# Program code

## Main form

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

namespace RFID
{
     //The key codes for the hardware buttons on the PDA
     public enum KeysHardware : int
     {
          Hardware1 = 193,
          Hardware2 = 194,
          Hardware3 = 195,
          Hardware4 = 196
     }

     public class MainForm : System.Windows.Forms.Form
     {
          //The design appearance
          private System.Windows.Forms.Label info;
          private System.Windows.Forms.Label systemInfo;
          private System.Windows.Forms.DataGrid dataGrid1;

          //The connection between the PDA and the RFID-reader
          private byte[] inputData;
          private Encoding enc;
          private PortManager port;
          private string displayString;
          private string message;
          private bool firstBatteryCheck;
          private bool lookingForTag;

          //The connection (PDA -- RFID); Message definition
          private const string batteryHigh = "À";
          private const string batteryLow = "Á";
```

```
private const string messageReceived = "OK";
private int count = 0;

//Hardware buttons
private MyMessageWindow messageWindow;
private bool tagFound;
private int buttonPressed;
private int write1counter, write2counter;

//Remove tag button
private System.Windows.Forms.Button removeTag;
private bool okToRemoveTag;
private bool erase;
private const int removeButton = 10;

//Search button
private System.Windows.Forms.Button search;
private int searchCount;
private const int searchButton = 11;
private bool searchTagFound;
private string searchObject;
private TagList searchList;
private TagList secondSearchList;
private bool wrong;
private string example;

//To keep button actions from using the same resort
private bool taken;
private bool seaking;

//The screen button in the search mode
private System.Windows.Forms.Button blank;
private System.Windows.Forms.Button a;
private System.Windows.Forms.Button d;
private System.Windows.Forms.Button g;
private System.Windows.Forms.Button j;
private System.Windows.Forms.Button m;
private System.Windows.Forms.Button p;
private System.Windows.Forms.Button t;
private System.Windows.Forms.Button x;
private System.Windows.Forms.Label buttonText;
private bool blankPressed;
private bool aPressed;
private bool dPressed;
private bool gPressed;
private bool jPressed;
```

```
private bool mPressed;
private bool pPressed;
private bool tPressed;
private bool xPressed;
private int knappLength;

//The recorder
Recorder theRecorder;
RecorderFlags flagStyle;
private IntPtr theWindow;
private IntPtr soundFile;

//The player
private Player thePlayer;

//The databases
private string taglist = "Program Files\\RFID\\";
private string track;
private string attribute;
private bool first;
private bool all;
private TagList list;
private TagList hemmet;
private TagList lund;
private int index;
private ListBox currentListBox;

//The appearance of the databases
private System.Windows.Forms.OpenFileDialog openFileDialog1;
private System.Windows.Forms.DataGridTableStyle tagStyle;
private System.Windows.Forms.DataGridTextBoxColumn ↙
dataGridTextBoxColumn1;
private System.Windows.Forms.DataGridTextBoxColumn ↙
dataGridTextBoxColumn2;
private System.Windows.Forms.DataGridTextBoxColumn ↙
dataGridTextBoxColumn3;
private System.Windows.Forms.DataGridTextBoxColumn ↙
dataGridTextBoxColumn4;
private System.Windows.Forms.ListBox theListBox;
private System.Windows.Forms.ListBox listBox0;
private System.Windows.Forms.ListBox listBox1;
private System.Windows.Forms.ListBox listBox00;
private System.Windows.Forms.ListBox listBox01;
private System.Windows.Forms.ListBox listBox02;
private System.Windows.Forms.ListBox listBox10;
private System.Windows.Forms.ListBox listBox11;
```

```
private System.Windows.Forms.ListBox listBox12;

//The timer
private System.Windows.Forms.Timer timer1;
private int tick;
private int letterCount;
private bool okToContinue;
private System.Windows.Forms.NumericUpDown changeIntervalBox;
private System.Windows.Forms.Button closeButton;
private System.Windows.Forms.Button changeIntervalButton;

public MainForm()
{
     InitializeComponent();

     //The connection between the PDA and the RFID-reader
     inputData = new byte[1];
     enc = Encoding.Default;
     port = new PortManager();
     port.DataReceived += new RFID.PortManager.CommEvent( ↙
     PortDataReceived);
     port.PortOpen("COM1:");
     displayString = "";
     firstBatteryCheck = true;
     lookingForTag = false;

     //Hardware buttons
     this.messageWindow = new MyMessageWindow(this);
     RegisterHKeys.RegisterRecordKey(this.messageWindow.Hwnd);
     tagFound = false;
     write1counter = 0;
     write2counter = 0;

     //Remove tag button
     okToRemoveTag = false;
     erase = false;

     //Search button
     searchCount = 0;
     searchTagFound = false;
     wrong = true;
     example = "";

     //To keep button actions from using the same resort
     taken = false;
     seaking = false;
```

```
//The screen button in the search mode
blankPressed = false;
aPressed = false;
dPressed = false;
gPressed = false;
jPressed = false;
mPressed = false;
pPressed = false;
tPressed = false;
xPressed = false;
knappLength = 0;
buttonText.Visible = false;

//The recorder
theRecorder = new Recorder();
flagStyle = RecorderFlags.VRS_NO_MOVE | ↙
RecorderFlags.VRS_NO_NOTIFY;
WindowInfo theWindowInfo = new WindowInfo();
theWindow = theWindowInfo.hWnd(null,this.Text.ToString());

//The player
thePlayer = new Player();
Player.PlaySound("\\Program Files\\uppspelning\\på.wav",0, ↙
(int) (int) PlayerFlags.SND_ASYNC);

//Opens the databases
first = false;
all = false;
list = new TagList();
hemmet = new TagList();
lund = new TagList();
lund = OpenList("lund.dat");
ChangeDatabase("hemmet");
hemmet = OpenList("hemmet.dat");

//The databases
index = 0;
track = "";
currentListBox = theListBox;
currentListBox.BringToFront();
currentListBox.SelectedIndex = index;
dataGrid1.DataSource = hemmet;

//The timer
tick = 0;
```

```
        letterCount = 0;
        okToContinue = true;
    }

    protected override void Dispose( bool disposing )
    {
        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()
    {
        System.Resources.ResourceManager resources = new ↙
        System.Resources.ResourceManager(typeof(MainForm));
        this.info = new System.Windows.Forms.Label();
        this.dataGrid1 = new System.Windows.Forms.DataGrid();
        this.tagStyle = new ↙
        System.Windows.Forms.DataGridTableStyle();
        this.dataGridTextBoxColumn1 = new ↙
        System.Windows.Forms.DataGridTextBoxColumn();
        this.dataGridTextBoxColumn2 = new ↙
        System.Windows.Forms.DataGridTextBoxColumn();
        this.dataGridTextBoxColumn3 = new ↙
        System.Windows.Forms.DataGridTextBoxColumn();
        this.dataGridTextBoxColumn4 = new ↙
        System.Windows.Forms.DataGridTextBoxColumn();
        this.removeTag = new System.Windows.Forms.Button();
        this.closeButton = new System.Windows.Forms.Button();
        this.systemInfo = new System.Windows.Forms.Label();
        this.openFileDialog1 = new ↙
        System.Windows.Forms.OpenFileDialog();
        this.theListBox = new System.Windows.Forms.ListBox();
        this.listBox0 = new System.Windows.Forms.ListBox();
        this.listBox1 = new System.Windows.Forms.ListBox();
        this.listBox00 = new System.Windows.Forms.ListBox();
        this.listBox01 = new System.Windows.Forms.ListBox();
        this.listBox02 = new System.Windows.Forms.ListBox();
        this.listBox10 = new System.Windows.Forms.ListBox();
        this.listBox11 = new System.Windows.Forms.ListBox();
        this.listBox12 = new System.Windows.Forms.ListBox();
        this.blank = new System.Windows.Forms.Button();
        this.a = new System.Windows.Forms.Button();
        this.d = new System.Windows.Forms.Button();
```

```
            this.g = new System.Windows.Forms.Button();
            this.j = new System.Windows.Forms.Button();
            this.m = new System.Windows.Forms.Button();
            this.p = new System.Windows.Forms.Button();
            this.t = new System.Windows.Forms.Button();
            this.x = new System.Windows.Forms.Button();
            this.timer1 = new System.Windows.Forms.Timer();
            this.buttonText = new System.Windows.Forms.Label();
            this.search = new System.Windows.Forms.Button();
            this.changeIntervalButton = new ↙
            System.Windows.Forms.Button();
            this.changeIntervalBox = new ↙
            System.Windows.Forms.NumericUpDown();
            //
            // info
            //
            this.info.Location = new System.Drawing.Point(176, 8);
            this.info.Size = new System.Drawing.Size(56, 64);
            //
            // dataGrid1
            //
            this.dataGrid1.Location = new System.Drawing.Point(16, 80);
            this.dataGrid1.Size = new System.Drawing.Size(208, 104);
            this.dataGrid1.TableStyles.Add(this.tagStyle);
            this.dataGrid1.Text = "dataGrid1";
            //
            // tagStyle
            //
this.tagStyle.GridColumnStyles.Add( ↙
            this.dataGridTextBoxColumn1);
            this.tagStyle.GridColumnStyles.Add( ↙
            this.dataGridTextBoxColumn2);
            this.tagStyle.GridColumnStyles.Add( ↙
            this.dataGridTextBoxColumn3);
            this.tagStyle.GridColumnStyles.Add( ↙
            this.dataGridTextBoxColumn4);
            this.tagStyle.MappingName = "tagList";
            //
            // dataGridTextBoxColumn1
            //
            this.dataGridTextBoxColumn1.HeaderText = "Nr";
            this.dataGridTextBoxColumn1.MappingName = "tagNumber";
            this.dataGridTextBoxColumn1.NullText = "(null)";
            this.dataGridTextBoxColumn1.Width = 52;
            //
            // dataGridTextBoxColumn2
```

