " Hooked
from last time", bDebug);
                            }
                        }
                    }
```

```
            int curStrongestTag = 0;
            int curStrongestRSSI = 0;
            byte gracePeriodCounter = 5; // Before de-throning the strongest
tag wait 5 samples

            bool hookTagQulified = false, dropTagQulified = false;

            while (!mre.WaitOne(500, true))
            {
                try
                {
                    if (hsTagReader1.Keys.Count > 0)
                    {
                        tagsReader1.Clear();
                        foreach (DictionaryEntry de in hsTagReader1)
                        {
                            tagsReader1.Add("Tag: " + de.Key + " SS:" +
((tagAttr)de.Value).RSSI + " OC:" + ((tagAttr)de.Value).occuranceCount);
                        }
                        if (bDebug)
                            Output1();
                    }

                    if (hsTagReader2.Keys.Count > 0)
                    {
                        tagsReader2.Clear();
                        foreach (DictionaryEntry de in hsTagReader2)
                        {
                            tagsReader2.Add("Tag: " + de.Key + " SS:" +
((tagAttr)de.Value).RSSI + " OC:" + ((tagAttr)de.Value).occuranceCount);
                        }
                        if (bDebug)
                            Output2();
                    }

                    if (DoorLock.isDoorOpen())
                    {
                        if (!DoorLock.isDoorLocked())
                            DoorLock.LockDoor();
                        TextBoxPrint(this.txDoorStatus, "OPEN", Color.Green);
                    }
                    else
                    {
                        //if (((TimeSpan)(DateTime.UtcNow -
doorUnlockedAt)).TotalSeconds > 5)
                        //    DoorLock.LockDoor();
                        TextBoxPrint(this.txDoorStatus, "CLOSED", Color.Red);
                    }

                    if (DoorLock.isDoorLocked())
                    {
                        //DoorLock.JustLockDoor();
                        TextBoxPrint(this.txLockStatus, "LOCKED", Color.Red);
                    }
                    else
                    {
```

```
                        TextBoxPrint(this.txLockStatus, "UNLOCKED",
Color.Green);
                    }

                    continue;
                    ulTag.Clear();

                    if (hsTagReader2.Keys.Count > 0)
                    {
                        tagsReader2.Clear();
                        foreach (DictionaryEntry de in hsTagReader2)
                        {
                            tagsReader2.Add("Tag: " + de.Key + " SS:" +
((tagAttr)de.Value).RSSI);
                            ulTag.Add(Convert.ToUInt64(de.Key) |
(Convert.ToUInt64(((tagAttr)de.Value).RSSI) << 32));
                        }
                        if (bDebug)
                            Output2();

                        ulTag.Sort();
                        curStrongestTag = (int)((ulong)
(((ulong)ulTag[ulTag.Count - 1]) & 0x00000000ffffffff));
                        curStrongestRSSI = (int)((ulong)
(((ulong)ulTag[ulTag.Count - 1]) >> 32));

                        // Check to see if current high tag is the same as
the old high tag
                        if (curStrongestTag == strongestTag &&
curStrongestRSSI >= RFIDmain.ssHookThreshold)
                        {
                            if (RFIDmain.bDebug)
                            {
                                if (!hookTagQulified && !RFIDmain.bHook)
                                    //this.lbState.Text = "Qualifing Tag for
Hook";
                                    LabelPrint(this.lbState, "Qualifing Tag
for Hook");
                            }

                            // Strong Counter callback
                            if (bDebug)
                                StrongCounter(OPTION_NORMAL);

                            // if it is increment counter of how long it has
been at the top
                            strongestTagCounter++;
                            if (strongestTagCounter > 5999)
                                strongestTagCounter = 6000;

                            if (strongestTagCounter > RFIDmain.hookWaitTime
&& !RFIDmain.bHook)
                            {
                                if (!hookTagQulified)
                                {
                                    Output("Hook Tag Qulified...", bDebug);
                                    if (RFIDmain.bDebug)
```

```
                                    {
                                        //this.lbState.Text = "Hook Tag
Qulified...";
                                        LabelPrint(this.lbState, "Hook Tag
Qulified...");
                                    }
                                }

                                hookTagQulified = true;

                                if (RFIDmain.bDebug)
                                {
                                    //this.lbState.Text = "Hooked";
                                    LabelPrint(this.lbState, "Hooked");
                                }

                                hookTagQulified = false;
                                RFIDmain.bHook = true;
                                //Publish("Hook;" + strongestTag + ";" +
RFIDmain.hookTimeStamp);

                                TagHooked(strongestTag.ToString(), 1);
                                //DoorLock.OpenDoor(2, "Feto");

                                WriteStateToFile();
                            }

                            // Reset grace period when you here from the
current strongest tag
                            gracePeriodCounter = 0;

                            Output("Strongest Tag: " +
strongestTag.ToString() + "   RSSI: " + (int)((ulong)
(((ulong)ulTag[ulTag.Count - 1]) >> 32)), bDebug);
                        }
                        else if (RFIDmain.bHook && (gracePeriodCounter <
RFIDmain.dropWaitTime || dropTagQulified))
                        {
                            if (((tagAttr)hsTagReader2[strongestTag]).RSSI <=
RFIDmain.ssDropThreshold)
                            {
                                if (RFIDmain.bDebug)
                                {
                                    if (!dropTagQulified && RFIDmain.bHook)
                                        //this.lbState.Text = "Disqualifing
Tag for Drop";
                                        LabelPrint(this.lbState,
"Disqualifing Tag for Drop");
                                }

                                if (bDebug)
                                {
                                    if (gracePeriodCounter % 2 == 0)
                                        StrongCounter(OPTION_GRACE);
                                    else
                                        StrongCounter(OPTION_NORMAL);
                                }
```

```
                                gracePeriodCounter++;

                                if (gracePeriodCounter >=
RFIDmain.dropWaitTime)
                                {
                                    if (!dropTagQulified)
                                    {
                                        Output("Drop Tag Qulified...",
bDebug);
                                        if (RFIDmain.bDebug)
                                        {
                                            //this.lbState.Text = "Drop Tag
Qulified...";
                                            LabelPrint(this.lbState, "Drop
Tag Qulified...");
                                        }
                                    }

                                    dropTagQulified = true;

                                    if (RFIDmain.bDebug)
                                    {
                                        //this.lbState.Text = "Dropped";
                                        LabelPrint(this.lbState, "Dropped");
                                    }

                                    dropTagQulified = false;
                                    RFIDmain.bHook = false;

                                    //Publish("Drop;" + strongestTag + ";" +
RFIDmain.dropTimeStamp);

                                    //DoorLock.LockDoor();

                                    WriteStateToFile();

                                    RFIDmain.strongestTag = -1;
                                }

                                Output("Lost Strongest Tag, Waiting Grace
Period: " + gracePeriodCounter.ToString(), bDebug);
                            }
                            else
                                gracePeriodCounter = 0;
                        }
                        // if the current high tag is not the same as old
then reset counter and set old tag to current tag
                        else
                        {
                            //                                          if
(RFIDmain.bHook)
                            //                                          {
                            //
      RFIDmain.bHook = false;
                            //
      Publish("Drop;" + strongestTag + ";");
                            //
```

```
                            //
      WriteStateToFile();
                            //                                                  }
                            gracePeriodCounter = 0; // reset garce period
when de-throning strongest tag
                            strongestTagCounter = 0;
                            strongestTag = curStrongestTag;

                            if (bDebug)
                                StrongTag();

                            // Snapshot and capture UTC time to get more
accurate time stamp of hook event
                            RFIDmain.hookTimeStamp =
DateTime.UtcNow.Hour.ToString().PadLeft(2, '0') +
DateTime.UtcNow.Minute.ToString().PadLeft(2, '0') +
DateTime.UtcNow.Second.ToString().PadLeft(2, '0')
                                + DateTime.UtcNow.Month.ToString().PadLeft(2,
'0') + DateTime.UtcNow.Day.ToString().PadLeft(2, '0') +
DateTime.UtcNow.Year.ToString().PadLeft(4, '0');

                            if (RFIDmain.bDebug)
                            {
                                //this.lbState.Text = "Un-Hooked Scanning";
                                LabelPrint(this.lbState, "Un-Hooked
Scanning");
                            }
                            Output("New Strongest Tag: " +
strongestTag.ToString() + "   RSSI: " + (int)((ulong)
(((ulong)ulTag[ulTag.Count - 1]) >> 32)), bDebug);
                        }
                    }
                }
                catch (Exception exc)
                {
                    Output("RFIDTagLogic: Error(s) Occured: " + exc.Message +
", Last Error: " + Marshal.GetLastWin32Error().ToString(), true);
                }
            }
        }

        private void TagHooked(string tag, byte fromReader)
        {
            try
            {
                string firstName = null, lastName = null;
                Settings.GetEmployeeInfo(tag, ref firstName, ref lastName);
                if (firstName != null)
                {
                    if (!DoorLock.isDoorOpen() && DoorLock.isDoorLocked())
                    {
                        if (fromReader == 2)
                            DoorLock.OpenDoor(1, firstName + ", " + lastName);
                        if (fromReader == 1)
                            DoorLock.OpenDoor(0, firstName + ", " + lastName);
                        //doorUnlockedAt = DateTime.UtcNow;
                    }
```

```
                    pbPicID.ImageLocation = Application.StartupPath + "\\" +
tag + ".jpg";
                    //TextBoxPrint(this.txLockStatus, "UNLOCKED",
Color.Green);
                    LabelPrint(this.lbName, firstName + ", " + lastName);
                    LabelPrint(this.lbID, tag);
                    LabelPrint(this.lbTime, DateTime.Now.ToString());
                    //Output(firstName + ", " + lastName + " CLOCKED IN @ " +
DateTime.Now.ToString(), true);
                }
            }
            catch (Exception exc)
            {
                Output("TagHooked: Error(s) Occured: " + exc.Message + ",
Last Error: " + Marshal.GetLastWin32Error().ToString(), true);
            }
        }

            private void WriteStateToFile()
            {
                  try
                  {
                        lock (this)
                        {
                              if (File.Exists(Application.StartupPath +
"\\LastState.txt"))
                                    File.Delete(Application.StartupPath + "\\
LastState.txt");

                              StreamWriter sw =
File.AppendText(Application.StartupPath + "\\LastState.txt");
                              if (RFIDmain.bHook)
                                    sw.WriteLine("State=Hook");
                              else
                                    sw.WriteLine("State=Drop");
                              sw.WriteLine("Tag=" + RFIDmain.strongestTag);
                              sw.Flush();
                              sw.Close();
                        }
                  }
                  catch (Exception exc)
                  {
                        throw new Exception(exc.Message);
                  }
            }

            private void monitorTags ()
            {
                  try
                  {
                        while (!mre.WaitOne(1500, true))
                        {
                              try
                              {
                        //Monitor Reader1
                        if (hsTagReader1.Keys.Count > 0)
                                    {
```

```
                            foreach (DictionaryEntry de in hsTagReader1)
                                    {
                                        if
(((tagAttr)de.Value).timeStamp.AddSeconds(RFIDmain.TAG_REMOVE_TIME) <
DateTime.UtcNow)
                                        {
                                    lock (hsTagReader1)
                                            {
                                                if
(int.Parse(de.Key.ToString()) == RFIDmain.strongestTag)
                                                {

      ((tagAttr)de.Value).RSSI = 0;
                                                }
                                                else
                                                {
                                      hsTagReader1.Remove(de.Key);