```
//
this.dataGridTextBoxColumn2.HeaderText = "Ljud1";
this.dataGridTextBoxColumn2.MappingName = "tagSound";
this.dataGridTextBoxColumn2.NullText = "(null)";
this.dataGridTextBoxColumn2.Width = 33;
//
// dataGridTextBoxColumn3
//
this.dataGridTextBoxColumn3.HeaderText = "Ljud2";
this.dataGridTextBoxColumn3.MappingName = "tagSound2";
this.dataGridTextBoxColumn3.NullText = "(null)";
this.dataGridTextBoxColumn3.Width = 33;
//
// dataGridTextBoxColumn4
//
this.dataGridTextBoxColumn4.HeaderText = "Attribut";
this.dataGridTextBoxColumn4.MappingName = "attribute";
this.dataGridTextBoxColumn4.NullText = "(null)";
this.dataGridTextBoxColumn4.Width = 64;
//
// removeTag
//
this.removeTag.Location = new System.Drawing.Point(0, 255);
this.removeTag.Size = new System.Drawing.Size(120, 39);
this.removeTag.Text = "Ta bort tagg";
this.removeTag.Click += new ↙
System.EventHandler(this.RemoveTag_Click);
//
// closeButton
//
this.closeButton.Location = new System.Drawing.Point(8, 8);
this.closeButton.Size = new System.Drawing.Size(56, 24);
this.closeButton.Text = "Avsluta";
this.closeButton.Click += new ↙
System.EventHandler(this.CloseButton_Click);
//
// systemInfo
//
this.systemInfo.Location = new System.Drawing.Point(16, ↙
192);
this.systemInfo.Size = new System.Drawing.Size(200, 32);
//
// theListBox
//
this.theListBox.Items.Add("Hemmet");
this.theListBox.Items.Add("Lund");
```

```
this.theListBox.Items.Add("Alla");
this.theListBox.Location = new System.Drawing.Point(72, 8);
this.theListBox.Size = new System.Drawing.Size(96, 67);
//
// listBox0
//
this.listBox0.Items.Add("Klädesplagg");
this.listBox0.Items.Add("Kakburk");
this.listBox0.Items.Add("Pryl");
this.listBox0.Location = new System.Drawing.Point(72, 8);
this.listBox0.Size = new System.Drawing.Size(96, 67);
this.listBox0.Visible = false;
//
// listBox1
//
this.listBox1.Items.Add("Gata");
this.listBox1.Items.Add("Apotek");
this.listBox1.Items.Add("Mataffär");
this.listBox1.Location = new System.Drawing.Point(72, 8);
this.listBox1.Size = new System.Drawing.Size(96, 67);
this.listBox1.Visible = false;
//
// listBox00
//
this.listBox00.Items.Add("Röd");
this.listBox00.Items.Add("Blå");
this.listBox00.Items.Add("Grön");
this.listBox00.Items.Add("Vit");
this.listBox00.Items.Add("Svart");
this.listBox00.Items.Add("Gul");
this.listBox00.Items.Add("Rosa");
this.listBox00.Items.Add("Orange");
this.listBox00.Location = new System.Drawing.Point(72, 8);
this.listBox00.Size = new System.Drawing.Size(96, 67);
this.listBox00.Visible = false;
//
// listBox01
//
this.listBox01.Items.Add("Med nötter");
this.listBox01.Items.Add("Utan nötter");
this.listBox01.Location = new System.Drawing.Point(72, 8);
this.listBox01.Size = new System.Drawing.Size(96, 67);
this.listBox01.Visible = false;
//
// listBox02
//
```

```
this.listBox02.Items.Add("Mobiltelefon");
this.listBox02.Items.Add("Nycklar");
this.listBox02.Items.Add("Plånbok");
this.listBox02.Items.Add("Gerdakort");
this.listBox02.Location = new System.Drawing.Point(72, 8);
this.listBox02.Size = new System.Drawing.Size(96, 67);
this.listBox02.Visible = false;
//
// listBox10
//
this.listBox10.Items.Add("Lilla fiskaregatan");
this.listBox10.Items.Add("Stora södergatan");
this.listBox10.Items.Add("Stationsvägen");
this.listBox10.Location = new System.Drawing.Point(72, 8);
this.listBox10.Size = new System.Drawing.Size(96, 67);
this.listBox10.Visible = false;
//
// listBox11
//
this.listBox11.Items.Add("Hjorten");
this.listBox11.Items.Add("Svanen");
this.listBox11.Items.Add("Getingen");
this.listBox11.Location = new System.Drawing.Point(72, 8);
this.listBox11.Size = new System.Drawing.Size(96, 67);
this.listBox11.Visible = false;
//
// listBox12
//
this.listBox12.Items.Add("Ica tuna");
this.listBox12.Items.Add("Netto");
this.listBox12.Items.Add("AG\'s");
this.listBox12.Items.Add("Malmborgs");
this.listBox12.Location = new System.Drawing.Point(72, 8);
this.listBox12.Size = new System.Drawing.Size(96, 67);
this.listBox12.Visible = false;
//
// blank
//
this.blank.Enabled = false;
this.blank.Size = new System.Drawing.Size(80, 85);
this.blank.Text = "_ . - \'";
this.blank.Visible = false;
this.blank.Click += new ↙
System.EventHandler(this.blank_Click);
//
// a
```

```
//
this.a.Enabled = false;
this.a.Font = new System.Drawing.Font("Microsoft Sans ↙
Serif", 10F, System.Drawing.FontStyle.Bold);
this.a.Location = new System.Drawing.Point(80, 0);
this.a.Size = new System.Drawing.Size(80, 85);
this.a.Text = "ABCÅÄ";
this.a.Visible = false;
this.a.Click += new System.EventHandler(this.a_Click);
//
// d
//
this.d.Enabled = false;
this.d.Location = new System.Drawing.Point(160, 0);
this.d.Size = new System.Drawing.Size(80, 85);
this.d.Text = "DEF";
this.d.Visible = false;
this.d.Click += new System.EventHandler(this.d_Click);
//
// g
//
this.g.Enabled = false;
this.g.Location = new System.Drawing.Point(0, 85);
this.g.Size = new System.Drawing.Size(80, 85);
this.g.Text = "GHI";
this.g.Visible = false;
this.g.Click += new System.EventHandler(this.g_Click);
//
// j
//
this.j.Enabled = false;
this.j.Location = new System.Drawing.Point(80, 85);
this.j.Size = new System.Drawing.Size(80, 85);
this.j.Text = "JKL";
this.j.Visible = false;
this.j.Click += new System.EventHandler(this.j_Click);
//
// m
//
this.m.Enabled = false;
this.m.Location = new System.Drawing.Point(160, 85);
this.m.Size = new System.Drawing.Size(80, 85);
this.m.Text = "MNOÖ";
this.m.Visible = false;
this.m.Click += new System.EventHandler(this.m_Click);
//
```

```
// p
//
this.p.Enabled = false;
this.p.Location = new System.Drawing.Point(0, 170);
this.p.Size = new System.Drawing.Size(80, 85);
this.p.Text = "PQRS";
this.p.Visible = false;
this.p.Click += new System.EventHandler(this.p_Click);
//
// t
//
this.t.Enabled = false;
this.t.Location = new System.Drawing.Point(80, 170);
this.t.Size = new System.Drawing.Size(80, 85);
this.t.Text = "TUV";
this.t.Visible = false;
this.t.Click += new System.EventHandler(this.t_Click);
//
// x
//
this.x.Enabled = false;
this.x.Location = new System.Drawing.Point(160, 170);
this.x.Size = new System.Drawing.Size(80, 85);
this.x.Text = "WXYZ";
this.x.Visible = false;
this.x.Click += new System.EventHandler(this.x_Click);
//
// timer1
//
this.timer1.Interval = 400;
this.timer1.Tick += new ↙
System.EventHandler(this.Timer1_Tick);
//
// buttonText
//
this.buttonText.Font = new System.Drawing.Font("Microsoft ↙
Sans Serif", 10F, System.Drawing.FontStyle.Bold);
this.buttonText.Location = new System.Drawing.Point(81, ↙
231);
this.buttonText.Size = new System.Drawing.Size(77, 22);
//
// search
//
this.search.Location = new System.Drawing.Point(120, 255);
this.search.Size = new System.Drawing.Size(120, 39);
this.search.Text = "Sök";
```

```
this.search.Click += new ↙
System.EventHandler(this.Search_Click);
//
// changeIntervalButton
//
this.changeIntervalButton.Location = new ↙
System.Drawing.Point(176, 192);
this.changeIntervalButton.Size = new System.Drawing.Size( ↙
48, 20);
this.changeIntervalButton.Text = "ändra";
this.changeIntervalButton.Click += new ↙
System.EventHandler(this.ChangeIntervalButton_Click);
//
// changeIntervalBox
//
this.changeIntervalBox.Increment = new System.Decimal(new ↙
int[] {100, 0, 0,  0});
this.changeIntervalBox.Location = new ↙
System.Drawing.Point(112, 192);
this.changeIntervalBox.Maximum = new System.Decimal(new ↙
int[] {1000, 0, 0, 0});
this.changeIntervalBox.Minimum = new System.Decimal(new ↙
int[] {300, 0, 0, 0});
this.changeIntervalBox.Size = new System.Drawing.Size(56, ↙
20);
this.changeIntervalBox.Value = new System.Decimal(new ↙
int[] {300, 0, 0, 0});
//
// MainForm
//
this.BackColor = System.Drawing.Color.PowderBlue;
this.ClientSize = new System.Drawing.Size(240, 293);
this.Controls.Add(this.changeIntervalBox);
this.Controls.Add(this.changeIntervalButton);
this.Controls.Add(this.systemInfo);
this.Controls.Add(this.search);
this.Controls.Add(this.buttonText);
this.Controls.Add(this.closeButton);
this.Controls.Add(this.listBox12);
this.Controls.Add(this.listBox11);
this.Controls.Add(this.listBox10);
this.Controls.Add(this.listBox02);
this.Controls.Add(this.listBox00);
this.Controls.Add(this.listBox01);
this.Controls.Add(this.listBox1);
this.Controls.Add(this.listBox0);
```

```
        this.Controls.Add(this.theListBox);
        this.Controls.Add(this.removeTag);
        this.Controls.Add(this.dataGrid1);
        this.Controls.Add(this.info);
        this.Controls.Add(this.blank);
        this.Controls.Add(this.d);
        this.Controls.Add(this.g);
        this.Controls.Add(this.j);
        this.Controls.Add(this.m);
        this.Controls.Add(this.p);
        this.Controls.Add(this.t);
        this.Controls.Add(this.x);
        this.Controls.Add(this.a);
        this.Icon = ((System.Drawing.Icon)(resources.GetObject( ↙
        "$this.Icon")));
        this.Text = "Form1";

    }
    #endregion

    static void Main()
    {
        Application.Run(new MainForm());
    }

    //
    //Screen buttons
    //
    private void RemoveTag_Click(object sender, System.EventArgs e)
    {
        //If not in search mode
        if (!erase)
        {
            if(!taken)
            {
                taken = true;
                buttonPressed = removeButton;
                if(okToRemoveTag == false)
                {
                    lookingForTag = true;
                    SendData(130);
                }
                else
                {
                    TheButtonIsPressed();
                }
```