      Output("TagID: " + de.Key.ToString() + " has been removed", bDebug);
                                                    break;
                                                }
                                            }
                                        }
                                    }
                            //Monitor Reader2
                            if (hsTagReader2.Keys.Count > 0)
                            {
                                foreach (DictionaryEntry de in hsTagReader2)
                                {
                                    if
(((tagAttr)de.Value).timeStamp.AddSeconds(RFIDmain.TAG_REMOVE_TIME) <
DateTime.UtcNow)
                                    {
                                        lock (hsTagReader2)
                                        {
                                            if (int.Parse(de.Key.ToString()) ==
RFIDmain.strongestTag)
                                            {
                                                ((tagAttr)de.Value).RSSI = 0;
                                            }
                                            else
                                            {
                                                hsTagReader2.Remove(de.Key);
                                                Output("TagID: " +
de.Key.ToString() + " has been removed", bDebug);
                                                break;
                                            }
                                        }
                                    }
                                }
                            }
                                  catch (Exception exc)
                                  {
```

```
                                    Output("monitorTags: Error(s) Occured: "
+ exc.Message + ", Last Error: " + Marshal.GetLastWin32Error().ToString(),
true);
                              }
                        }
                  }
                  catch (Exception exc)
                  {
                        Output("monitorTags: Error(s) Occured: " +
exc.Message + ", Last Error: " + Marshal.GetLastWin32Error().ToString(),
true);
                  }
            }

            private void configComm(IntPtr handle)
            {
                  try
                  {
                        DCB dcb = new DCB();

                        //                IntPtr buffer =
Marshal.AllocCoTaskMem( Marshal.SizeOf( msg ));
                        //                Marshal.StructureToPtr( msg,
buffer, false );
                        //
                        //                SerialComClass.FillMemory(

                        // Get Com State
                        if (!SerialComClass.GetCommState (handle, ref dcb))
                              Output("Error calling GetCommState, Last Error:
" + Marshal.GetLastWin32Error().ToString(), true);
                        //                //string oldbaud =
dcb.BaudRate.ToString();
                        //
                        dcb.DCBLength = (uint)Marshal.SizeOf(dcb);
                        dcb.BaudRate = RFIDmain.baudRate; //CBR_57600;
                        dcb.ByteSize = 8;
                        dcb.Parity = Parity.None;
                        dcb.StopBits = StopBits.One;
                        dcb.Binary = true;
                        dcb.DtrControl = DtrControl.Enable;
                        dcb.RtsControl = RtsControl.Enable;
                        dcb.AbortOnError = false;
                        dcb.OutxCtsFlow = false;
                        dcb.OutxDsrFlow = false;

                        //bool b = SerialComClass.BuildCommDCB("57600,n,8,1",
ref dcb);

                        // Set Com State
                        bool success = SerialComClass.SetCommState(handle,
ref dcb);

                        if (!success)
                        {
                              Output("Error calling SetCommState, Last Error:
" + Marshal.GetLastWin32Error().ToString(), true);
```

```
                        }
                        else
                              Output("Baudrate is set to: " +
dcb.BaudRate.ToString(), true);

                        // Configure Read timeout values
                        COMMTIMEOUTS commTimeouts = new COMMTIMEOUTS();

                        if (!SerialComClass.GetCommTimeouts(handle, ref
commTimeouts))
                              Output("Error calling GetCommTimeouts, Last
Error: " + Marshal.GetLastWin32Error().ToString(), true);

                        commTimeouts.ReadIntervalTimeout = MAXDWORD;
                        commTimeouts.ReadTotalTimeoutConstant = 150;
                        commTimeouts.ReadTotalTimeoutMultiplier = 0;
                        commTimeouts.WriteTotalTimeoutConstant = 150;
                        commTimeouts.WriteTotalTimeoutMultiplier = 0;

                        // Set timeouts
                        success = SerialComClass.SetCommTimeouts(handle, ref
commTimeouts);

                        if (!success)
                        {
                              Output("Error calling SetCommTimeouts, Last
Error: " + Marshal.GetLastWin32Error().ToString(), true);
                        }
                        else
                              Output("Timeouts set successfully", true);
                  }
                  catch (Exception exc)
                  {
                        Output("configComm: Error(s) Occured: " + exc.Message
+ ", Last Error: " + Marshal.GetLastWin32Error().ToString(), true);
                  }
            }

            private void Output(string print, bool bDebug)
            {
                  try
                  {
                        if (RFIDmain.bDebug)
                        {
                              if (txOutput.InvokeRequired)
                              {
                                    OutputCallback delgCb = new
OutputCallback(Output);
                                    Invoke(delgCb, new object[] { print,
bDebug });
                              }
                              else
                              {
                                    if (txOutput.Text.Split('\n').Length < 10)
                                          txOutput.Text += print + "\r\n";
                                    else
                                          txOutput.Text = print + "\r\n";
```

```
                        }
                    }
                    else
                    {
                        if (bDebug)
                        {
                            lock (this)
                            {
                                if
(File.Exists(Application.StartupPath + "\\" + RFIDmain.logFileName))
                                {
                                    FileInfo fi = new
FileInfo(Application.StartupPath + "\\" + RFIDmain.logFileName);
                                    if (fi.Length > 100*1000)  //
if log file exeeds 100k delete it

      File.Delete(Application.StartupPath + "\\" + RFIDmain.logFileName);
                                }
                                StreamWriter sw =
File.AppendText(Application.StartupPath + "\\" + RFIDmain.logFileName);

      sw.WriteLine(DateTime.UtcNow.ToString() + "\t" + print + "\r");
                                sw.Flush();
                                sw.Close();
                            }
                        }
                    }
                }
                catch (Exception exc)
                {
                    throw new Exception("Output: Error(s) Occured: " +
exc.Message + ", Last Error: " + Marshal.GetLastWin32Error().ToString());
                }
            }

        private void Output1()
        {
            try
            {
                if (txOutput1.InvokeRequired)
                {
                    OutputCallback1 delgCb = new OutputCallback1(Output1);
                    Invoke(delgCb);
                }
                else
                {
                    txOutput1.Text = "";
                    for (int i = 0; i < tagsReader1.Count; i++)
                    {
                        txOutput1.Text += (string)tagsReader1[i] + "\r\n";
                    }
                }
            }
            catch (Exception exc)
            {
                throw new Exception("Output1: Error(s) Occured: " +
exc.Message + ", Last Error: " + Marshal.GetLastWin32Error().ToString());
```

```
            }
        }

            private void Output2()
            {
                  try
                  {
                        if (txOutput2.InvokeRequired)
                        {
                              OutputCallback2 delgCb = new
OutputCallback2(Output2);
                              Invoke(delgCb);
                        }
                        else
                        {
                              txOutput2.Text = "";
                              for (int i = 0; i < tagsReader2.Count; i++)
                              {
                                    txOutput2.Text += (string)tagsReader2[i]
+ "\r\n";
                              }
                        }
                  }
                  catch (Exception exc)
                  {
                        throw new Exception("Output2: Error(s) Occured: " +
exc.Message + ", Last Error: " + Marshal.GetLastWin32Error().ToString());
                  }
            }

        private void LabelPrint(Label label, string print)
        {
            try
            {
                if (label.InvokeRequired)
                {
                    LabelPrintCallback delgCb = new
LabelPrintCallback(LabelPrint);
                    Invoke(delgCb, new object[] { label, print });
                }
                else
                {
                    label.Text = print;
                }
            }
            catch (Exception exc)
            {
                throw new Exception("LabelPrint: Error(s) Occured: " +
exc.Message + ", Last Error: " + Marshal.GetLastWin32Error().ToString());
            }
        }

        private void TextBoxPrint(TextBox tb, string print, Color color)
        {
            try
            {
                if (tb.InvokeRequired)
```

```
                {
                    TextBoxPrintCallback delgCb = new
TextBoxPrintCallback(TextBoxPrint);
                    Invoke(delgCb, new object[] { tb, print, color });
                }
                else
                {
                    tb.Text = print;
                    tb.BackColor = color;
                }
            }
            catch (Exception exc)
            {
                throw new Exception("TextBoxPrint: Error(s) Occured: " +
exc.Message + ", Last Error: " + Marshal.GetLastWin32Error().ToString());
            }
        }

            private void StrongTag()
            {
                  try
                  {
                        if (lbStrongTag.InvokeRequired)
                        {
                              LabelStrongTagCallback delgCb = new
LabelStrongTagCallback(StrongTag);
                              Invoke(delgCb);
                        }
                        else
                              lbStrongTag.Text = strongestTag.ToString();
                  }
                  catch (Exception exc)
                  {
                        throw new Exception("StrongTag: Error(s) Occured: " +
exc.Message + ", Last Error: " + Marshal.GetLastWin32Error().ToString());
                  }
            }

            private void StrongCounter(byte option)
            {
                  try
                  {
                        if (lbStrongCounter.InvokeRequired)
                        {
                              LabelCallback delgCb = new
LabelCallback(StrongCounter);
                              Invoke(delgCb, new object []{ option });
                        }
                        else
                        {
                              if (((strongestTagCounter+40) % 100) == 0)
                                    strongestTagCounter += 40;
                              lbStrongCounter.Text =
strongestTagCounter.ToString().PadLeft(4,'0').Insert(2,":");
                              if (option == OPTION_NORMAL)
                                    lbStrongCounter.ForeColor =
Color.DarkGreen;
```

```
                              else if (option == OPTION_GRACE)
                                    lbStrongCounter.ForeColor = Color.Red;
                              else
                                    lbStrongCounter.ForeColor =
Color.DarkGreen;
                        }
                  }
                  catch (Exception exc)
                  {
                        throw new Exception("StringCounter: Error(s) Occured:
" + exc.Message + ", Last Error: " + Marshal.GetLastWin32Error().ToString());
                  }
            }

            private void btnStart_Click(object sender, System.EventArgs e)
            {
                  try
                  {
                        //Re-read XML configurations
                        GetConfigurations();

                    //Load list of active employees
                    LoadEmployees();

                    DoorLock.OpenPort();
                    DoorLock.LockDoor();

                        lbStrongTag.Text = RFIDmain.strongestTag.ToString();
                        btnStart.Enabled = false;
                        menuItemConfig.Enabled = false;
                        CconfigForm.Hide();

                        mre = new ManualResetEvent(false);

                threadReader1 = new Thread(new
ThreadStart(receiveFromReader1));
                        threadReader1.Start();

                threadReader2 = new Thread(new
ThreadStart(receiveFromReader2));
                threadReader2.Start();

                threadTagLogic = new Thread(new ThreadStart(RFIDTagLogic));
                threadTagLogic.Start();

                threadMonitorTags = new Thread(new ThreadStart(monitorTags));
                threadMonitorTags.Start();
                        //receiveFromReader();
                  }
                  catch (Exception exc)
                  {
                        Output("btnStart: Error(s) Occured: " + exc.Message +
", Last Error: " + Marshal.GetLastWin32Error().ToString(), true);
                  }
            }

            private void ShutdownAll()
```

```
            {
            try
            {
                if (threadReader1 != null || threadReader2 != null ||
threadTagLogic != null || threadMonitorTags != null)
                {
                    mre.Set();
                    if (threadReader1.IsAlive)
                    {
                        if (!threadReader1.Join(2000))
                            threadReader1.Abort();
                    }
                    if (threadReader2.IsAlive)
                    {
                        if (!threadReader2.Join(2000))
                            threadReader2.Abort();
                    }
                    if (threadTagLogic.IsAlive)
                    {
                        if (!threadTagLogic.Join(2000))
                            threadTagLogic.Abort();
                    }
                    if (threadMonitorTags.IsAlive)
                    {
                        if (!threadMonitorTags.Join(2000))
                            threadMonitorTags.Abort();
                    }
                }