```
        }
    }
    //In search mode
    else if (wrong == true && buttonText.Text != string.Empty)
    {
        Player.PlaySound("\\Program Files\\uppspelning\\ ↙
        bip.wav",0,(int) (int) PlayerFlags.SND_ASYNC);
        buttonText.Text = buttonText.Text.Remove(( ↙
        buttonText.Text.Length-1), 1);
        this.Focus();
    }
    else if (buttonText.Text == string.Empty)
        this.Focus();
    else
    {
        wrong = true;
        switch(example)
        {
            case "mataffär":
                Player.PlaySound("\\Program Files\\ ↙
                uppspelning\\malmborgs.wav", 0,(int) (int) ↙
                PlayerFlags.SND_ASYNC);
                example = "malmborgs";
                wrong = false;
                break;

            case "svanen":
                Player.PlaySound("\\Program Files\\ ↙
                uppspelning\\svart.wav",0,(int) (int) ↙
                PlayerFlags.SND_ASYNC);
                example = "svart";
                wrong = false;
                break;

            case "gerdakort":
                Player.PlaySound("\\Program Files\\ ↙
                uppspelning\\getingen.wav",0,(int) (int) ↙
                PlayerFlags.SND_ASYNC);
                example = "getingen";
                wrong = false;
                break;

            case "stationsvägen":
                Player.PlaySound("\\Program Files\\ ↙
                uppspelning\\stora södergatan.wav",0, ↙
                (int) (int) PlayerFlags.SND_ASYNC);
```

```
                    example = "stora södergatan";
                    wrong = false;
                    break;

               default:
                    if (buttonText.Text != string.Empty)
                    {
                         Player.PlaySound("\\Program Files\\ ↙
                         uppspelning\\bip.wav",0,(int) (int) ↙
                         PlayerFlags.SND_ASYNC);
                         buttonText.Text = buttonText.Text. ↙
                         Remove((buttonText.Text.Length-1), 1);
                         this.Focus();
                    }
                    break;
          }
     }
}

//Button to search a specific tag
private void Search_Click(object sender, System.EventArgs e)
{
     if(seaking)
     {
          seaking = false;
          taken = false;
          this.Focus();
     }
     else if(!taken)
     {
          taken = true;
          TagList tem = list;
          ChangeDatabase("alla");
          searchList = list;
          list = tem;
          if (searchCount == 0)
          {
               Player.PlaySound("\\Program Files\\uppspelning\\ ↙
               sökläge.wav",0,(int) (int) PlayerFlags.SND_ASYNC);
               ShowButtons();
               this.Focus();
               taken = false;
          }
          else if(searchCount == 1)
          {
               if (wrong == false)
```

```
{
     buttonText.Text = example;
     wrong = true;
}
if(buttonText.Text == string.Empty)
{
     Player.PlaySound("\\Program Files\\ ↙
     uppspelning\\glomt_att_ange_attribut.wav", ↙
     0,(int) (int) PlayerFlags.SND_ASYNC);
     searchCount = 1;
     taken = false;
}
else
if(listBox00.Items.Contains((object)buttonText. ↙
Text[0].ToString().ToUpper()+buttonText.Text. ↙
Remove(0,1)) ↙
|| listBox01.Items.Contains((object)buttonText. ↙
Text[0].ToString().ToUpper()+buttonText.Text. ↙
Remove(0,1)) ↙
|| listBox02.Items.Contains((object)buttonText. ↙
Text[0].ToString().ToUpper()+buttonText.Text. ↙
Remove(0,1)) ↙
|| listBox10.Items.Contains((object)buttonText. ↙
Text[0].ToString().ToUpper()+buttonText.Text. ↙
Remove(0,1)) ↙
|| listBox11.Items.Contains((object)buttonText. ↙
Text[0].ToString().ToUpper()+buttonText.Text. ↙
Remove(0,1)) ↙
|| listBox12.Items.Contains((object)buttonText. ↙
Text[0].ToString().ToUpper()+buttonText.Text. ↙
Remove(0,1)))
{
     buttonPressed = searchButton;
     lookingForTag = true;
     seaking = true;
     searchObject = buttonText.Text[0]. ↙
     ToString().ToUpper()+buttonText.Text. ↙
     Remove(0,1);
     SendData(130);
}
else if(listBox0.Items.Contains((object) ↙
buttonText.Text[0].ToString().ToUpper()↙
+buttonText.Text.Remove(0,1)))
{
     buttonPressed = searchButton;
     lookingForTag = true;
```

```
                    seaking = true;
                    searchObject = null;
                    SendData(130);
                }
                else if(listBox1.Items.Contains((object) ↙
                buttonText.Text[0].ToString().ToUpper()↙
                +buttonText.Text.Remove(0,1)))
                {
                        buttonPressed = searchButton;
                        lookingForTag = true;
                        seaking = true;
                        searchObject = null;
                        SendData(130);
                }
                else if(theListBox.Items.Contains((object) ↙
                buttonText.Text[0].ToString().ToUpper()↙
                +buttonText.Text.Remove(0,1)))
                {
                        TagList temp = list;
                        ChangeDatabase(buttonText.Text);
                        secondSearchList = list;
                        list = temp;
                        buttonPressed = searchButton;
                        lookingForTag = true;
                        seaking = true;
                        searchObject = "Allt";
                        SendData(130);
                }
                else
                {
                    Player.PlaySound("\\Program Files\\ ↙
                    uppspelning\\ingen_tagg_med_attributet.wav", ↙
                    0,(int) (int) PlayerFlags.SND_ASYNC);
                    buttonText.Text = "";
                    searchCount = 1;
                    taken = false;
                }
            }
        }
    }

    //Ends the program
    private void CloseButton_Click(object sender, System.EventArgs e)
    {
        port.PortClose();
        Player.PlaySound("\\Program Files\\uppspelning\\ ↙
```

```
        av.wav",0,(int) (int) PlayerFlags.SND_ASYNC);

        //Saves the databases
        ChangeDatabase("hemmet");
        SaveList("hemmet.dat");
        ChangeDatabase("lund");
        SaveList("lund.dat");

        this.Close();
    }

    //
    //Hardware buttons
    //
    //This method is called from the class myMessageWindow if a
    hardware button is pressed
    public void ButtonPressed(int button)
    {
        if (button == (int)KeysHardware.Hardware3)
        {
            if(seaking)
            {
                seaking = false;
                systemInfo.Text = "Avbryter sökning!";
                Player.PlaySound("\\Program Files\\uppspelning\\ ↙
                avbryter_sökning.wav",0,(int) (int) ↙
                PlayerFlags.SND_ASYNC);
                if(buttonText.Visible)
                {
                    Player.PlaySound("\\Program Files\\ ↙
                    uppspelning\\startlage.wav",0,(int) (int) ↙
                    PlayerFlags.SND_ASYNC);
                    searchCount = 0;
                    HideButtons();
                    this.Focus();
                }
            }
            else if(!taken)
                seaking = true;
        }
        else if(seaking)
        {
            seaking = false;
            systemInfo.Text = "Avbryter sökning!";
            Player.PlaySound("\\Program Files\\uppspelning\\ ↙
```

```
                avbryter_sökning.wav",0,(int) (int) ↙
                PlayerFlags.SND_ASYNC);
                if(buttonText.Visible)
                {
                     Player.PlaySound("\\Program Files\\uppspelning\\ ↙
                     startlage.wav",0,(int) (int) ↙
                     PlayerFlags.SND_ASYNC);
                }
                write1counter = 0;
                okToRemoveTag = false;
                searchCount = 0;
                HideButtons();
                this.Focus();
           }
           if(!taken)
           {
                taken = true;
                buttonPressed = button;
                if(write1counter == 0 && write2counter == 0 && ↙
                okToRemoveTag == false && !buttonText.Visible)
                {
                     lookingForTag = true;
                     SendData(130);
                }
                else
                {
                     this.TheButtonIsPressed();
                }
           }
      }

      //
      //Actions that take place when a button is pressed
      //
      private void TheButtonIsPressed()
      {
           switch(buttonPressed)
           {
                case (int) KeysHardware.Hardware1:
                     systemInfo.Text = "Nr 1";
                     if(write1counter == 1 || okToRemoveTag == true ↙
                     || buttonText.Visible)
                     {
                          Player.PlaySound("\\Program Files\\ ↙
                          uppspelning\\startlage.wav",0,(int) (int) ↙
                          PlayerFlags.SND_ASYNC);
```

```
            write1counter = 0;
            okToRemoveTag = false;
            searchCount = 0;
            seaking = false;
            HideButtons();
            this.Focus();
        }
        else if(tagFound && write2counter == 0)
        {
            Tag temp = list.Find(message);
            if(temp == null)
            {
                if (!SearchOtherDatabases(theListBox. ↙
                SelectedItem.ToString().ToLower()))
                {
                    Player.PlaySound("\\Program ↙
                    Files\\uppspelning\\ej_inlagd. ↙
                    wav",0,(int) (int) ↙
                    PlayerFlags.SND_ASYNC);
                }
                write2counter = 0;
                this.Focus();
            }
            else if(temp.tagSound2 != "")
            {
                Player.PlaySound("\\Program ↙
                Files\\uppspelning\\har_beskrivning2. ↙
                wav",0,(int) (int) ↙
                PlayerFlags.SND_ASYNC);
                write2counter++;
            }
            else
            {
                Player.PlaySound("\\Program Files\\ ↙
                uppspelning\\spela_in_ ↙
                beskrivning.wav",0,(int) (int) ↙
                PlayerFlags.SND_ASYNC);
                write2counter++;
            }
        }
        else if(write2counter == 1)
        {
            Player.PlaySound("\\Program Files\\ ↙
            uppspelning\\spela_in.wav", ↙
            0,(int) PlayerFlags.SND_SYNC);
```

```
                soundFile = theRecorder.Create(flagStyle, ↙
                theWindow, 300, 300, "\\Program Files\\ ↙
                RFID\\taggljud\\"+message+"b.wav");
                theRecorder.Record(soundFile);
                write2counter++;
            }
            else if (write2counter == 2)
            {
                soundFile = theRecorder.Create(flagStyle, ↙
                theWindow, 300, 300, "\\Program Files\\ ↙
                RFID\\taggljud\\"+message+"b.wav");
                theRecorder.Close(soundFile);
                write2counter = 0;