                // FIXME do something with bSuccess var. Possibly log to file
                bool bSuccess = CloseCom(ref handleReader1);
                CloseCom(ref handleReader2);
                DoorLock.ClosePort();
                //Application.Exit();
            }
            catch (Exception exc)
            {
                Output("btnClose: Error(s) Occured: " + exc.Message + ", Last
Error: " + Marshal.GetLastWin32Error().ToString(), true);
            }
            finally
            {
                Application.Exit();
            }
            }

            private void btnClose_Click(object sender, System.EventArgs e)
            {
                  ShutdownAll();
            }

//          private void RFIDmain_Closing(object sender,
System.ComponentModel.CancelEventArgs e)
//          {
//                btnClose_Click(sender, e);
//          }
```

```
            private void btnStop_Click(object sender, System.EventArgs e)
            {
                  try
                  {
                 if (threadReader1 != null || threadReader2 != null ||
threadTagLogic != null || threadMonitorTags != null)
                        {
                              mre.Set();
                              if (threadReader1.IsAlive)
                              {
                        if (!threadReader1.Join(2000))
                            threadReader1.Abort();
                              }
                    if (threadReader2.IsAlive)
                    {
                        if (!threadReader2.Join(2000))
                            threadReader2.Abort();
                    }
                    if (threadTagLogic.IsAlive)
                    {
                        if (!threadTagLogic.Join(2000))
                            threadTagLogic.Abort();
                    }
                    if (threadMonitorTags.IsAlive)
                              {
                        if (!threadMonitorTags.Join(2000))
                            threadMonitorTags.Abort();
                              }
                        }

                        // FIXME do something with bSuccess var. Possibly log
to file
                bool bSuccess = CloseCom(ref handleReader1);
                CloseCom(ref handleReader2);
                DoorLock.ClosePort();
                        btnStart.Enabled = true;
                        menuItemConfig.Enabled = true;
                  }
                  catch (Exception exc)
                  {
                        Output("btnClose: Error(s) Occured: " + exc.Message +
", Last Error: " + Marshal.GetLastWin32Error().ToString(), true);
                  }
            }

            private void menuItem2_Click(object sender, System.EventArgs e)
            {
                  try
                  {
                        CconfigForm.Show();
                        CconfigForm.Focus();
                  }
                  catch(Exception exc)
                  {
                        Output("menuItemConfig_Click: Error(s) Occured: " +
exc.Message + ", Last Error: " + Marshal.GetLastWin32Error().ToString(),
true);
```

```
            }
        }

    public enum EmployeeState
    {
        clockedIn = 0,
        clockedOut = 1
    }

        public class tagAttr
        {
              public byte RSSI;
              public DateTime timeStamp;
        public byte occuranceCount;
        }

    public class employee
    {
        public string firstName;
        public string lastName;
        public string id;
        public EmployeeState state;
    }

    private void button1_Click(object sender, EventArgs e)
    {
        if (DoorLock.getComport().IsOpen)
            DoorLock.LockDoor();
        else
        {
            DoorLock.OpenPort();
            DoorLock.LockDoor();
        }
    }

    private void button2_Click(object sender, EventArgs e)
    {
        if (DoorLock.getComport().IsOpen)
            DoorLock.OpenDoor(0, "Admin Unlock");
        else
        {
            DoorLock.OpenPort();
            DoorLock.OpenDoor(0, "Admin Unlock");
        }
    }

    private void btnUnlockTimer_Click(object sender, EventArgs e)
    {
        if (DoorLock.getComport().IsOpen)
            DoorLock.OpenDoorManually();
        else
        {
            DoorLock.OpenPort();
            DoorLock.OpenDoorManually();
        }
    }
}
```

```
}
```

SerialComClass.cs

```
using System;
using System.Collections.Specialized;
using System.Runtime.InteropServices;

namespace RFIDapp
{
      public enum DtrControl
      {
            Disable = 0,
            Enable = 1,
            Handshake = 2
      }

      public enum RtsControl
      {
            Disable = 0,
            Enable = 1,
            Handshake = 2,
            Toggle = 3
      }

      public enum Parity
      {
            None = 0,
            Odd = 1,
            Even = 2,
            Mark = 3,
            Space = 4,
      }

      public enum StopBits
      {
            One = 0,
            One5 = 1,
            Two = 2
      }

      [StructLayout(LayoutKind.Sequential)]
      public struct DCB
      {
            internal uint DCBLength;
            internal uint BaudRate;
            private BitVector32 Flags;

            //I've missed some members...
            private uint wReserved;        // not currently used
            internal uint XonLim;           // transmit XON threshold
            internal uint XoffLim;          // transmit XOFF threshold

            internal byte ByteSize;
            internal Parity Parity;
            internal StopBits StopBits;
```

```
            //...and some more
            internal char XonChar;          // Tx and Rx XON character
            internal char XoffChar;         // Tx and Rx XOFF character
            internal char ErrorChar;        // error replacement character
            internal char EofChar;          // end of input character
            internal char EvtChar;          // received event character
            private uint wReserved1;       // reserved; do not use

            private static readonly int fBinary;
            private static readonly int fParity;
            private static readonly int fOutxCtsFlow;
            private static readonly int fOutxDsrFlow;
            private static readonly BitVector32.Section fDtrControl;
            private static readonly int fDsrSensitivity;
            private static readonly int fTXContinueOnXoff;
            private static readonly int fOutX;
            private static readonly int fInX;
            private static readonly int fErrorChar;
            private static readonly int fNull;
            private static readonly BitVector32.Section fRtsControl;
            private static readonly int fAbortOnError;

            static DCB()
            {
                  // Create Boolean Mask
                  int previousMask;
                  fBinary = BitVector32.CreateMask();
                  fParity = BitVector32.CreateMask(fBinary);
                  fOutxCtsFlow = BitVector32.CreateMask(fParity);
                  fOutxDsrFlow = BitVector32.CreateMask(fOutxCtsFlow);
                  previousMask = BitVector32.CreateMask(fOutxDsrFlow);
                  previousMask = BitVector32.CreateMask(previousMask);
                  fDsrSensitivity = BitVector32.CreateMask(previousMask);
                  fTXContinueOnXoff = BitVector32.CreateMask(fDsrSensitivity);
                  fOutX = BitVector32.CreateMask(fTXContinueOnXoff);
                  fInX = BitVector32.CreateMask(fOutX);
                  fErrorChar = BitVector32.CreateMask(fInX);
                  fNull = BitVector32.CreateMask(fErrorChar);
                  previousMask = BitVector32.CreateMask(fNull);
                  previousMask = BitVector32.CreateMask(previousMask);
                  fAbortOnError = BitVector32.CreateMask(previousMask);

                  // Create section Mask
                  BitVector32.Section previousSection;
                  previousSection = BitVector32.CreateSection(1);
                  previousSection = BitVector32.CreateSection(1,
previousSection);
                  previousSection = BitVector32.CreateSection(1,
previousSection);
                  previousSection = BitVector32.CreateSection(1,
previousSection);
                  fDtrControl = BitVector32.CreateSection(2, previousSection);
                  previousSection = BitVector32.CreateSection(1, fDtrControl);
                  previousSection = BitVector32.CreateSection(1,
previousSection);
```

```
                  previousSection = BitVector32.CreateSection(1,
previousSection);
                  previousSection = BitVector32.CreateSection(1,
previousSection);
                  previousSection = BitVector32.CreateSection(1,
previousSection);
                  previousSection = BitVector32.CreateSection(1,
previousSection);
                  fRtsControl = BitVector32.CreateSection(3, previousSection);
                  previousSection = BitVector32.CreateSection(1, fRtsControl);
            }

            public bool Binary
            {
                  get { return Flags[fBinary]; }
                  set { Flags[fBinary] = value; }
            }

            public bool CheckParity
            {
                  get { return Flags[fParity]; }
                  set { Flags[fParity] = value; }
            }

            public bool OutxCtsFlow
            {
                  get { return Flags[fOutxCtsFlow]; }
                  set { Flags[fOutxCtsFlow] = value; }
            }

            public bool OutxDsrFlow
            {
                  get { return Flags[fOutxDsrFlow]; }
                  set { Flags[fOutxDsrFlow]  = value; }
            }

            public DtrControl DtrControl
            {
                  get { return (DtrControl)Flags[fDtrControl]; }
                  set { Flags[fDtrControl] = (int)value; }
            }

            public bool DsrSensitivity
            {
                  get { return Flags[fDsrSensitivity]; }
                  set { Flags[fDsrSensitivity] = value; }
            }

            public bool TxContinueOnXoff
            {
                  get { return Flags[fTXContinueOnXoff]; }
                  set { Flags[fTXContinueOnXoff] = value; }
            }

            public bool OutX
            {
                  get { return Flags[fOutX]; }
```

```
                set { Flags[fOutX] = value; }
            }

            public bool InX
            {
                get { return Flags[fInX]; }
                set { Flags[fInX] = value; }
            }

            public bool ReplaceErrorChar
            {
                get { return Flags[fErrorChar]; }
                set { Flags[fErrorChar] = value; }
            }

            public bool Null
            {
                get { return Flags[fNull]; }
                set { Flags[fNull] = value; }
            }

            public RtsControl RtsControl
            {
                get { return (RtsControl)Flags[fRtsControl]; }
                set { Flags[fRtsControl]  = (int)value; }
            }

            public bool AbortOnError
            {
                get { return Flags[fAbortOnError]; }
                set { Flags[fAbortOnError] = value; }
            }
        }

        [StructLayout(LayoutKind.Sequential)]
        public struct COMMTIMEOUTS
        {
            public int ReadIntervalTimeout;                 // Maximum time
allowed to elapse between the arrival of two bytes on the communications
line, in milliseconds
            public int ReadTotalTimeoutMultiplier;  // Multiplier used to
calculate the total time-out period for read operations, in milliseconds
            public int ReadTotalTimeoutConstant;      // Constant used to
calculate the total time-out period for read operations, in milliseconds
            public int WriteTotalTimeoutMultiplier; // Multiplier used to
calculate the total time-out period for write operations, in milliseconds
            public int WriteTotalTimeoutConstant;     // Constant used to
calculate the total time-out period for write operations, in milliseconds
        }

        /// <summary>
        /// Summary description for SerialComClass.
        /// </summary>
        public class SerialComClass
        {
            public SerialComClass()
            {
```

```
}

[DllImport("kernel32", SetLastError = true)]
public static extern unsafe System.IntPtr CreateFile
      (
      string FileName,          // file name
      uint DesiredAccess,       // access mode
      uint ShareMode,           // share mode
      uint SecurityAttributes,  // Security Attributes
      uint CreationDisposition, // how to create
      uint FlagsAndAttributes,  // file attributes
      int hTemplateFile         // handle to template file
      );

[DllImport("kernel32", SetLastError = true)]
public static extern unsafe bool ReadFile
      (
      System.IntPtr hFile,      // handle to file
      void* pBuffer,            // data buffer
      int NumberOfBytesToRead,  // number of bytes to read
      int* pNumberOfBytesRead,  // number of bytes read
      int Overlapped            // overlapped buffer
      );

[DllImport("kernel32", SetLastError = true)]
public static extern unsafe bool WriteFile
      (
      System.IntPtr hFile,      // handle to file
      void* lpBuffer,            // data buffer
      int nNumberOfBytesToWrite,  // number of bytes to read
      int* lpNumberOfBytesWritten,  // number of bytes read
      int lpOverlapped            // overlapped buffer
      );

[DllImport("kernel32", SetLastError = true)]
public static extern unsafe bool CloseHandle
      (
      System.IntPtr hObject // handle to object
      );

[DllImport("kernel32", SetLastError = true)]
public static extern unsafe bool SetCommState
      (
      System.IntPtr hCommDev, // handle to Device Comm
      ref DCB lpDCB                // DCB struct object
      );

[DllImport("kernel32", SetLastError = true)]
public static extern unsafe bool GetCommState
      (
      System.IntPtr hCommDev, // handle to Devive Comm
      ref DCB lpDCB                // DCB struct object
      );

[DllImport("kernel32", SetLastError = true)]
public static extern unsafe bool SetCommTimeouts
      (
```

```
                  System.IntPtr hCommDev,             // handle to Device Comm
                  ref COMMTIMEOUTS lpCOMMTIMEOUTS     // COMMTIMEOUTS struct
object
                  );

            [DllImport("kernel32", SetLastError = true)]
            public static extern unsafe bool GetCommTimeouts
                  (
                  System.IntPtr hCommDev,             // handle to Devive Comm
                  ref COMMTIMEOUTS lpCOMMTIMEOUTS     // COMMTIMEOUTS struct
object
                  );

            [DllImport("kernel32", SetLastError = true)]
            public static extern unsafe bool BuildCommDCB
                  (
                  string lpDef,
                  ref DCB lpDCB
                  );

            [DllImport("kernel32", SetLastError = true)]
            public static extern System.IntPtr CreateEvent
                  (
                  System.IntPtr lpEventAttributes,
                  bool bManualReset,
                  bool bInitialState,
                  string lpName
                  );
      }
}
```

Configuration.cs

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Runtime.InteropServices;

namespace RFIDapp
{
      /// <summary>
      /// Summary description for Configuration.
      /// </summary>
      public class Configuration : System.Windows.Forms.Form
      {
            private System.Windows.Forms.Button btnSave;
            private System.Windows.Forms.GroupBox groupBox1;
            private System.Windows.Forms.Button btnCancel;
            private System.Windows.Forms.Label label6;
            private System.Windows.Forms.RadioButton rbDebugTrue;
            private System.Windows.Forms.RadioButton rbDebugFalse;
            private System.Windows.Forms.Label label1;
            private System.Windows.Forms.TextBox tbComPortL;
            private System.Windows.Forms.Label label2;
```

```
            private System.Windows.Forms.Label label3;
            private System.Windows.Forms.TextBox tbTabExpLimt;
            private System.Windows.Forms.GroupBox groupBox2;
            private System.Windows.Forms.Label label4;
            private System.Windows.Forms.Label label5;
            private System.Windows.Forms.Label label7;
            private System.Windows.Forms.GroupBox groupBox3;
            private System.Windows.Forms.Label label8;
            private System.Windows.Forms.Label label9;
            private System.Windows.Forms.Label label10;
            private System.Windows.Forms.TextBox tbSiteCodes;
            private System.Windows.Forms.TextBox tbHookSSDuration;
            private System.Windows.Forms.TextBox tbHookSSThreshold;
            private System.Windows.Forms.TextBox tbHookDistThreshold;
            private System.Windows.Forms.TextBox tbDropDistThreshold;
            private System.Windows.Forms.TextBox tbDropSSThreshold;
            private System.Windows.Forms.TextBox tbDropSSDuration;
        private TextBox tbComPortW;
        private Label label11;
            /// <summary>
            /// Required designer variable.
            /// </summary>
            private System.ComponentModel.Container components = null;

            [DllImport("user32.dll")]
            public static extern int SetWindowText
                  (
                  IntPtr hwnd,
                  string str
                  );

            public Configuration()
            {
                  //
                  // Required for Windows Form Designer support
                  //
                  InitializeComponent();

                  RFIDmain.GetConfigurations();
                  if (RFIDmain.bDebug)
                        rbDebugTrue.Checked = true;
                  tbComPortL.Text = RFIDmain.ComPortR1;
            tbComPortW.Text = RFIDmain.ComPortR2;
                  tbTabExpLimt.Text = RFIDmain.TAG_REMOVE_TIME.ToString();
                  tbSiteCodes.Text = RFIDmain.SITE_CODE;

                  //Hook Params
                  tbHookSSDuration.Text = RFIDmain.hookWaitTime.ToString();
                  tbHookSSThreshold.Text =
RFIDmain.ssHookThreshold.ToString();

                  //Drop Params
                  tbDropSSDuration.Text = RFIDmain.dropWaitTime.ToString();
                  tbDropSSThreshold.Text =
RFIDmain.ssDropThreshold.ToString();

                  SetWindowText(this.Handle, "RFIDapp_Configuration");
```

```
}

/// <summary>
/// Clean up any resources being used.
/// </summary>
protected override void Dispose( bool disposing )
{
      if( disposing )
      {
            if(components != null)
            {
                  components.Dispose();
            }
      }
      base.Dispose( disposing );
}

#region Windows Form Designer generated code
/// <summary>
/// Required method for Designer support - do not modify
/// the contents of this method with the code editor.
/// </summary>
private void InitializeComponent()
{
this.btnSave = new System.Windows.Forms.Button();
this.groupBox1 = new System.Windows.Forms.GroupBox();
this.groupBox2 = new System.Windows.Forms.GroupBox();
this.tbHookDistThreshold = new System.Windows.Forms.TextBox();
this.tbHookSSThreshold = new System.Windows.Forms.TextBox();
this.label7 = new System.Windows.Forms.Label();
this.label5 = new System.Windows.Forms.Label();
this.label4 = new System.Windows.Forms.Label();
this.tbHookSSDuration = new System.Windows.Forms.TextBox();
this.tbTabExpLimt = new System.Windows.Forms.TextBox();
this.tbSiteCodes = new System.Windows.Forms.TextBox();
this.label3 = new System.Windows.Forms.Label();
this.label2 = new System.Windows.Forms.Label();
this.tbComPortL = new System.Windows.Forms.TextBox();
this.label1 = new System.Windows.Forms.Label();
this.rbDebugFalse = new System.Windows.Forms.RadioButton();
this.rbDebugTrue = new System.Windows.Forms.RadioButton();
this.label6 = new System.Windows.Forms.Label();
this.btnCancel = new System.Windows.Forms.Button();
this.groupBox3 = new System.Windows.Forms.GroupBox();
this.label8 = new System.Windows.Forms.Label();
this.label9 = new System.Windows.Forms.Label();
this.label10 = new System.Windows.Forms.Label();
this.tbDropDistThreshold = new System.Windows.Forms.TextBox();
this.tbDropSSThreshold = new System.Windows.Forms.TextBox();
this.tbDropSSDuration = new System.Windows.Forms.TextBox();
this.tbComPortW = new System.Windows.Forms.TextBox();
this.label11 = new System.Windows.Forms.Label();
this.groupBox1.SuspendLayout();
this.groupBox2.SuspendLayout();
this.groupBox3.SuspendLayout();
this.SuspendLayout();
//
```

```
            // btnSave
            //
            this.btnSave.Enabled = false;
            this.btnSave.Font = new System.Drawing.Font("Arial", 15F,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point, ((byte)
(0)));
            this.btnSave.Location = new System.Drawing.Point(256, 392);
            this.btnSave.Name = "btnSave";
            this.btnSave.Size = new System.Drawing.Size(136, 50);
            this.btnSave.TabIndex = 3;
            this.btnSave.Text = "Save";
            //
            // groupBox1
            //
            this.groupBox1.Controls.Add(this.label11);
            this.groupBox1.Controls.Add(this.tbComPortW);
            this.groupBox1.Controls.Add(this.groupBox2);
            this.groupBox1.Controls.Add(this.tbTabExpLimt);
            this.groupBox1.Controls.Add(this.tbSiteCodes);
            this.groupBox1.Controls.Add(this.label3);
            this.groupBox1.Controls.Add(this.label2);
            this.groupBox1.Controls.Add(this.tbComPortL);
            this.groupBox1.Controls.Add(this.label1);
            this.groupBox1.Controls.Add(this.rbDebugFalse);
            this.groupBox1.Controls.Add(this.rbDebugTrue);
            this.groupBox1.Controls.Add(this.label6);
            this.groupBox1.Controls.Add(this.btnCancel);
            this.groupBox1.Controls.Add(this.btnSave);
            this.groupBox1.Controls.Add(this.groupBox3);
            this.groupBox1.Location = new System.Drawing.Point(8, 0);
            this.groupBox1.Name = "groupBox1";
            this.groupBox1.Size = new System.Drawing.Size(552, 448);
            this.groupBox1.TabIndex = 5;
            this.groupBox1.TabStop = false;
            this.groupBox1.Text = "Parameters";
            //
            // groupBox2
            //
            this.groupBox2.Controls.Add(this.tbHookDistThreshold);
            this.groupBox2.Controls.Add(this.tbHookSSThreshold);
            this.groupBox2.Controls.Add(this.label7);
            this.groupBox2.Controls.Add(this.label5);
            this.groupBox2.Controls.Add(this.label4);
            this.groupBox2.Controls.Add(this.tbHookSSDuration);
            this.groupBox2.Location = new System.Drawing.Point(8, 136);
            this.groupBox2.Name = "groupBox2";
            this.groupBox2.Size = new System.Drawing.Size(536, 120);
            this.groupBox2.TabIndex = 14;
            this.groupBox2.TabStop = false;
            this.groupBox2.Text = "Hook Parameters";
            //
            // tbHookDistThreshold
            //
            this.tbHookDistThreshold.Location = new System.Drawing.Point(248,
88);
            this.tbHookDistThreshold.Name = "tbHookDistThreshold";
            this.tbHookDistThreshold.ReadOnly = true;
```

```
            this.tbHookDistThreshold.Size = new System.Drawing.Size(280, 20);
            this.tbHookDistThreshold.TabIndex = 21;
            this.tbHookDistThreshold.Text = "0";
            //
            // tbHookSSThreshold
            //
            this.tbHookSSThreshold.Location = new System.Drawing.Point(248,
56);
            this.tbHookSSThreshold.Name = "tbHookSSThreshold";
            this.tbHookSSThreshold.ReadOnly = true;
            this.tbHookSSThreshold.Size = new System.Drawing.Size(280, 20);
            this.tbHookSSThreshold.TabIndex = 20;
            this.tbHookSSThreshold.Text = "0";
            //
            // label7
            //
            this.label7.Font = new System.Drawing.Font("Microsoft Sans
Serif", 14F, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, ((byte)(0)));
            this.label7.Location = new System.Drawing.Point(8, 88);
            this.label7.Name = "label7";
            this.label7.Size = new System.Drawing.Size(224, 24);
            this.label7.TabIndex = 17;
            this.label7.Text = "Distance Threshold:";
            //
            // label5
            //
            this.label5.Font = new System.Drawing.Font("Microsoft Sans
Serif", 14F, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, ((byte)(0)));
            this.label5.Location = new System.Drawing.Point(8, 56);
            this.label5.Name = "label5";
            this.label5.Size = new System.Drawing.Size(240, 24);
            this.label5.TabIndex = 16;
            this.label5.Text = "Signal Strength Threshold:";
            //
            // label4
            //
            this.label4.Font = new System.Drawing.Font("Microsoft Sans
Serif", 14F, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, ((byte)(0)));
            this.label4.Location = new System.Drawing.Point(8, 24);
            this.label4.Name = "label4";
            this.label4.Size = new System.Drawing.Size(224, 24);
            this.label4.TabIndex = 15;
            this.label4.Text = "Signal Strength Duration:";
            //
            // tbHookSSDuration
            //
            this.tbHookSSDuration.Location = new System.Drawing.Point(248,
24);
            this.tbHookSSDuration.Name = "tbHookSSDuration";
            this.tbHookSSDuration.ReadOnly = true;
            this.tbHookSSDuration.Size = new System.Drawing.Size(280, 20);
            this.tbHookSSDuration.TabIndex = 19;
            this.tbHookSSDuration.Text = "0";
            //
```

```
            // tbTabExpLimt
            //
            this.tbTabExpLimt.Location = new System.Drawing.Point(246, 102);
            this.tbTabExpLimt.Name = "tbTabExpLimt";
            this.tbTabExpLimt.ReadOnly = true;
            this.tbTabExpLimt.Size = new System.Drawing.Size(32, 20);
            this.tbTabExpLimt.TabIndex = 13;
            this.tbTabExpLimt.Text = "30";
            //
            // tbSiteCodes
            //
            this.tbSiteCodes.Location = new System.Drawing.Point(304, 56);
            this.tbSiteCodes.Multiline = true;
            this.tbSiteCodes.Name = "tbSiteCodes";
            this.tbSiteCodes.ReadOnly = true;
            this.tbSiteCodes.Size = new System.Drawing.Size(240, 64);
            this.tbSiteCodes.TabIndex = 12;
            this.tbSiteCodes.Text = "0";
            //
            // label3
            //
            this.label3.Font = new System.Drawing.Font("Microsoft Sans
Serif", 14F, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, ((byte)(0)));
            this.label3.Location = new System.Drawing.Point(8, 102);
            this.label3.Name = "label3";
            this.label3.Size = new System.Drawing.Size(240, 24);
            this.label3.TabIndex = 11;
            this.label3.Text = "Expired Tag Duration Limit:";
            //
            // label2
            //
            this.label2.Font = new System.Drawing.Font("Microsoft Sans
Serif", 14F, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, ((byte)(0)));
            this.label2.Location = new System.Drawing.Point(296, 24);
            this.label2.Name = "label2";
            this.label2.Size = new System.Drawing.Size(136, 24);
            this.label2.TabIndex = 10;
            this.label2.Text = "Site Code:";
            //
            // tbComPortL
            //
            this.tbComPortL.Location = new System.Drawing.Point(158, 54);
            this.tbComPortL.Name = "tbComPortL";
            this.tbComPortL.ReadOnly = true;
            this.tbComPortL.Size = new System.Drawing.Size(120, 20);
            this.tbComPortL.TabIndex = 9;
            this.tbComPortL.Text = "COM6:";
            //
            // label1
            //
            this.label1.Font = new System.Drawing.Font("Microsoft Sans
Serif", 9.75F, System.Drawing.FontStyle.Bold,
System.Drawing.GraphicsUnit.Point, ((byte)(0)));
            this.label1.Location = new System.Drawing.Point(8, 55);
            this.label1.Name = "label1";
```

```
            this.label1.Size = new System.Drawing.Size(148, 24);
            this.label1.TabIndex = 8;
            this.label1.Text = "COM Port L-Series:";
            //
            // rbDebugFalse
            //
            this.rbDebugFalse.Enabled = false;
            this.rbDebugFalse.Location = new System.Drawing.Point(228, 24);
            this.rbDebugFalse.Name = "rbDebugFalse";
            this.rbDebugFalse.Size = new System.Drawing.Size(56, 24);
            this.rbDebugFalse.TabIndex = 7;
            this.rbDebugFalse.Text = "False";
            this.rbDebugFalse.CheckedChanged += new
System.EventHandler(this.rbDebugFalse_CheckedChanged);
            //
            // rbDebugTrue
            //
            this.rbDebugTrue.Enabled = false;
            this.rbDebugTrue.Location = new System.Drawing.Point(160, 24);
            this.rbDebugTrue.Name = "rbDebugTrue";
            this.rbDebugTrue.Size = new System.Drawing.Size(48, 24);
            this.rbDebugTrue.TabIndex = 6;
            this.rbDebugTrue.Text = "True";
            this.rbDebugTrue.CheckedChanged += new
System.EventHandler(this.rbDebugTrue_CheckedChanged);
            //
            // label6
            //
            this.label6.Font = new System.Drawing.Font("Microsoft Sans
Serif", 14F, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, ((byte)(0)));
            this.label6.Location = new System.Drawing.Point(8, 24);
            this.label6.Name = "label6";
            this.label6.Size = new System.Drawing.Size(136, 24);
            this.label6.TabIndex = 5;
            this.label6.Text = "Debug Mode:";
            //
            // btnCancel
            //
            this.btnCancel.DialogResult =
System.Windows.Forms.DialogResult.Cancel;
            this.btnCancel.Font = new System.Drawing.Font("Arial", 15F,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point, ((byte)
(0)));
            this.btnCancel.Location = new System.Drawing.Point(408, 392);
            this.btnCancel.Name = "btnCancel";
            this.btnCancel.Size = new System.Drawing.Size(136, 50);
            this.btnCancel.TabIndex = 4;
            this.btnCancel.Text = "Cancel";
            this.btnCancel.Click += new
System.EventHandler(this.btnCancel_Click);
            //
            // groupBox3
            //
            this.groupBox3.Controls.Add(this.label8);
            this.groupBox3.Controls.Add(this.label9);
            this.groupBox3.Controls.Add(this.label10);
```

```
            this.groupBox3.Controls.Add(this.tbDropDistThreshold);
            this.groupBox3.Controls.Add(this.tbDropSSThreshold);
            this.groupBox3.Controls.Add(this.tbDropSSDuration);
            this.groupBox3.Location = new System.Drawing.Point(8, 264);
            this.groupBox3.Name = "groupBox3";
            this.groupBox3.Size = new System.Drawing.Size(536, 120);
            this.groupBox3.TabIndex = 18;
            this.groupBox3.TabStop = false;
            this.groupBox3.Text = "Drop Parameters";
            //
            // label8
            //
            this.label8.Font = new System.Drawing.Font("Microsoft Sans
Serif", 14F, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, ((byte)(0)));
            this.label8.Location = new System.Drawing.Point(8, 88);
            this.label8.Name = "label8";
            this.label8.Size = new System.Drawing.Size(224, 24);
            this.label8.TabIndex = 17;
            this.label8.Text = "Distance Threshold:";
            //
            // label9
            //
            this.label9.Font = new System.Drawing.Font("Microsoft Sans
Serif", 14F, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, ((byte)(0)));
            this.label9.Location = new System.Drawing.Point(8, 56);
            this.label9.Name = "label9";
            this.label9.Size = new System.Drawing.Size(240, 24);
            this.label9.TabIndex = 16;
            this.label9.Text = "Signal Strength Threshold:";
            //
            // label10
            //
            this.label10.Font = new System.Drawing.Font("Microsoft Sans
Serif", 14F, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, ((byte)(0)));
            this.label10.Location = new System.Drawing.Point(8, 24);
            this.label10.Name = "label10";
            this.label10.Size = new System.Drawing.Size(224, 24);
            this.label10.TabIndex = 15;
            this.label10.Text = "Signal Strength Duration:";
            //
            // tbDropDistThreshold
            //
            this.tbDropDistThreshold.Location = new System.Drawing.Point(248,
88);
            this.tbDropDistThreshold.Name = "tbDropDistThreshold";
            this.tbDropDistThreshold.ReadOnly = true;
            this.tbDropDistThreshold.Size = new System.Drawing.Size(280, 20);
            this.tbDropDistThreshold.TabIndex = 24;
            this.tbDropDistThreshold.Text = "0";
            //
            // tbDropSSThreshold
            //
            this.tbDropSSThreshold.Location = new System.Drawing.Point(248,
56);
```

```
            this.tbDropSSThreshold.Name = "tbDropSSThreshold";
            this.tbDropSSThreshold.ReadOnly = true;
            this.tbDropSSThreshold.Size = new System.Drawing.Size(280, 20);
            this.tbDropSSThreshold.TabIndex = 23;
            this.tbDropSSThreshold.Text = "0";
            //
            // tbDropSSDuration
            //
            this.tbDropSSDuration.Location = new System.Drawing.Point(248,
24);
            this.tbDropSSDuration.Name = "tbDropSSDuration";
            this.tbDropSSDuration.ReadOnly = true;
            this.tbDropSSDuration.Size = new System.Drawing.Size(280, 20);
            this.tbDropSSDuration.TabIndex = 22;
            this.tbDropSSDuration.Text = "0";
            //
            // tbComPortW
            //
            this.tbComPortW.Location = new System.Drawing.Point(158, 78);
            this.tbComPortW.Name = "tbComPortW";
            this.tbComPortW.ReadOnly = true;
            this.tbComPortW.Size = new System.Drawing.Size(120, 20);
            this.tbComPortW.TabIndex = 19;
            this.tbComPortW.Text = "COM6:";
            //
            // label11
            //
            this.label11.Font = new System.Drawing.Font("Microsoft Sans
Serif", 9.75F, System.Drawing.FontStyle.Bold,
System.Drawing.GraphicsUnit.Point, ((byte)(0)));
            this.label11.Location = new System.Drawing.Point(9, 78);
            this.label11.Name = "label11";
            this.label11.Size = new System.Drawing.Size(147, 24);
            this.label11.TabIndex = 20;
            this.label11.Text = "COM Port W-Series:";
            //
            // Configuration
            //
            this.AutoScaleBaseSize = new System.Drawing.Size(5, 13);
            this.CancelButton = this.btnCancel;
            this.ClientSize = new System.Drawing.Size(568, 454);
            this.ControlBox = false;
            this.Controls.Add(this.groupBox1);
            this.FormBorderStyle =
System.Windows.Forms.FormBorderStyle.FixedSingle;
            this.Name = "Configuration";
            this.Text = "Configuration";
            this.groupBox1.ResumeLayout(false);
            this.groupBox1.PerformLayout();
            this.groupBox2.ResumeLayout(false);
            this.groupBox2.PerformLayout();
            this.groupBox3.ResumeLayout(false);
            this.groupBox3.PerformLayout();
            this.ResumeLayout(false);

            }
            #endregion
```

```
            private void btnCancel_Click(object sender, System.EventArgs e)
            {
                  this.Hide();
            }

            private void rbDebugTrue_CheckedChanged(object sender,
System.EventArgs e)
            {
                  if (rbDebugTrue.Checked)
                        rbDebugFalse.Checked = false;
                  else
                        rbDebugFalse.Checked = true;
            }

            private void rbDebugFalse_CheckedChanged(object sender,
System.EventArgs e)
            {
                  if (rbDebugFalse.Checked)
                        rbDebugTrue.Checked = false;
                  else
                        rbDebugTrue.Checked = true;
            }
      }
}
```