                Tag temp = list.Find(message);
                list.Remove(temp);
                temp.changeTagSound2("b.wav");
                list.Add(temp);
                dataGrid1.DataSource = null;
                dataGrid1.DataSource = list;
                CurrencyManager cm = (CurrencyManager) ↙
                this.dataGrid1.BindingContext[list];
                if (cm != null)
                {
                    cm.Refresh();
                }
                Player.PlaySound("\\Program Files\\ ↙
                uppspelning\\sparad.wav",0,(int) (int) ↙
                PlayerFlags.SND_ASYNC);
                this.Focus();
            }
            else if(!tagFound)
            {
                this.Focus();
            }
            taken = false;
            break;

        case (int) KeysHardware.Hardware2:
            systemInfo.Text = "Nr 2";
            if(write2counter == 1 || okToRemoveTag == true ↙
            || buttonText.Visible)
            {
                Player.PlaySound("\\Program Files\\ ↙
                uppspelning\\startlage.wav",0,(int) ↙
                PlayerFlags.SND_ASYNC);
```

```
            write2counter = 0;
            okToRemoveTag = false;
            searchCount = 0;
            seaking = false;
            HideButtons();
            this.Focus();
        }
        else if(tagFound && write1counter == 0)
        {
            if(list.Find(message) != null)
            {
                Player.PlaySound("\\Program Files\\ ↙
                uppspelning\\fanns_redan.wav", ↙
                0,(int) PlayerFlags.SND_ASYNC);
                write1counter++;
            }
            else if (SearchOtherDatabases(theListBox. ↙
            SelectedItem.ToString().ToLower()))
            {
            }
            else
            {
                Player.PlaySound("\\Program Files\\ ↙
                uppspelning\\spela_in_beskrivning.wav", ↙
                0,(int) PlayerFlags.SND_ASYNC);
                write1counter++;
            }
        }
        else if (write1counter == 1)
        {
            Player.PlaySound("\\Program Files\\ ↙
            uppspelning\\spela_in.wav", ↙
            0,(int) PlayerFlags.SND_SYNC);
            soundFile = theRecorder.Create(flagStyle, ↙
            theWindow, 300, 300, "\\Program Files\\ ↙
            RFID\\taggljud\\"+message+"a.wav");
            theRecorder.Record(soundFile);
            write1counter++;
        }
        else if (write1counter == 2)
        {
            soundFile = theRecorder.Create(flagStyle, ↙
            theWindow, 300, 300, "\\Program Files\\ ↙
            RFID\\taggljud\\"+message+"a.wav");
            theRecorder.Close(soundFile);
            write1counter = 0;
```

```
                    Tag temp = list.Find(message);
                    list.Remove(temp);
                    attribute = currentListBox.SelectedItem. ↙
                    ToString();
                    if(temp == null || temp.tagSound2 == "")
                    {
                         list.Add(new Tag(message, "a.wav", ↙
                         attribute));
                    }
                    else
                    {
                         list.Add(new Tag(message, "a.wav", ↙
                         "b.wav", attribute));
                    }
                    dataGrid1.DataSource = null;
                    dataGrid1.DataSource = list;
                    CurrencyManager cm = (CurrencyManager) ↙
                    this.dataGrid1.BindingContext[list];
                    if (cm != null)
                    {
                         cm.Refresh();
                    }
                    Player.PlaySound("\\Program Files\\ ↙
                    uppspelning\\sparad.wav",0,(int) ↙
                    PlayerFlags.SND_ASYNC);
                    this.Focus();
               }
               else if(!tagFound)
               {
                    this.Focus();
               }
               taken = false;
               break;

          case (int) KeysHardware.Hardware3:
               systemInfo.Text = "Nr 3";
               if(write1counter == 1 || write2counter == 1 || ↙
               okToRemoveTag == true || buttonText.Visible)
               {
                    Player.PlaySound("\\Program Files\\ ↙
                    uppspelning\\startlage.wav", ↙
                    0,(int) PlayerFlags.SND_ASYNC);
                    write1counter = 0;
                    write2counter = 0;
                    okToRemoveTag = false;
                    searchCount = 0;
```

```
            seaking = false;
            HideButtons();
            taken = false;
            this.Focus();
        }
        else if(tagFound && list.Find(message) != null)
        {
            Tag t = list.Find(message);
            if(track == "" || IsInList(currentListBox. ↙
            SelectedItem.ToString(), t) || t.attribute ↙
            == currentListBox.SelectedItem.ToString())
            {
                Player.PlaySound("\\Program Files\\ ↙
                RFID\\taggljud\\"+message+"a.wav", ↙
                0,(int) PlayerFlags.SND_ASYNC);
                seaking = false;
                taken = false;
            }
            else if(seaking)
            {
                lookingForTag = true;
                SendData(130);
            }
        }
        else if(tagFound && all)
        {
            Player.PlaySound("\\Program Files\\ ↙
            uppspelning\\inget_ljud_pa_taggen.wav", ↙
            0,(int) PlayerFlags.SND_ASYNC);
            taken = false;
            this.Focus();
        }
        else
        {
            if(seaking)
            {
                lookingForTag = true;
                SendData(130);
            }
            else
            {
                taken = false;
            }
            this.Focus();
        }
        break;
```

```
case (int) KeysHardware.Hardware4:
     systemInfo.Text = "Nr 4";
     if(write1counter == 1 || write2counter == 1 ||
     okToRemoveTag == true)
     {
          Player.PlaySound("\\Program Files\\
          uppspelning\\startlage.wav",
          0,(int) PlayerFlags.SND_ASYNC);
          write1counter = 0;
          write2counter = 0;
          okToRemoveTag = false;
          searchCount = 0;
          seaking = false;
          HideButtons();
          this.Focus();
     }
     else if(buttonText.Visible && searchTagFound)
     {
          Tag temp = searchList.Find(message);
          if(temp != null && temp.tagSound2 == "")
          {
               Player.PlaySound("\\Program Files\\
               uppspelning\\inget_ljud_pa_taggen.wav",
               0,(int) PlayerFlags.SND_ASYNC);
               this.Focus();
          }
          else if(temp != null)
               Player.PlaySound("\\Program Files\\
               RFID\\taggljud\\"+message+"b.wav",
               0,(int) PlayerFlags.SND_ASYNC);
     }
     else if(tagFound && list.Find(message) != null)
     {
          Tag temp = list.Find(message);
          if(temp.tagSound2 == "")
          {
               Player.PlaySound("\\Program Files\\
               uppspelning\\inget_ljud_pa_taggen.wav",
               0,(int) PlayerFlags.SND_ASYNC);
               this.Focus();
          }
          else
          {
               Player.PlaySound("\\Program Files\\
               RFID\\taggljud\\"+message+"b.wav",
```

```
                    0,(int) PlayerFlags.SND_ASYNC);
               }
               this.Focus();
          }
          else if(tagFound)
          {
               SearchOtherDatabases(theListBox. ↙
               SelectedItem.ToString().ToLower());
               this.Focus();
          }
          else
          {
               this.Focus();
          }
          taken = false;
          break;

     case (int) removeButton:
          if(tagFound && list.Find(message) != null && ↙
          okToRemoveTag == false)
          {
               Tag t = list.Find(message);
               if(track == "" || IsInList(currentListBox. ↙
               SelectedItem.ToString(), t) || t.attribute ↙
               == currentListBox.SelectedItem.ToString()) ↙
               {
                    okToRemoveTag = true;
                    Player.PlaySound("\\Program Files\\ ↙
                    uppspelning\\hittade_tag.wav",
                    0,(int) PlayerFlags.SND_SYNC);
                    Player.PlaySound("\\Program Files\\ ↙
                    RFID\\taggljud\\"+message+"a.wav",
                    0,(int) PlayerFlags.SND_SYNC);
                    Player.PlaySound("\\Program Files\\ ↙
                    uppspelning\\ta_bort_fran_databas.wav",
                    0,(int) PlayerFlags.SND_ASYNC);
               }
               else
               {
                    Player.PlaySound("\\Program Files\\ ↙
                    uppspelning\\är_i_fel_meny.wav", ↙
                    0,(int) PlayerFlags.SND_ASYNC);
               }
          }
          else if(tagFound && okToRemoveTag == false)
          {
```

```
                    if(!SearchOtherDatabases(theListBox. ↙
                    SelectedItem.ToString().ToLower()))
                    {
                         Player.PlaySound("\\Program Files\\ ↙
                         uppspelning\\inget_ljud_pa_taggen.wav", ↙
                         0,(int) PlayerFlags.SND_ASYNC);
                         this.Focus();
                    }
               }
               else if (okToRemoveTag == true)
               {
                    okToRemoveTag = false;
                    File.Delete("\\Program Files\\RFID\\ ↙
                    taggljud\\"+message+"a.wav");
                    File.Delete("\\Program Files\\RFID\\ ↙
                    taggljud\\"+message+"b.wav");
                    Tag temp = (Tag) list.Find(message);
                    list.Remove(temp);
                    if(all)
                    {
                         hemmet.Remove(temp);
                         lund.Remove(temp);
                    }
                    dataGrid1.DataSource = null;
                    dataGrid1.DataSource = list;
                    Player.PlaySound("\\Program Files\\ ↙
                    uppspelning\\tagg_borttagen.wav", ↙
                    0,(int) PlayerFlags.SND_ASYNC);
                    this.Focus();
               }
               else
               {
                    this.Focus();
               }
               taken = false;
               break;

          case (int) searchButton :
               if(tagFound && searchList.Find(message) != null)
               {
                    Tag t = searchList.Find(message);
                    if(searchObject != null && t.attribute == ↙
                    searchObject)
                    {
                         Player.PlaySound("\\Program Files\\ ↙
                         RFID\\taggljud\\"+message+"a.wav", ↙
```

```
                0,(int) PlayerFlags.SND_ASYNC);
                searchTagFound = true;
                seaking = false;
                taken = false;
            }
            else if(searchObject == null && IsInList( ↙
            buttonText.Text[0].ToString().ToUpper() ↙
            +buttonText.Text.Remove(0,1),t))
            {
                Player.PlaySound("\\Program Files\\ ↙
                RFID\\taggljud\\"+message+"a.wav", ↙
                0,(int) PlayerFlags.SND_ASYNC);
                searchTagFound = true;
                seaking = false;
                taken = false;
            }
            else if(searchObject == "Allt" && ↙
            secondSearchList.Find(t.tagNumber) != null)
            {
                Player.PlaySound("\\Program Files\\ ↙
                RFID\\taggljud\\"+message+"a.wav", ↙
                0,(int) PlayerFlags.SND_ASYNC);
                searchTagFound = true;
                seaking = false;
                taken = false;
            }
            else if(seaking)
            {
                lookingForTag = true;
                searchTagFound = false;
                SendData(130);
            }
            else
            {
                searchTagFound = false;
                taken = false;
            }
        }
        else if(tagFound && searchObject == "Allt")
        {
            Player.PlaySound("\\Program Files\\ ↙
            uppspelning\\inget_ljud_pa_taggen.wav", ↙
            0,(int) PlayerFlags.SND_ASYNC);
            searchTagFound = false;
            taken = false;
        }
```

```
                else
                {
                    if(seaking)
                    {
                        lookingForTag = true;
                        searchTagFound = false;
                        SendData(130);
                    }
                    else
                    {
                        searchTagFound = false;
                        taken = false;
                    }
                }
                this.Focus();
                break;
            }
        }

        //
        //The scroll button
        //

        //Actions that take place when the scroll button is used
        protected override void OnKeyDown(KeyEventArgs keyg)
        {
            if(!taken && write1counter == 0 && write2counter == 0)
            {
                switch(keyg.KeyData)
                {
                    case Keys.Left:
                        systemInfo.Text = "LEFT";
                        if (buttonText.Visible)
                        {
                            for (int i = 0; i < buttonText.Text. ↙
                            Length; i++)
                            {
                                Player.PlaySound("\\Program Files ↙
                                \\uppspelning\\"+buttonText.Text. ↙
                                Substring(i,1)+".wav",0,(int) ↙
                                PlayerFlags.SND_SYNC);
                            }
                        }
                        else
                        {
                            if (track == "0" || track == "1")
```

```
                    currentListBox = theListBox;
               else if (track == "00" || track == "01" ↙
               || track == "02")
                    currentListBox = listBox0;
               else if (track == "10" || track == "11" ↙
               || track == "12")
                    currentListBox = listBox1;

               if (track.Length == 1)
                    index = int.Parse(track);
               else if (track.Length > 0)
                    index = int.Parse(track.Remove(0,1));
               currentListBox.Show();
               currentListBox.BringToFront();
               if (track.Length != 0)
                    track =track.Remove((track.Length- ↙
                    1),1);
               Player.PlaySound("\\Program Files\\ ↙
               uppspelning\\"+currentListBox. ↙
               SelectedItem.ToString().ToLower()+ ↙
               ".wav",0,(int) PlayerFlags.SND_ASYNC);
          }
          break;

     case Keys.Right:
          systemInfo.Text = "RIGHT";
          if (buttonText.Visible)
          {
               for (int i = 0; i < buttonText.Text. ↙
               Length; i++)
               {
                    Player.PlaySound("\\Program Files ↙
                    \\uppspelning\\"+buttonText.Text. ↙
                    Substring(i,1)+".wav",0,(int) ↙
                    PlayerFlags.SND_SYNC);
               }
          }
          else
          {
               if (track.Length < 2)
               {
                    if (theListBox.SelectedIndex != 2)
                    {
                         track = track + index. ↙
                         ToString();
                         index = 0;
```

```
                    SetCurrentListBox();
                    Player.PlaySound("\\Program
                    Files\\uppspelning\\"
                    +currentListBox.SelectedItem.
                    ToString().ToLower()+".wav",
                    0,(int) PlayerFlags.SND_ASYNC);
               }
          }
     }
     break;

case Keys.Down:
     systemInfo.Text = "DOWN";

     if (buttonText.Visible)
     {
          for (int i = 0; i < buttonText.Text.
          Length; i++)
          {
               Player.PlaySound("\\Program Files
               \\uppspelning\\"+buttonText.Text.
               Substring(i,1)+".wav",0,
               (int) PlayerFlags.SND_SYNC);
          }
     }
     else
     {
          if(index == (currentListBox.Items.
          Count-1))
               index = 0;
          else
               index++;

          SetCurrentListBox();
          Player.PlaySound("\\Program Files\\
          uppspelning\\"+currentListBox.
          SelectedItem.ToString().ToLower()
          +".wav", 0,(int) PlayerFlags.SND_ASYNC);
     }
     break;

case Keys.Up:
     systemInfo.Text = "UP";
     if (buttonText.Visible)
     {
          for (int i = 0; i < buttonText.Text.
```

```
                    Length; i++)
                    {
                        Player.PlaySound("\\Program Files ↙
                        \\uppspelning\\"+buttonText. ↙
                        Text.Substring(i,1)+".wav", ↙
                        0,(int) PlayerFlags.SND_SYNC);
                    }
                }
                else
                {
                    if(index == 0)
                        index = (currentListBox.Items. ↙
                        Count-1);
                    else
                        index--;

                    SetCurrentListBox();
                    Player.PlaySound("\\Program Files\\ ↙
                    uppspelning\\"+currentListBox. ↙
                    SelectedItem.ToString().ToLower() ↙
                    +".wav",0,(int) PlayerFlags.SND_ASYNC);
                }
                break;

            case Keys.Return:
                systemInfo.Text = "ENTER";
                if (buttonText.Visible)
                {
                    for (int i = 0; i < buttonText.Text. ↙
                    Length; i++)
                    {
                        Player.PlaySound("\\Program Files ↙
                        \\uppspelning\\"+buttonText. ↙
                        Text.Substring(i,1)+".wav", ↙
                        0,(int) PlayerFlags.SND_SYNC);
                    }
                }
                else
                {
                    SetCurrentListBox();
                    Player.PlaySound("\\Program Files\\ ↙
                    uppspelning\\"+currentListBox. ↙
                    SelectedItem.ToString().ToLower() ↙
                    +".wav",0,(int) PlayerFlags.SND_ASYNC);
                }
                break;
```

```
                default:
                    break;
            }
        }
    }

    //
    //The databases
    //

    //To change the listBox and/or databases
    private void SetCurrentListBox()
    {
        if(track == "")
        {
            currentListBox.SelectedIndex = index;
            ChangeDatabase(currentListBox.SelectedItem. ↙
            ToString().ToLower());
            dataGrid1.DataSource = list;
            dataGrid1.Refresh();
            dataGrid1.Show();
            dataGrid1.BringToFront();
        }
        else if (track == "0")
            currentListBox = listBox0;
        else if (track == "1")
            currentListBox = listBox1;
        else if (track == "00")
            currentListBox = listBox00;
        else if (track == "01")
            currentListBox = listBox01;
        else if (track == "02")
            currentListBox = listBox02;
        else if (track == "10")
            currentListBox = listBox10;
        else if (track == "11")
            currentListBox = listBox11;
        else if (track == "12")
            currentListBox = listBox12;

        currentListBox.Show();
        currentListBox.BringToFront();
        currentListBox.SelectedIndex = index;
    }
```

```
public void ChangeDatabase(string database)
{
     switch(database)
     {
          case "hemmet":
               all = false;
               list = hemmet;
               break;

          case "lund":
               all = false;
               list = lund;
               break;

          case "alla":
               all = true;
               TagList temp = new TagList();
               temp.CopyTo(hemmet, lund);
               list = temp;
               break;
     }
}

//To find out if the tag found belongs to another database than the active
public bool SearchOtherDatabases(string database)
{
     switch(database)
     {
          case "hemmet":
               list = lund;
               if(list.Find(message) != null)
               {
                    Player.PlaySound("\\Program Files\\uppspelning\\finns_redan_i_lund.wav", 0,(int) PlayerFlags.SND_ASYNC);
                    list = hemmet;
                    return true;
               }
               list = hemmet;
               return false;
               break;

          case "lund":
               list = hemmet;
               if(list.Find(message) != null)
```

```
                {
                    Player.PlaySound("\\Program Files\\ ↙
                    uppspelning\\finns_redan_i_hemmet.wav", ↙
                    0,(int) PlayerFlags.SND_ASYNC);
                    list = lund;
                    return true;
                }
                list = lund;
                return false;
                break;

            default:
                return false;
                break;
        }
    }

    //Finds out if a tag belongs to a subgroup of "string a"
    public bool IsInList(string a, Tag t)
    {
        switch(a)
        {
            case "Klädesplagg" :

                if(listBox00.Items.Contains((object)t.attribute) ↙
                || t.attribute == "Klädesplagg")
                    return true;
                else
                    return false;
                break;

            case "Kakburk" :
                if(listBox01.Items.Contains((object)t.attribute) ↙
                || t.attribute == "Kakburk")
                    return true;
                else
                    return false;
                break;

            case "Pryl" :
                if(listBox02.Items.Contains((object)t.attribute) ↙
                || t.attribute == "Pryl")
                    return true;
                else
                    return false;
                break;
```

```
            case "Gata" :
                if(listBox10.Items.Contains((object)t.attribute) ↙
                || t.attribute == "Gata")
                    return true;
                else
                    return false;
                break;

            case "Apotek" :
                if(listBox11.Items.Contains((object)t.attribute) ↙
                || t.attribute == "Apotek")
                    return true;
                else
                    return false;
                break;

            case "Mataffär" :
                if(listBox12.Items.Contains((object)t.attribute) ↙
                || t.attribute == "Mataffär")
                    return true;
                else
                    return false;
                break;

            default :
                return false;
                break;
        }
    }

    //Saves the databases to files
    private void SaveList(string filename)
    {
        string taglist_temp;
        taglist_temp = taglist.Insert(taglist.Length, filename);
        //If the files already exists
        if(!first)
        {
            Stream myStream = File.OpenWrite(taglist_temp);
            if(myStream != null)
            {
                BinaryWriter writer = new BinaryWriter(myStream);
                writer.Write(list.Count);
                foreach(Tag t in list)
                {
```

```
                    writer.Write(t.tagNumber);
                    writer.Write(t.tagSound);
                    writer.Write(t.attribute);
                    writer.Write(t.tagSound2);
               }
               writer.Close();
               myStream.Close();
          }
     }
     //If the files doesn't exists
     else
     {
          Stream myStream = new FileStream( @taglist_temp, ↙
          System.IO.FileMode.Create);
          if(myStream != null)
          {
               BinaryWriter writer = new BinaryWriter(myStream);
               writer.Write(list.Count);
               foreach(Tag t in list)
               {
                    writer.Write(t.tagNumber);
                    writer.Write(t.tagSound);
                    writer.Write(t.attribute);
                    writer.Write(t.tagSound2);
               }
               writer.Close();
               myStream.Close();
          }
     }
}

private TagList OpenList(string filename)
{
     openFileDialog1.FileName = taglist + filename;
     try
     {
          Stream myStream  = ↙
          File.OpenRead(openFileDialog1.FileName);
          if(myStream != null)
          {
               BinaryReader reader = new BinaryReader(myStream);
               int iCount = reader.ReadInt32();
               for(int x=0; x < iCount; x++)
               {
                    Tag t = new Tag(reader.ReadString(),↙
                    reader.ReadString(), reader.ReadString());
```