Settings.cs

```
using System;
using System.Collections;
using System.Data.SqlClient;
using System.Xml;
using System.Web;
using System.Reflection;
using System.IO;
using System.Diagnostics;

namespace RFIDapp
{
      /// <summary>
      /// Summary description for Connection.
      /// </summary>
      public class Settings
      {
            public Settings()
            {
            }

            public static string GetPath()
            {
#if DEBUG
            return Directory.GetCurrentDirectory();
#else
                  Process oLocal =
System.Diagnostics.Process.GetCurrentProcess();
                  ProcessModule oMain = oLocal.MainModule;
```

```
                return Path.GetDirectoryName(oMain.FileName);
#endif
            }

        public static void GetProgramSettings(ref string DebugMode, ref
string ComPortR1, ref string ComPortR2, ref string SiteCode, ref string
ExpiredTagDuration)
            {
                  try
                  {
                        string path = Settings.GetPath() + "\\RFIDConfig.xml";
                        XmlDocument xmlDoc = new XmlDocument();
                        xmlDoc.Load(path);
                        XmlNodeList xmlList =
xmlDoc.GetElementsByTagName("Settings");
                        XmlNode node;

                        for (int i = 0; i < xmlList[0].ChildNodes.Count; i++)
                        {
                              node = xmlList[0].ChildNodes[i];
                              if (node.Name == "DebugMode")
                                    DebugMode = node.InnerText;
                    if (node.Name == "ComPortReader1")
                                    ComPortR1 = node.InnerText;
                    if (node.Name == "ComPortReader2")
                        ComPortR2 = node.InnerText;
                              if (node.Name == "SiteCode")
                                    SiteCode = node.InnerText;
                              if (node.Name == "ExpiredTagDuration")
                                    ExpiredTagDuration = node.InnerText;
                        }
                  }
                  catch (Exception exc)
                  {
                        throw new Exception("Settings.GetProgramSettings
function failed! " + exc.Message);
                  }
            }

            public static void GetHookParams(ref string SSDuration, ref
string SSThreshold)
            {
                  try
                  {
                        string path = Settings.GetPath() + "\\RFIDConfig.xml";
                        XmlDocument xmlDoc = new XmlDocument();
                        xmlDoc.Load(path);
                        XmlNodeList xmlList =
xmlDoc.GetElementsByTagName("HookParams");
                        XmlNode node;

                        for (int i = 0; i < xmlList[0].ChildNodes.Count; i++)
                        {
                              node = xmlList[0].ChildNodes[i];
                              if (node.Name == "SignalStrengthDuration")
                                    SSDuration = node.InnerText;
                              if (node.Name == "SignalStrengthThreshold")
```

```
                                    SSThreshold = node.InnerText;
                        }
                  }
                  catch (Exception exc)
                  {
                        throw new Exception("Settings.GetHookParams function
failed! " + exc.Message);
                  }
            }

            public static void GetDropParams(ref string SSDuration, ref
string SSThreshold)
            {
                  try
                  {
                        string path = Settings.GetPath() + "\\RFIDConfig.xml";
                        XmlDocument xmlDoc = new XmlDocument();
                        xmlDoc.Load(path);
                        XmlNodeList xmlList =
xmlDoc.GetElementsByTagName("DropParams");
                        XmlNode node;

                        for (int i = 0; i < xmlList[0].ChildNodes.Count; i++)
                        {
                              node = xmlList[0].ChildNodes[i];
                              if (node.Name == "SignalStrengthDuration")
                                    SSDuration = node.InnerText;
                              if (node.Name == "SignalStrengthThreshold")
                                    SSThreshold = node.InnerText;
                        }
                  }
                  catch (Exception exc)
                  {
                        throw new Exception("Settings.GetDropParams function
failed! " + exc.Message);
                  }
            }

        public static void GetEmployeeInfo(string id, ref string firstName,
ref string lastName)
        {
            try
            {
                string path = Settings.GetPath() + "\\Employees.xml";
                XmlDocument xmlDoc = new XmlDocument();
                xmlDoc.Load(path);
                XmlNodeList xmlList =
xmlDoc.GetElementsByTagName("Employees");
                XmlNode node;

                for (int i = 0; i < xmlList[0].ChildNodes.Count; i++)
                {
                    node = xmlList[0].ChildNodes[i];
                    if (node.Name == "Employee" &&
node.Attributes.GetNamedItem("ID").InnerText == id)
                    {
```

```
                        firstName =
node.Attributes.GetNamedItem("FirstName").InnerText;
                        lastName =
node.Attributes.GetNamedItem("LastName").InnerText;
                    }
                }
            }
            catch (Exception exc)
            {
                throw new Exception("Settings.GetEmployeeInfo function
failed! " + exc.Message);
            }
        }

        public static void GetEmployeeInfo(ref Hashtable Employees)
        {
            try
            {
                string path = Settings.GetPath() + "\\Employees.xml";
                XmlDocument xmlDoc = new XmlDocument();
                xmlDoc.Load(path);
                XmlNodeList xmlList =
xmlDoc.GetElementsByTagName("Employees");
                XmlNode node;

                RFIDapp.RFIDmain.employee ep = new RFIDmain.employee();

                for (int i = 0; i < xmlList[0].ChildNodes.Count; i++)
                {
                    node = xmlList[0].ChildNodes[i];
                    if (node.Name == "Employee")
                    {
                        ep.firstName =
node.Attributes.GetNamedItem("FirstName").InnerText;
                        ep.lastName =
node.Attributes.GetNamedItem("LastName").InnerText;
                        ep.id = node.Attributes.GetNamedItem("ID").InnerText;
                        ep.state = RFIDmain.EmployeeState.clockedOut;
                        Employees.Add(node.Attributes.GetNamedItem("ID").Inne
rText, ep);
                    }
                }
            }
            catch (Exception exc)
            {
                throw new Exception("Settings.GetEmployeeInfo function
failed! " + exc.Message);
            }
        }
    }
}
```