```
                    String temp = reader.ReadString();
                    t.changeTagSound2(temp);
                    list.Add(t);
               }
               reader.Close();
               myStream.Close();
               if (list.Count == 0)
               {
               }
               else
               {
                    dataGrid1.TableStyles.Clear();
                    dataGrid1.TableStyles.Add(tagStyle);
                    dataGrid1.DataSource = list;
                    dataGrid1.Refresh();
                    dataGrid1.Show();
                    dataGrid1.BringToFront();
               }
          }
     }
     //If there was no files to open
     catch(FileNotFoundException)
     {
          first = true;
          list = new TagList();
          if (list.Count == 0)
          {
          }
          else
          {
               dataGrid1.TableStyles.Clear();
               dataGrid1.TableStyles.Add(tagStyle);
               dataGrid1.DataSource = list;
          }
     }
     return list;
}

//
//The connection between the PDA and the RFID-reader
//

//Is called from the class PortManager if data is received from ↙
the RFID-reader
private void PortDataReceived()
{
```

```
inputData = port.Input;
displayString = enc.GetString(inputData, 0, ↙
inputData.Length);

message = string.Empty;
systemInfo.Text += displayString + " ";
message = displayString;
displayString = string.Empty;

systemInfo.Text = " ";
systemInfo.Text = "Data received: ";
switch(message)
{
     case batteryHigh :
          if(firstBatteryCheck)
          {
               firstBatteryCheck = false;
               taken = false;
          }
          if(lookingForTag)
          {
               info.Text = "HITTAR INGEN TAG";
               lookingForTag = false;
               count = 0;
               this.Focus();
               this.TheButtonIsPressed();
          }
          break;

     case batteryLow :
          if(firstBatteryCheck)
          {
               Player.PlaySound("\\Program Files\\ ↙
               uppspelning\\ladda_batteriet.wav", ↙
               0,(int) PlayerFlags.SND_ASYNC);
               firstBatteryCheck = false;
               taken = false;
          }

          if(lookingForTag)
          {
               Player.PlaySound("\\Program Files\\ ↙
               uppspelning\\ladda_batteriet.wav", ↙
               0,(int) PlayerFlags.SND_ASYNC);
               info.Text = "HITTAR INGEN TAG";
               lookingForTag = false;
```

```
                    count = 0;
                    this.Focus();
                    this.TheButtonIsPressed();
               }
               break;

          case messageReceived :
               break;

          default :
               if(lookingForTag && count == 1)
               {
                    info.Text = "TAG FUNNEN: ";
                    lookingForTag = false;
                    count = 0;
                    tagFound = true;
               }
               else if(lookingForTag)
               {
                    count++;
               }
               if(tagFound && count == 1)
               {
                    message = message.Remove(message.Length-1, 1);
                    info.Text ="TAG FUNNEN: "+ message;
                    count = 0;
                    this.TheButtonIsPressed();
                    tagFound = false;
               }
               else if(tagFound)
               {
                    count++;
               }
               break;
     }
}

private void SendData(int data)
{
     byte[] outputData = new byte[1];
     outputData[0] = Convert.ToByte(data);
     systemInfo.Text = outputData[0].ToString();
     port.PortWrite(outputData);
     systemInfo.Text = "Har sent data";
}
```

```
//
//The timer
//

//A timer used to the screen buttons in the search mode
private void Timer1_Tick(object sender, System.EventArgs e)
{
     tick++;
     //After the time delay the letter pressed is printed on ↙
     the screen
     if (tick == 1)
     {
          if (aPressed)
          {
               aPressed = false;
               if (letterCount == 1)
                    buttonText.Text = buttonText.Text + "a";
               if (letterCount == 2)
                    buttonText.Text = buttonText.Text + "b";
               if (letterCount == 3)
                    buttonText.Text = buttonText.Text + "c";
               if (letterCount == 4)
                    buttonText.Text = buttonText.Text + "å";
               if (letterCount == 5)
                    buttonText.Text = buttonText.Text + "ä";
          }
          else if (dPressed)
          {
               dPressed = false;
               if (letterCount == 1)
                    buttonText.Text = buttonText.Text + "d";

               if (letterCount == 2)
                    buttonText.Text = buttonText.Text + "e";
               if (letterCount == 3)
                    buttonText.Text = buttonText.Text + "f";
          }
          else if (gPressed)
          {
               gPressed = false;
               if (letterCount == 1)
                    buttonText.Text = buttonText.Text + "g";
               if (letterCount == 2)
                    buttonText.Text = buttonText.Text + "h";
               if (letterCount == 3)
                    buttonText.Text = buttonText.Text + "i";
```

```
}
else if (jPressed)
{
     jPressed = false;
     if (letterCount == 1)
          buttonText.Text = buttonText.Text + "j";
     if (letterCount == 2)
          buttonText.Text = buttonText.Text + "k";
     if (letterCount == 3)
          buttonText.Text = buttonText.Text + "l";
}
else if (mPressed)
{
     mPressed = false;
     if (letterCount == 1)
          buttonText.Text = buttonText.Text + "m";
     if (letterCount == 2)
          buttonText.Text = buttonText.Text + "n";
     if (letterCount == 3)
          buttonText.Text = buttonText.Text + "o";
     if (letterCount == 4)
          buttonText.Text = buttonText.Text + "ö";
}
else if (pPressed)
{
     pPressed = false;
     if (letterCount == 1)
          buttonText.Text = buttonText.Text + "p";
     if (letterCount == 2)
          buttonText.Text = buttonText.Text + "q";
     if (letterCount == 3)
          buttonText.Text = buttonText.Text + "r";
     if (letterCount == 4)
          buttonText.Text = buttonText.Text + "s";
}
else if (tPressed)
{
     tPressed = false;
     if (letterCount == 1)
          buttonText.Text = buttonText.Text + "t";
     if (letterCount == 2)
          buttonText.Text = buttonText.Text + "u";
     if (letterCount == 3)
          buttonText.Text = buttonText.Text + "v";
}
else if (xPressed)
```

```
{
     xPressed = false;
     if (letterCount == 1)
          buttonText.Text = buttonText.Text + "w";
     if (letterCount == 2)
          buttonText.Text = buttonText.Text + "x";
     if (letterCount == 3)
          buttonText.Text = buttonText.Text + "y";
     if (letterCount == 4)
          buttonText.Text = buttonText.Text + "z";
}
else if (blankPressed)
{
     blankPressed = false;
     if (letterCount == 1)
          buttonText.Text = buttonText.Text + " ";
     if (letterCount == 2)
          buttonText.Text = buttonText.Text + ".";
     if (letterCount == 3)
          buttonText.Text = buttonText.Text + "-";
     if (letterCount == 4)
          buttonText.Text = buttonText.Text + "'";
}
Player.PlaySound("\\Program Files\\uppspelning\\ ↙
"+buttonText.Text.Remove(0,(buttonText.Text. ↙
Length- 1))+".wav",0,(int) PlayerFlags.SND_SYNC);
letterCount = 0;
timer1.Enabled = false;
tick = 0;
//When two letters has been pressed a proposition is ↙
given to the word being typed
if (buttonText.Text.Length == 2)
{
     switch(buttonText.Text)
     {
          case "he":
               Player.PlaySound("\\Program Files\\ ↙
               uppspelning\\hemmet.wav", ↙
               0,(int) PlayerFlags.SND_ASYNC);
               example = "hemmet";
               wrong = false;
               break;

          case "lu":
               Player.PlaySound("\\Program Files\\ ↙
               uppspelning\\lund.wav", ↙
```

```
                0,(int) PlayerFlags.SND_ASYNC);
                example = "lund";
                wrong = false;
                break;

            case "al":
                Player.PlaySound("\\Program Files\\ ↙
                uppspelning\\alla.wav", ↙
                0,(int) PlayerFlags.SND_ASYNC);
                example = "alla";
                wrong = false;
                break;

            case "kl":
                Player.PlaySound("\\Program Files\\ ↙
                uppspelning\\klädesplagg.wav", ↙
                0,(int) PlayerFlags.SND_ASYNC);
                example = "klädesplagg";
                wrong = false;
                break;

            case "ka":
                Player.PlaySound("\\Program Files\\ ↙
                uppspelning\\kakburk.wav", ↙
                0,(int) PlayerFlags.SND_ASYNC);
                example = "kakburk";
                wrong = false;
                break;

            case "pr":
                Player.PlaySound("\\Program Files\\ ↙
                uppspelning\\pryl.wav", ↙
                0,(int) PlayerFlags.SND_ASYNC);
                example = "pryl";
                wrong = false;
                break;

            case "ga":
                Player.PlaySound("\\Program Files\\ ↙
                uppspelning\\gata.wav", ↙
                0,(int) PlayerFlags.SND_ASYNC);
                example = "gata";
                wrong = false;
                break;

            case "ap":
```

```
            Player.PlaySound("\\Program Files\\ ↙
            uppspelning\\apotek.wav", ↙
            0,(int) PlayerFlags.SND_ASYNC);
            example = "apotek";
            wrong = false;
            break;

        case "ma":
            Player.PlaySound("\\Program Files\\ ↙
            uppspelning\\mataffär.wav", ↙
            0,(int) PlayerFlags.SND_ASYNC);
            example = "mataffär";
            wrong = false;
            break;

        case "rö":
            Player.PlaySound("\\Program Files\\ ↙
            uppspelning\\röd.wav", ↙
            0,(int) PlayerFlags.SND_ASYNC);
            example = "röd";
            wrong = false;
            break;

        case "bl":
            Player.PlaySound("\\Program Files\\ ↙
            uppspelning\\blå.wav", ↙
            0,(int) PlayerFlags.SND_ASYNC);
            example = "blå";
            wrong = false;
            break;

        case "gr":
            Player.PlaySound("\\Program Files\\ ↙
            uppspelning\\grön.wav", ↙
            0,(int) PlayerFlags.SND_ASYNC);
            example = "grön";
            wrong = false;
            break;

        case "vi":
            Player.PlaySound("\\Program Files\\ ↙
            uppspelning\\vit.wav", ↙
            0,(int) PlayerFlags.SND_ASYNC);
            example = "vit";
            wrong = false;
            break;
```

```
case "sv":
     Player.PlaySound("\\Program Files\\ ↙
     uppspelning\\svanen.wav", ↙
     0,(int) PlayerFlags.SND_ASYNC);
     example = "svanen";
     wrong = false;
     break;

case "ro":
     Player.PlaySound("\\Program Files\\ ↙
     uppspelning\\rosa.wav", ↙
     0,(int) PlayerFlags.SND_ASYNC);
     example = "rosa";
     wrong = false;
     break;

case "gu":
     Player.PlaySound("\\Program Files\\ ↙
     uppspelning\\gul.wav", ↙
     0,(int) PlayerFlags.SND_ASYNC);
     example = "gul";
     wrong = false;
     break;

case "or":
     Player.PlaySound("\\Program Files\\ ↙
     uppspelning\\orange.wav", ↙
     0,(int) PlayerFlags.SND_ASYNC);
     example = "orange";
     wrong = false;
     break;

case "me":
     Player.PlaySound("\\Program Files\\ ↙
     uppspelning\\med nötter.wav", ↙
     0,(int) PlayerFlags.SND_ASYNC);
     example = "med nötter";
     wrong = false;
     break;

case "ut":
     Player.PlaySound("\\Program Files\\ ↙
     uppspelning\\utan nötter.wav", ↙
     0,(int) PlayerFlags.SND_ASYNC);
     example = "utan nötter";
```