DoorController.cs

```
//
// RFID Door Lock Controller
```

```
//          by
//  Fetah Basic and Ken Dean
//
// Control communication between the
// micrcontroller and pc using a com port
//
//
//

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.IO.Ports;
using System.Threading;

namespace RFIDapp
{
    public partial class DoorController : Form
    {
        private Button button1;
        private Label label1;
        private ProgressBar progressBar1;
        private System.Windows.Forms.Timer timer1;
        private IContainer components;
        private TextBox textBox1;
        private TextBox textBox2;
        private TextBox textBox3;
        private TextBox textBox4;
        private Label label2;
        private Label label3;
        private Label label4;
        private Label label5;
        private Label label6;
        private Label label7;
        private ComPort DoorLock = new ComPort();
        delegate void SetTextCallback(string text);
        delegate void SetColorCallback(System.Drawing.Color text);

        public DoorController()
        {
            InitializeComponent();
            DoorLock.OpenPort();
            DoorLock.getComport().DataReceived +=new
System.IO.Ports.SerialDataReceivedEventHandler(isDoorOpen);
            TextBox tb1 = textBox1;
            TextBox tb2 = textBox3;
            TextBox tb3 = textBox4;
            tb1.KeyPress += new KeyPressEventHandler(keypressed);
            tb2.KeyPress += new KeyPressEventHandler(keypressed);
            tb3.KeyPress += new KeyPressEventHandler(keypressed);
            DoorLock.employeeName = textBox1.Text;
            DoorLock.greeting[0] = textBox3.Text;
            DoorLock.greeting[1] = textBox4.Text;
```

```
            DoorLock.getDoorStatus();
        }

        // door open or closed indicator call back
        void DoorController_DataReceived(object sender,
System.IO.Ports.SerialDataReceivedEventArgs e)
        {
            throw new NotImplementedException();
        }

        private void InitializeComponent()
        {
            this.components = new System.ComponentModel.Container();
            this.button1 = new System.Windows.Forms.Button();
            this.label1 = new System.Windows.Forms.Label();
            this.progressBar1 = new System.Windows.Forms.ProgressBar();
            this.timer1 = new System.Windows.Forms.Timer(this.components);
            this.textBox1 = new System.Windows.Forms.TextBox();
            this.textBox2 = new System.Windows.Forms.TextBox();
            this.textBox3 = new System.Windows.Forms.TextBox();
            this.textBox4 = new System.Windows.Forms.TextBox();
            this.label2 = new System.Windows.Forms.Label();
            this.label3 = new System.Windows.Forms.Label();
            this.label4 = new System.Windows.Forms.Label();
            this.label5 = new System.Windows.Forms.Label();
            this.label6 = new System.Windows.Forms.Label();
            this.label7 = new System.Windows.Forms.Label();
            this.SuspendLayout();
            //
            // button1
            //
            this.button1.Font = new System.Drawing.Font("Microsoft Sans
Serif", 12F, System.Drawing.FontStyle.Bold,
System.Drawing.GraphicsUnit.Point, ((byte)(0)));
            this.button1.Location = new System.Drawing.Point(287, 192);
            this.button1.Name = "button1";
            this.button1.Size = new System.Drawing.Size(244, 76);
            this.button1.TabIndex = 0;
            this.button1.Text = "Open Door";
            this.button1.UseVisualStyleBackColor = true;
            this.button1.Click += new
System.EventHandler(this.button1_Click_1);
            //
            // label1
            //
            this.label1.BackColor = System.Drawing.Color.LimeGreen;
            this.label1.Font = new System.Drawing.Font("Microsoft Sans
Serif", 12F, System.Drawing.FontStyle.Bold,
System.Drawing.GraphicsUnit.Point, ((byte)(0)));
            this.label1.Location = new System.Drawing.Point(287, 48);
            this.label1.Name = "label1";
            this.label1.Size = new System.Drawing.Size(244, 52);
            this.label1.TabIndex = 1;
            this.label1.Text = "Door is Closed";
            this.label1.TextAlign =
System.Drawing.ContentAlignment.MiddleCenter;
            //
```

```
            // progressBar1
            //
            this.progressBar1.Location = new System.Drawing.Point(287, 119);
            this.progressBar1.Name = "progressBar1";
            this.progressBar1.RightToLeft =
System.Windows.Forms.RightToLeft.Yes;
            this.progressBar1.RightToLeftLayout = true;
            this.progressBar1.Size = new System.Drawing.Size(244, 56);
            this.progressBar1.Step = 1;
            this.progressBar1.Style =
System.Windows.Forms.ProgressBarStyle.Continuous;
            this.progressBar1.TabIndex = 2;
            //
            // timer1
            //
            this.timer1.Enabled = true;
            this.timer1.Interval = 50;
            this.timer1.Tick += new System.EventHandler(this.timer1_Tick);
            //
            // textBox1
            //
            this.textBox1.AcceptsReturn = true;
            this.textBox1.Font = new System.Drawing.Font("Microsoft Sans
Serif", 12F, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, ((byte)(0)));
            this.textBox1.Location = new System.Drawing.Point(33, 48);
            this.textBox1.MaxLength = 20;
            this.textBox1.Name = "textBox1";
            this.textBox1.Size = new System.Drawing.Size(228, 26);
            this.textBox1.TabIndex = 3;
            this.textBox1.TabStop = false;
            this.textBox1.TextAlign =
System.Windows.Forms.HorizontalAlignment.Center;
            this.textBox1.TextChanged += new
System.EventHandler(this.textBox1_TextChanged);
            //
            // textBox2
            //
            this.textBox2.Font = new System.Drawing.Font("Microsoft Sans
Serif", 12F, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, ((byte)(0)));
            this.textBox2.Location = new System.Drawing.Point(33, 103);
            this.textBox2.MaxLength = 20;
            this.textBox2.Name = "textBox2";
            this.textBox2.Size = new System.Drawing.Size(228, 26);
            this.textBox2.TabIndex = 4;
            this.textBox2.TabStop = false;
            this.textBox2.Text = "08/23/08 12:55:37 PM";
            this.textBox2.TextAlign =
System.Windows.Forms.HorizontalAlignment.Center;
            //
            // textBox3
            //
            this.textBox3.AcceptsReturn = true;
            this.textBox3.Font = new System.Drawing.Font("Microsoft Sans
Serif", 12F, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, ((byte)(0)));
```

```
            this.textBox3.Location = new System.Drawing.Point(33, 158);
            this.textBox3.MaxLength = 20;
            this.textBox3.Name = "textBox3";
            this.textBox3.Size = new System.Drawing.Size(228, 26);
            this.textBox3.TabIndex = 5;
            this.textBox3.TabStop = false;
            this.textBox3.Text = "Hello";
            this.textBox3.TextAlign =
System.Windows.Forms.HorizontalAlignment.Center;
            this.textBox3.TextChanged += new
System.EventHandler(this.textBox3_TextChanged);
            //
            // textBox4
            //
            this.textBox4.AcceptsReturn = true;
            this.textBox4.Font = new System.Drawing.Font("Microsoft Sans
Serif", 12F, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, ((byte)(0)));
            this.textBox4.Location = new System.Drawing.Point(33, 213);
            this.textBox4.MaxLength = 20;
            this.textBox4.Name = "textBox4";
            this.textBox4.Size = new System.Drawing.Size(228, 26);
            this.textBox4.TabIndex = 6;
            this.textBox4.TabStop = false;
            this.textBox4.Text = "Goodbye";
            this.textBox4.TextAlign =
System.Windows.Forms.HorizontalAlignment.Center;
            this.textBox4.TextChanged += new
System.EventHandler(this.textBox4_TextChanged);
            //
            // label2
            //
            this.label2.Font = new System.Drawing.Font("Microsoft Sans
Serif", 8.25F, System.Drawing.FontStyle.Bold,
System.Drawing.GraphicsUnit.Point, ((byte)(0)));
            this.label2.Location = new System.Drawing.Point(30, 77);
            this.label2.Name = "label2";
            this.label2.Size = new System.Drawing.Size(231, 23);
            this.label2.TabIndex = 7;
            this.label2.Text = "Employee Name";
            this.label2.TextAlign =
System.Drawing.ContentAlignment.MiddleCenter;
            //
            // label3
            //
            this.label3.Font = new System.Drawing.Font("Microsoft Sans
Serif", 8.25F, System.Drawing.FontStyle.Bold,
System.Drawing.GraphicsUnit.Point, ((byte)(0)));
            this.label3.Location = new System.Drawing.Point(30, 132);
            this.label3.Name = "label3";
            this.label3.Size = new System.Drawing.Size(231, 23);
            this.label3.TabIndex = 8;
            this.label3.Text = "Time and Date";
            this.label3.TextAlign =
System.Drawing.ContentAlignment.MiddleCenter;
            //
            // label4
```

```
            //
            this.label4.Font = new System.Drawing.Font("Microsoft Sans
Serif", 8.25F, System.Drawing.FontStyle.Bold,
System.Drawing.GraphicsUnit.Point, ((byte)(0)));
            this.label4.Location = new System.Drawing.Point(30, 187);
            this.label4.Name = "label4";
            this.label4.Size = new System.Drawing.Size(231, 23);
            this.label4.TabIndex = 9;
            this.label4.Text = "Arriving message";
            this.label4.TextAlign =
System.Drawing.ContentAlignment.MiddleCenter;
            //
            // label5
            //
            this.label5.Font = new System.Drawing.Font("Microsoft Sans
Serif", 8.25F, System.Drawing.FontStyle.Bold,
System.Drawing.GraphicsUnit.Point, ((byte)(0)));
            this.label5.Location = new System.Drawing.Point(30, 242);
            this.label5.Name = "label5";
            this.label5.Size = new System.Drawing.Size(231, 23);
            this.label5.TabIndex = 10;
            this.label5.Text = "Departing message";
            this.label5.TextAlign =
System.Drawing.ContentAlignment.MiddleCenter;
            //
            // label6
            //
            this.label6.Font = new System.Drawing.Font("Microsoft Sans
Serif", 12F, System.Drawing.FontStyle.Bold,
System.Drawing.GraphicsUnit.Point, ((byte)(0)));
            this.label6.Location = new System.Drawing.Point(287, -1);
            this.label6.Name = "label6";
            this.label6.Size = new System.Drawing.Size(244, 49);
            this.label6.TabIndex = 11;
            this.label6.Text = "Door Control and Status";
            this.label6.TextAlign =
System.Drawing.ContentAlignment.MiddleCenter;
            //
            // label7
            //
            this.label7.Font = new System.Drawing.Font("Microsoft Sans
Serif", 12F, System.Drawing.FontStyle.Bold,
System.Drawing.GraphicsUnit.Point, ((byte)(0)));
            this.label7.Location = new System.Drawing.Point(33, -1);
            this.label7.Name = "label7";
            this.label7.Size = new System.Drawing.Size(228, 49);
            this.label7.TabIndex = 12;
            this.label7.Text = "LCD Display Data";
            this.label7.TextAlign =
System.Drawing.ContentAlignment.MiddleCenter;
            //
            // DoorController
            //
            this.ClientSize = new System.Drawing.Size(558, 295);
            this.Controls.Add(this.label7);
            this.Controls.Add(this.label6);
            this.Controls.Add(this.textBox1);
```

```
            this.Controls.Add(this.label2);
            this.Controls.Add(this.label3);
            this.Controls.Add(this.textBox2);
            this.Controls.Add(this.label4);
            this.Controls.Add(this.label5);
            this.Controls.Add(this.textBox3);
            this.Controls.Add(this.progressBar1);
            this.Controls.Add(this.label1);
            this.Controls.Add(this.button1);
            this.Controls.Add(this.textBox4);
            this.Name = "DoorController";
            this.Text = "Microcontroller interface";
            this.ResumeLayout(false);
            this.PerformLayout();

        }

        // Opens door manually
        private void button1_Click_1(object sender, EventArgs e)
        {
            this.button1.Enabled = false;
            DoorLock.OpenDoorManually();
            progressBar1.Value = progressBar1.Maximum;
            timer1.Start();
        }

        // Display a progress bar for time left that the door will remain
unlocked
        protected void timer1_Tick(object sender, System.EventArgs e)
        {
            if (progressBar1.Value > progressBar1.Minimum)
            {
                progressBar1.Value = progressBar1.Value - progressBar1.Step;
            }
            else
            {
                timer1.Stop();
                this.button1.Enabled = true;
                DoorLock.employeeName = "";
            }
        }

        // Indicates the door is open on the GUI
        private void setDoorOpen()
        {
            this.SetText("Door is Open");
            this.SetColor(System.Drawing.Color.Red);
        }

        // Indicates the door is closed on the GUI
        private void setDoorClosed()
        {
            this.SetText("Door is Closed");
            this.SetColor(System.Drawing.Color.LimeGreen);
        }

        // test for changes and reflect those changes on the GUI
```

```
void isDoorOpen(object sender, SerialDataReceivedEventArgs e)
{
    if (DoorLock.isDoorOpen())
    {
        setDoorOpen();
    }
    else
    {
        setDoorClosed();
    }
    this.setName(Convert.ToString(DoorLock.employeeName));
    this.setTime(DateTime.Now.ToString("MM/dd/yy hh:mm:ss tt"));
    this.setComing(Convert.ToString(DoorLock.greeting[0]));
    this.setGoing(Convert.ToString(DoorLock.greeting[1]));
}

// check for multiple threads on GUI
private void SetText(string text)
{
    if (this.label1.InvokeRequired)
    {
        SetTextCallback d = new SetTextCallback(SetText);
        this.Invoke(d, new object[] { text });
    }
    else
    {
        this.label1.Text = text;
    }
}

// check for multiple threads on GUI
private void SetColor(System.Drawing.Color text)
{
    if (this.label1.InvokeRequired)
    {
        SetColorCallback d = new SetColorCallback(SetColor);
        this.Invoke(d, new object[] { text });
    }
    else
    {
        this.label1.BackColor = text;
    }
}

// check for multiple threads on GUI
private void setName(string text)
{
    if (this.textBox1.InvokeRequired)
    {
        SetTextCallback d = new SetTextCallback(setName);
        this.Invoke(d, new object[] { text });
    }
    else
    {
        this.textBox1.Text = text;
    }
}
```

```
// check for multiple threads on GUI
private void setTime(string text)
{
    if (this.textBox2.InvokeRequired)
    {
        SetTextCallback d = new SetTextCallback(setTime);
        this.Invoke(d, new object[] { text });
    }
    else
    {
        this.textBox2.Text = text;
    }
}

// check for multiple threads on GUI
private void setComing(string text)
{
    if (this.textBox3.InvokeRequired)
    {
        SetTextCallback d = new SetTextCallback(setComing);
        this.Invoke(d, new object[] { text });
    }
    else
    {
        this.textBox3.Text = text;
    }
}

// check for multiple threads on GUI
private void setGoing(string text)
{
    if (this.textBox4.InvokeRequired)
    {
        SetTextCallback d = new SetTextCallback(setGoing);
        this.Invoke(d, new object[] { text });
    }
    else
    {
        this.textBox4.Text = text;
    }
}

// Send to the LCD
private void textBox1_TextChanged(object sender, EventArgs e)
{
    DoorLock.employeeName = textBox1.Text;
}

// Send to the LCD
private void textBox3_TextChanged(object sender, EventArgs e)
{
    DoorLock.greeting[0] = textBox3.Text;
}

// sSend to the LCD
private void textBox4_TextChanged(object sender, EventArgs e)
```

```
        {
            DoorLock.greeting[1] = textBox4.Text;
        }

        // change focus if text is entered
        private void keypressed(Object o, KeyPressEventArgs e)
        {
            if (e.KeyChar == (char)Keys.Return)
            {
                e.Handled = true;
                button1.Focus();
                this.button1.TabIndex = 0;
            }
        }
    }
}
```

ComPort.cs

```
//
// RFID Door Lock Controller
//          by
//  Fetah Basic and Ken Dean
//
// Control communication between the
// micrcontroller and pc using a com port
//
//
//

using System;
using System.Text;
using System.Drawing;
using System.IO.Ports;
using System.Windows.Forms;
using System.Timers;

namespace RFIDapp
{
    public class ComPort
    {
        static SerialPort comPort;
        private static bool isOpen = false;
        public string employeeName;
        public string[] greeting = { "Hello", "Goodbye", "Manual unlock" };
        private string[] clockinout = { "ClockedOut Opening", "ClockedIn
Opening", "Computer override" };
        private string[] LCDcommand = { "\x03", "\x04", "\x05", "\x06" };
        private string[] lockCommand = { "\x02", "\x01"};
        public static bool isLocked = true;

        // Set up com port 3 for 9600 baud, 8 data bits, no parity, 1 stop bit
        public void OpenPort()
        {
            comPort = new SerialPort();
            comPort.PortName = "COM4";
```

```
            comPort.BaudRate = 9600;
            comPort.Parity =
(System.IO.Ports.Parity)Enum.Parse(typeof(System.IO.Ports.Parity), "None");
            comPort.DataBits = 8;
            comPort.StopBits =
(System.IO.Ports.StopBits)Enum.Parse(typeof(System.IO.Ports.StopBits), "One");
            comPort.Handshake = (Handshake)Enum.Parse(typeof(Handshake),
"None");
            comPort.ReadTimeout = 500;
            comPort.WriteTimeout = 500;
            comPort.DataReceived += new
SerialDataReceivedEventHandler(DoorController_DataReceived);
            comPort.Open();
        }

        public void ClosePort()
        {
            if (comPort.IsOpen)
                comPort.Close();
        }

        // Open the door using the computer
        public void OpenDoorManually()
        {
            comPort.WriteLine("\x00");
            SendMessage(2, employeeName);
            isLocked = false;
        }

        // Open the door from RFID signal
        public void OpenDoor(int clockType, string employee)
        {
            //comPort.WriteLine("\x02");
            comPort.WriteLine(lockCommand[clockType]);
            SendMessage(clockType, employee);
            isLocked = false;
        }

        // Lock the door from no RFID signal
        public void LockDoor()
        {
            comPort.WriteLine("\x03");
            isLocked = true;
        }

        public void JustLockDoor()
        {
            comPort.WriteLine("\x08");
            isLocked = true;
        }

        // Send messages to LCD displays on the microcoller
        public void SendMessage(int clockType, string employee)
        {
            comPort.WriteLine(LCDcommand[clockType]);
            comPort.WriteLine(centerText(greeting[clockType]));
            comPort.WriteLine(centerText(employee));
```

```
            comPort.WriteLine(centerText(clockinout[clockType]));
            comPort.WriteLine(DateTime.Now.ToString("MM/dd/yy hh:mm:ss tt"));
        }

        // Clocked out action
        public void clockedOut(string employee)
        {
            comPort.WriteLine("\x03"); //Lock door
            comPort.WriteLine("\x04"); //Displays user message to LCD unit 1.
            //comPort.WriteLine("\x08");
            comPort.WriteLine(centerText(employee));
            comPort.WriteLine(centerText("Clocked Out"));
            comPort.WriteLine(DateTime.Now.ToString("MM/dd/yy hh:mm:ss tt"));
            isLocked = true;
        }

        // Clocked in action
        public void clockedIn(string employee)
        {
            comPort.WriteLine("\x03"); //Lock door
            comPort.WriteLine("\x05"); //Displays user message to LCD unit 2.
            //comPort.WriteLine("\x08");
            comPort.WriteLine(centerText(employee));
            comPort.WriteLine(centerText("Clocked In"));
            comPort.WriteLine(DateTime.Now.ToString("MM/dd/yy hh:mm:ss tt"));
            isLocked = true;
        }

        // Receive signal from microcontroller indicating that the door is
open or closed
        public void DoorController_DataReceived(object sender,
SerialDataReceivedEventArgs e)
        {
            int bytes = comPort.BytesToRead;
            byte[] comBuffer = new byte[bytes];
            comPort.Read(comBuffer, 0, bytes);
            isOpen = (comBuffer[0] > 0);
        }

        // Give GUI a handle for callbacks
        public SerialPort getComport()
        {
            return comPort;
        }

        // Check the status of the door
        public bool isDoorOpen()
        {
            return isOpen;
        }

        // Check the status of the door
        public bool isDoorLocked()
        {
            return isLocked;
        }
```

```
        // Pad text to center it on LCD display
        private string centerText(string text)
        {
            StringBuilder padleft = new StringBuilder();
            StringBuilder padright = new StringBuilder();
            padleft.Append("".PadLeft((20 - text.Length) / 2));
            padright.Append("".PadLeft(20 - (padleft.Length + text.Length)));
            return padleft + text + padright;
        }

        // Find the initial condition of the door
        public void getDoorStatus()
        {
            comPort.WriteLine("\x06");
        }
    }
}
```