```
                wrong = false;
                break;

            case "mo":
                Player.PlaySound("\\Program Files\\ ↙
                uppspelning\\mobiltelefon.wav", ↙
                0,(int) PlayerFlags.SND_ASYNC);
                example = "mobiltelefon";
                wrong = false;
                break;

            case "ny":
                Player.PlaySound("\\Program Files\\ ↙
                uppspelning\\nycklar.wav", ↙
                0,(int) PlayerFlags.SND_ASYNC);
                example = "nycklar";
                wrong = false;
                break;

            case "pl":
                Player.PlaySound("\\Program Files\\ ↙
                uppspelning\\plånbok.wav", ↙
                0,(int) PlayerFlags.SND_ASYNC);
                example = "plånbok";
                wrong = false;
                break;

            case "ge":
                Player.PlaySound("\\Program Files\\ ↙
                uppspelning\\gerdakort.wav", ↙
                0,(int) PlayerFlags.SND_ASYNC);
                example = "gerdakort";
                wrong = false;
                break;

            case "li":
                Player.PlaySound("\\Program Files\\ ↙
                uppspelning\\lilla fiskaregatan.wav", ↙
                0,(int) PlayerFlags.SND_ASYNC);
                example = "lilla fiskaregatan";
                wrong = false;
                break;

            case "st":
                Player.PlaySound("\\Program Files\\ ↙
                uppspelning\\stationsvägen.wav", ↙
```

```
                    0,(int) PlayerFlags.SND_ASYNC);
                    example = "stationsvägen";
                    wrong = false;
                    break;

               case "hj":
                    Player.PlaySound("\\Program Files\\ ↙
                    uppspelning\\hjorten.wav", ↙
                    0,(int) PlayerFlags.SND_ASYNC);
                    example = "hjorten";
                    wrong = false;
                    break;

               case "ic":
                    Player.PlaySound("\\Program Files\\ ↙
                    uppspelning\\ica tuna.wav", ↙
                    0,(int) PlayerFlags.SND_ASYNC);
                    example = "ica tuna";
                    wrong = false;
                    break;

               case "ne":
                    example = "netto";
                    wrong = false;
                    Player.PlaySound("\\Program Files\\ ↙
                    uppspelning\\netto.wav", ↙
                    0,(int) PlayerFlags.SND_ASYNC);
                    break;

               case "ag":
                    example = "ag's";
                    wrong = false;
                    Player.PlaySound("\\Program Files\\ ↙
                    uppspelning\\ag's.wav", ↙
                    0,(int)(int) PlayerFlags.SND_ASYNC);
                    break;
          }
     }
     if (timer1.Interval == 1)
     {
          timer1.Interval = int.Parse( ↙
          changeIntervalBox.Text);
          okToContinue = true;
          letterCount = 1;
     }
}
```

```
}

//Starts the timer with a short delay
private void FastClick()
{
     okToContinue = false;
     timer1.Enabled = false;
     timer1.Interval = 1;
     tick = 0;
     timer1.Enabled = true;
}

//Starts the timer with a long delay
private void SlowClick()
{
     timer1.Enabled = false;
     tick = 0;
     timer1.Enabled = true;
     letterCount++;
     if (letterCount == (knappLength + 1))
          letterCount = 1;
}

//To change the timer delay manually
private void ChangeIntervalButton_Click(object sender, ↙
System.EventArgs e)
{
     timer1.Interval = int.Parse(changeIntervalBox.Text);
     this.Focus();
}

//
//The screen button in the search mode
//

//Button to generate the letters a, b, c, å and ä
private void a_Click(object sender, System.EventArgs e)
{
     knappLength = a.Text.Length;
     if (dPressed || gPressed || jPressed || mPressed ↙
     || pPressed || tPressed || xPressed || blankPressed)
          FastClick();
     else if (okToContinue)
     {
          aPressed = true;
          SlowClick();
```

```
        }
        this.Focus();
    }

    //Button to generate the letters d, e and f
    private void d_Click(object sender, System.EventArgs e)
    {
        knappLength = d.Text.Length;
        if (aPressed || gPressed || jPressed || mPressed ↙
        || pPressed || tPressed || xPressed || blankPressed)
            FastClick();
        else if (okToContinue)
        {
            dPressed = true;
            SlowClick();
        }
        this.Focus();
    }

    //Button to generate the letters g, h and i
    private void g_Click(object sender, System.EventArgs e)
    {
        knappLength = g.Text.Length;
        if (aPressed || dPressed || jPressed || mPressed ↙
        || pPressed || tPressed || xPressed || blankPressed)
            FastClick();
        else if (okToContinue)
        {
            gPressed = true;
            SlowClick();
        }
        this.Focus();
    }

    //Button to generate the letters j, k and l
    private void j_Click(object sender, System.EventArgs e)
    {
        knappLength = j.Text.Length;
        if (aPressed || dPressed || gPressed || mPressed ↙
        || pPressed || tPressed || xPressed || blankPressed)
            FastClick();
        else if (okToContinue)
        {
            jPressed = true;
            SlowClick();
        }
```

```
        this.Focus();
    }

    //Button to generate the letters m, n, o and ö
    private void m_Click(object sender, System.EventArgs e)
    {
        knappLength = m.Text.Length;
        if (aPressed || dPressed || gPressed || jPressed ↙
        || pPressed || tPressed || xPressed || blankPressed)
            FastClick();
        else if (okToContinue)
        {
            mPressed = true;
            SlowClick();
        }
        this.Focus();
    }

    //Button to generate the letters p, q, r and s
    private void p_Click(object sender, System.EventArgs e)
    {
        knappLength = p.Text.Length;
        if (aPressed || dPressed || gPressed || jPressed ↙
        || mPressed || tPressed || xPressed || blankPressed)
            FastClick();
        else if (okToContinue)
        {
            pPressed = true;
            SlowClick();
        }
        this.Focus();
    }

    //Button to generate the letters t, u and v
    private void t_Click(object sender, System.EventArgs e)
    {
        knappLength = t.Text.Length;
        if (aPressed || dPressed || gPressed || jPressed ↙
        || mPressed || pPressed || xPressed || blankPressed)
            FastClick();
        else if (okToContinue)
        {
            tPressed = true;
            SlowClick();
        }
        this.Focus();
```

```
}

//Button to generate the letters x, y and z
private void x_Click(object sender, System.EventArgs e)
{
     knappLength = x.Text.Length;
     if (aPressed || dPressed || gPressed || jPressed ↙
     || mPressed || pPressed || tPressed || blankPressed)
          FastClick();
     else if (okToContinue)
     {
          xPressed = true;
          SlowClick();
     }
     this.Focus();
}

//Button to generate the symbols 'blanksteg', 'punkt', ↙
'bindestreck' and 'apostrof'
private void blank_Click(object sender, System.EventArgs e)
{
     knappLength = blank.Text.Length;
     if (aPressed  || dPressed || gPressed || jPressed ↙
     || mPressed || pPressed || tPressed || xPressed)
          FastClick();
     else if (okToContinue)
     {
          blankPressed = true;
          SlowClick();
     }
     this.Focus();
}

//Shows the screen buttons when search mode is entered
private void ShowButtons()
{
     searchCount++;
     erase = true;
     changeIntervalButton.Enabled = false;
     removeTag.Text = "Sudda";

     a.Show();
     a.Enabled = true;
     a.BringToFront();
     d.Show();
     d.Enabled = true;
```

```
        d.BringToFront();
        g.Show();
        g.Enabled = true;
        g.BringToFront();
        j.Show();
        j.Enabled = true;
        j.BringToFront();
        m.Show();
        m.Enabled = true;
        m.BringToFront();
        p.Show();
        p.Enabled = true;
        p.BringToFront();
        t.Show();
        t.Enabled = true;
        t.BringToFront();
        x.Show();
        x.Enabled = true;
        x.BringToFront();
        blank.Show();
        blank.Enabled = true;
        blank.BringToFront();

        buttonText.Visible = true;
        buttonText.Show();
        buttonText.Enabled = true;
        buttonText.BringToFront();
        buttonText.Text = "";
    }

    //Hides the screen buttons when search mode is entered
    private void HideButtons()
    {
        searchCount = 0;
        erase = false;
        changeIntervalButton.Enabled = true;
        removeTag.Text = "Ta bort Tagg";

        a.Hide();
        a.Enabled = false;
        d.Hide();
        d.Enabled = false;
        g.Hide();
        g.Enabled = false;
        j.Hide();
        j.Enabled = false;
```

```
                m.Hide();
                m.Enabled = false;
                p.Hide();
                p.Enabled = false;
                t.Hide();
                t.Enabled = false;
                x.Hide();
                x.Enabled = false;
                blank.Hide();
                blank.Enabled = false;

                buttonText.Visible = false;
                buttonText.Hide();
                buttonText.Enabled = false;
                this.Focus();
            }
        }
    }
```

# MyMessageWindow class

```
using System;
using Microsoft.WindowsCE.Forms;
using System.Windows.Forms;

namespace RFID
{
     public class MyMessageWindow : MessageWindow
     {
          //The code for the hardware buttons
          public const int WM_HOTKEY = 0x0312;
          MainForm form;

          //Constructor, which connects the message window to the main ↙
          program
          public MyMessageWindow(MainForm form)
          {
               this.form = form;
          }

          //Listens to the internal messages and sends the messages from ↙
          the hardware buttons to the main program
          protected override void WndProc(ref Message msg)
          {
               switch(msg.Msg)
               {
                    case WM_HOTKEY:
                         form.ButtonPressed(msg.WParam.ToInt32());
                         return;
               }
               base.WndProc(ref msg);
          }
     }
}
```

# RegisterHKeys class

```
using System;
using System.Runtime.InteropServices;

namespace RFID
{
     public enum KeyModifiers
     {
          Windows = 8 //Key modifier for Windows
     }
     //Uses the two methods RegisterHotKey and UnregisterFunction1 in
     the coredll.dll file to get control over the four hardware buttons
     on the PDA
     public class RegisterHKeys
     {
          public static void RegisterRecordKey(IntPtr hWnd)
          {
               //Write2 button
               UnregisterFunc1(KeyModifiers.Windows,
               (int)KeysHardware.Hardware1);
               RegisterHotKey(hWnd, (int) KeysHardware.Hardware1,
               KeyModifiers.Windows, (int) KeysHardware.Hardware1);