Appendix C – ARMite Microcontroller Source

```
'================================
'
'     Senior Project ECE 4710
'
'     RFID Door Lock Controller
'           Microcontroller Program
'
'                 by
'
'           Fetah Basic
'                 and
'           Kenneth Dean
'
'================================

#include <Serial.bas>
#include <Control.bas>

'=============================================================================
'
'     Async Serial Interface --
'
'     required initialization code
'
'           copyright Coridium Corp, 2008,   may be used with Coridium
Hardware or licensees
'=============================================================================
dim i as integer
dim i as integer
dim i as integer
dim i as integer
dim i as integer
dim i as integer
dim i as integer
dim i as integer
dim i as integer
dim i as integer
dim i as integer
dim i as integer
dim i as integer
dim i as integer
dim i as integer
dim i as integer
dim i as integer
dim i as integer
dim i as integer
dim i as integer
dim i as integer
dim i as integer
dim i as integer
dim i as integer
dim i as integer
dim i as integer
dim i as integer
```

```
dim i as integer
dim i as integer
dim i as integer

#ifndef USE_SERIAL
#define USE_SERIAL

GOSUB doInitSerial                  ' if done inline this initializes
SERINtimeout
goto jumpAroundSerial

DIM BAUD(16)

doInitSerial:     ' by doing it this way the variables become global

SERINtimeout = 500000         ' this should be initialized
lastSERpin = 0                      ' last pin serial input from, this saves
time when restarting -- add 1 as 0 is initial value
lastBitTime = 0                     ' last bit time
lastNegTrue = 0                     ' set this to be 1 for negative true
serial -- default positive true

return

SUB InitSerial
      GOSUB doInitSerial
END SUB


#ifndef CR
      #define CR  chr(13)
      #define     LF    chr(10)
#endif


' bit banged serial input


FUNCTION RXD(pin)
      DIM  ch AS INTEGER
      DIM  i AS INTEGER
      DIM  start AS INTEGER

      if (lastSERpin -1 <> pin) then
            lastSERpin = pin+1
            if SERINtimeout = 0 then SERINtimeout = 500000  ' it may not be
initialized on the first call
            INPUT pin
            lastBitTime = ( 2000000 / BAUD(pin) + 1) >>1    ' round the value
up gets within 3% for 115Kb
      endif

      start = TIMER
      ' look for stop/idle
      i = SERINtimeout>>3
      if (lastNegTrue) then
            while IN(pin)
```

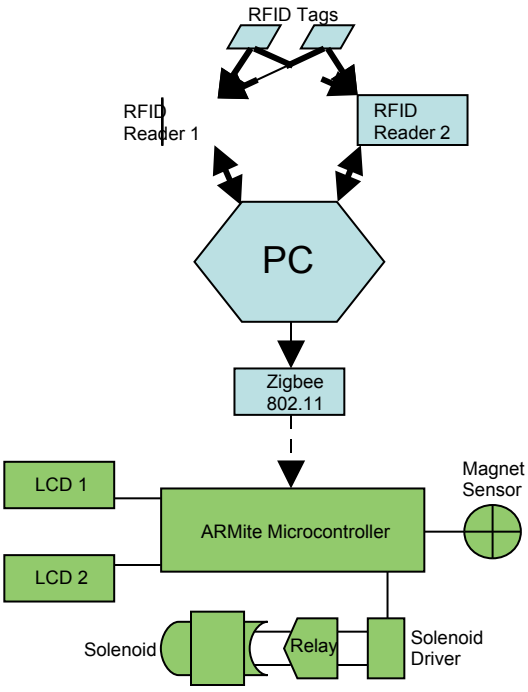
```
                  if (TIMER-start > i) then return -1
            loop
      else
            while IN(pin) = 0
                  if (TIMER-start > i) then return -1
            loop
      endif

      ' look for start bit
      if (lastNegTrue) then
            while IN(pin) = 0
                  if (TIMER-start > SERINtimeout) then return -1
            loop
      else
            while IN(pin)
                  if (TIMER-start > SERINtimeout) then return -1
            loop
      endif

      INTERRUPT 0                         ' takes 2.5 uSec
      start = TIMER-3                     ' compensate for startup time
      i = lastBitTime >> 1
      while ((TIMER-start) < i)     ' waits 1/2 baud time -- using the 1usec
ticker
      loop

      ch = 0
      start = TIMER
      i = 1
      while i < $100
            while ((TIMER - start) < lastBitTime)
            loop
            if IN(pin) then   ch = i or ch
            i = i << 1
            start = start + lastBitTime
      loop
      INTERRUPT 1

      if lastNegTrue then ch = ch xor 0xFF

      return ch
END FUNCTION


' if cnt = 0 then read until carriage-return or 0

FUNCTION SERIN (pin, bd, posTrue, INcnt, BYREF INlist as string)

      DIM  ch AS INTEGER
      DIM  timeout AS INTEGER
      DIM  j AS INTEGER

      if posTrue then lastNegTrue = 0 else lastNegTrue = 1
      BAUD(pin) = bd

      timeout = 0
      ch = 0
```

```
      j = 0
      do
            ch = RXD(pin)
            if ch = -1 then
                  INlist(j) = 255
                  j = j+1
                  timeout = -1
                  exit
            endif

            if (INcnt = 0) then
                  if ch = 0 then exit
 #ifndef TERMINATE_ON_0_ONLY
                  if CR = ch or LF = ch then exit
 #endif

            endif

            INlist(j) = ch
            j = j+1
      until j = INcnt

      INlist(j) = 0

      return timeout
END FUNCTION


' send a single character

SUB TXD(pin, ch)
      DIM  start AS INTEGER
      DIM  i AS INTEGER

      if (lastSERpin-1 <> pin) then
            lastSERpin = pin+1
            OUTPUT pin
            lastBitTime = ( 2000000 / BAUD(pin) + 1) >>1    ' round the value
up gets within 3% for 115Kb
      endif

      ch = (ch<<1) + $600           ' add start and stop bit  -- 2 stop bits
needed??, no but have to wait for 1 to go out
      if lastNegTrue then ch = not ch

      INTERRUPT 0

      i = 1
      start = TIMER

      while i < $800
            if(ch and i) then
                  HIGH pin
            else
                  LOW pin
            endif
```

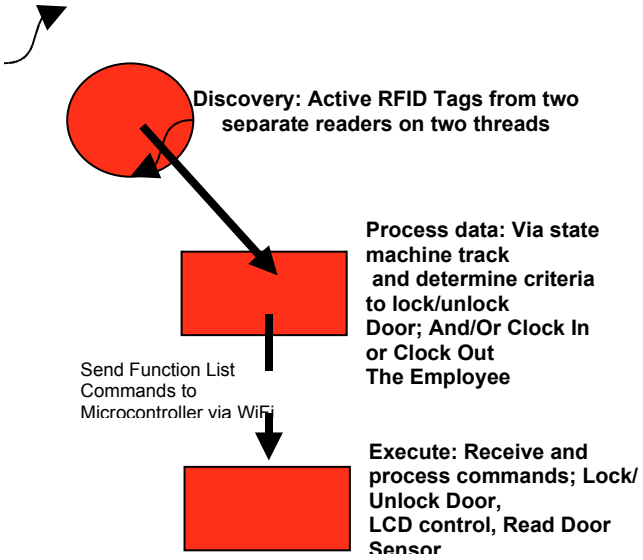
```
            i = i << 1

            while ((TIMER - start) < lastBitTime)
            loop
            start = start + lastBitTime
      loop

      INTERRUPT 1

END SUB

' if cnt = 0 then write until carriage-return or 0

SUB SEROUT( pin, bd, posTrue, cnt, BYREF s AS STRING)

      DIM  j AS INTEGER

      if posTrue then lastNegTrue = 0 else lastNegTrue = 1
      BAUD(pin) = bd

      j = 0

      do
            if cnt = 0 then
                  if s(j) = 0 then exit
            endif
            TXD(pin, s(j))

            j=j+1

      until j = cnt

END SUB

jumpAroundSerial:


#endif




'==================================
'
'     Senior Project ECE 4710
'
'     RFID Door Lock Controller
'           Microcontroller Program
'
'                 by
'
'           Fetah Basic
'                 and
'           Kenneth Dean
'
'==================================
```

```
#define low 0
#define high 1
#define door 0                                          'port for the door
#define ajar 1                                          'port for the status of
the door
#define led 15                                          'port for the LED
dim i as integer
dim counter as integer


main:
      output(led)                                       'set port to light LED
      output(door)                                      'set port to open the
door
      out(led) = high                                   'don't illuminate the
LED
      out(door) = high                                'don't open the door
      baud0(9600)                                       'set baud rate between
ZigBee and ARM7

start:
      while(rxd0 = -1)                                'listen for instructions
            if(timer > 1000) then                     'reduce the number of times
the status is updated
                  counter = counter + 1
                  timer = 0
                  if(counter > 1000) then
                        txd0(chr(out(ajar)))          'send status of the door
                        counter = 0
                  end if
            end if
      loop

      out(led) = low                                    'light the LED
      out(door) = low                                   'open the door
      for i = 0 to 30
            wait(100)                                   'delay so person can
open the door
            txd0(chr(out(ajar)))                      'send status of the door
      next i
      out(led) = high                                   'shut off LED
      out(door) = high                                'lock the door
      goto start
end
```

Appendix D – Block Diagrams



A.Figure 1 Hardware Block Diagram



A.Figure 2 Software Flow Diagram