               //Write1 button
               UnregisterFunc1(KeyModifiers.Windows,
               (int)KeysHardware.Hardware2);
               RegisterHotKey(hWnd, (int) KeysHardware.Hardware2,
               KeyModifiers.Windows, (int) KeysHardware.Hardware2);

               //Read1 button
               UnregisterFunc1(KeyModifiers.Windows,
               (int)KeysHardware.Hardware3);
               RegisterHotKey(hWnd, (int) KeysHardware.Hardware3,
               KeyModifiers.Windows, (int) KeysHardware.Hardware3);

               //Read2 button
               UnregisterFunc1(KeyModifiers.Windows,
               (int)KeysHardware.Hardware4);
               RegisterHotKey(hWnd, (int) KeysHardware.Hardware4,
               KeyModifiers.Windows, (int) KeysHardware.Hardware4);
          }

          [DllImport("coredll.dll", SetLastError=true)]
          public static extern bool RegisterHotKey(
```

```
            IntPtr hWnd, // handle to window
            int id, // hot key identifier
            KeyModifiers Modifiers, // key-modifier options
            int key //virtual-key code
            );

        [DllImport("coredll.dll")]
        private static extern bool UnregisterFunc1(KeyModifiers ↙
        modifiers, int keyID);
    }
}
```

# Player class

```
using System;
using System.Runtime.InteropServices;  // DllImport()

namespace RFID
{
     public enum PlayerFlags : int
     {
          SND_ASYNC = 0x0001,    // play asynchronously
          SND_SYNC = 0x0000    // play synchronously
     }

     public class Player
     {
          //Plays the sound file fname
          [DllImport("coredll.dll")]
          public static extern bool  PlaySound(string fname, int Mod, int ↙
          flag);
     }
}
```

# TagList class

```
using System;
using System.Windows.Forms;

namespace RFID
{
     public class TagList : System.Collections.ArrayList
     {

          //Constructor
          public TagList()
          {
          }

          //Merges two tagLists to a new list (this)
          public void CopyTo(TagList list, TagList list2)
          {
               foreach(Tag t in list)
               {
                    this.Add(t);
               }
               foreach(Tag t2 in list2)
               {
                    this.Add(t2);
               }
          }

          //Finds the tag with the ID i
          public Tag Find(string i)
          {
               Tag temp = null;
               foreach(Tag t in this)
               {
                    if(t.tagNumber == i)
                    {
                         return t;
                    }
                    else
                         temp = null;
               }
               return temp;
          }
     }
}
```

# PortManager class

```
using System;
using System.Windows.Forms;
using System.Data;
using System.IO;
using System.Threading;
using System.Runtime.InteropServices;
using System.Collections;
using System.Text;

namespace RFID
{
     [Flags]
     public enum CommEventFlags : int
     {
          NONE = 0x0000,     //No flags
          RXCHAR = 0x0001,   //Any Character received
          RXFLAG = 0x0002,   //Received specified flag character
          TXEMPTY = 0x0004,  //Tx buffer Empty
          CTS = 0x0008,      //CTS changed
          DSR = 0x0010,      //DSR changed
          RLSD = 0x0020,     //RLSD changed
          BREAK = 0x0040,    //BREAK received
          ERR = 0x0080,      //Line status error
          RING = 0x0100,     //ring detected
          PERR = 0x0200,     //printer error
          RX80FULL = 0x0400, //rx buffer is at 80%
          EVENT1 = 0x0800,   //provider event
          EVENT2 = 0x1000,   //provider event
          POWER = 0x2000,    //wince power notification
          ALLCE = 0x3FFF,    //mask of all flags for CE
          ALLPC = BREAK | CTS | DSR | ERR | RING | RLSD | RXCHAR | RXFLAG ↙
          | TXEMPTY //Mask for all flags under desktop Windows
     }

     internal enum EventFlags
     {
          EVENT_PULSE = 1,
          EVENT_RESET = 2,
          EVENT_SET = 3
     }

     [Flags]
     public enum CommErrorFlags : int
```

```
{
     RXOVER = 0x0001,    // Receive overrun
     OVERRUN = 0x0002,   // Overrun
     RXPARITY = 0x0004, // Parity error
     FRAME = 0x0008,     // Frame error
     BREAK = 0x0010,     // BREAK received
     TXFULL = 0x0100,    // Transmit buffer full
     IOE = 0x0400,       // IO Error
     MODE = 0x8000       // Requested mode not supported
}

[Flags]
public enum CommModemStatusFlags : int
{
     MS_CTS_ON = 0x0010,     // The CTS (Clear To Send) signal is on.
     MS_DSR_ON = 0x0020,     // The DSR (Data Set Ready) signal is on.
     MS_RING_ON = 0x0040,    // The ring indicator signal is on.
     MS_RLSD_ON = 0x0080     // The RLSD (Receive Line Signal ↙
     Detect) signal is on.
}

internal enum CommEscapes : uint
{
     SETXOFF  = 1, // Causes transmission to act as if an XOFF ↙
     character has been received.
     SETXON = 2,    // Causes transmission to act as if an XON ↙
     character has been received.
     SETRTS = 3,    // Sends the RTS (Request To Send) signal.
     CLRRTS = 4,    // Clears the RTS (Request To Send) signal
     SETDTR = 5,    // Sends the DTR (Data Terminal Ready) signal.
     CLRDTR = 6,    // Clears the DTR (Data Terminal Ready) signal.
     SETBREAK = 8, // Suspends character transmission and places ↙
     the transmission line in a break state until the ClearCommBreak ↙
     function is called (or EscapeCommFunction is called with the ↙
     CLRBREAK extended function code). The SETBREAK extended ↙
     function code is identical to the SetCommBreak function. This ↙
     extended function does not flush data that has not been ↙
     transmitted.
     CLRBREAK = 9, // Restores character transmission and places ↙
     the transmission line in a nonbreak state. The CLRBREAK ↙
     extended function code is identical to the ClearCommBreak ↙
     function
     SETIR = 10,    //Set the port to IR mode.
     CLRIR = 11     // Set the port to non-IR mode.
}
```

```
internal enum APIErrors : int
{
     ERROR_FILE_NOT_FOUND = 2,    // Port not found
     ERROR_INVALID_NAME = 123,    // Invalid port name
     ERROR_ACCESS_DENIED = 5,     // Access denied
     ERROR_INVALID_HANDLE = 6,    // invalid handle
     ERROR_IO_PENDING = 997       // IO pending
}

internal enum APIConstants : uint
{
     WAIT_OBJECT_0 = 0x00000000,
     WAIT_ABANDONED = 0x00000080,
     WAIT_ABANDONED_0 = 0x00000080,
     WAIT_FAILED = 0xffffffff,
     INFINITE = 0xffffffff
}

[StructLayout(LayoutKind.Sequential)]
internal class CommStat
{
     //private BitVector32 bitfield; // UKI added for CLR bitfield ↙
     support
     public UInt32 cbInQue   = 0;
     public UInt32 cbOutQue  = 0;

     // Helper constants for manipulating the bit fields.
     [Flags]
          private enum commFlags
     {
          fCtsHoldMask = 0x01,
          fDsrHoldMask = 0x02,
          fRlsdHoldMask = 0x04,
          fXoffHoldMask = 0x08,
          fXoffSentMask = 0x10,
          fEofMask = 0x20,
          fTximMask = 0x40
     };
}

public class PortManager
{

     //För att kunna lyssna om det finns något på porten
     private Thread eventThread;
     private ManualResetEvent threadStarted = new ↙
```

```
ManualResetEvent(false);

private IntPtr closeEvent;
private string closeEventName;

private int rxBufferSize = 1024;
private Queue rxFIFO;
private int rthreshold = 1;

private int txBufferSize = 1024;
private byte[] txBuffer;

private Mutex rxBufferBusy = new Mutex();
private int inputLength;

private IntPtr txOverlapped = IntPtr.Zero;
private IntPtr rxOverlapped = IntPtr.Zero;

//Serial Constants
public int BaudRate = 57600;     //Serial baudrate setting
public byte ByteSize =  8;       //Serial byte trans size
public byte Parity = 0;          //Serial: 0-4=no,odd,even, ↙
mark,space
public byte StopBits =  1;       //Serial: 0,1,2 = 1, 1.5, 2
public int ReadTimeout;          //Serial
public Thread ecgThread;         //ECG Process thread

//COMM Port WIN32 File Handle
private int hComm = INVALID_HANDLE_VALUE;  //Default serial ↙
handle to "invalid" (WINBASE)
public bool Opened = false;      //Default port monitor to ↙
"closed"

//WIN32 API Constants
//Set win32 cons (including RTS/DTR commands) from WINBASE.h
private const uint GENERIC_READ = 0x80000000;  //Needed for ↙
"Createfile"
private const uint GENERIC_WRITE = 0x40000000;
private const int OPEN_EXISTING = 3;
private const int INVALID_HANDLE_VALUE = -1;
private const int SETRTS = 3;    //Needed to power PersonalECG
private const int SETDTR = 5;
private const int CLRRTS = 4;
private const int CLRDTR = 6;

public struct COMMPROP{public uint dwProvSubType;}
```

```
public struct DCB
{
     //Serial Definitions Block
     public int DCBlength;   // sizeof(DCB)
     public int BaudRate;    // Current baud rate
     public uint flags;
     public ushort wReserved;     // Not currently used
     public ushort XonLim;        // Transmit XON threshold
     public ushort XoffLim;       // Transmit XOFF threshold
     public byte ByteSize;        // Number of bits/byte, 4-8 (8)
     public byte Parity;          // 0-4=no,odd,even,mark,space ↙
     (0)
     public byte StopBits;        // 0,1,2 = 1, 1.5, 2 (0)
     public char XonChar;         // Tx and Rx XON character
     public char XoffChar;        // Tx and Rx XOFF character
     public char ErrorChar;       // Error replacement character
     public char EofChar;         // End of input character
     public char EvtChar;         // Received event character
     public ushort wReserved1;    // Reserved; do not use
     public int fRtsControl;      // Control of RTS line to ↙
     power PersonalECG
     public int fDtrControl;      // Control of DTR line to ↙
     power PersonalECG

}
private struct COMMTIMEOUTS
{
     //Serial Port Timeout Block
     public int ReadIntervalTimeout;
     public int ReadTotalTimeoutMultiplier;
     public int ReadTotalTimeoutConstant;
     public int WriteTotalTimeoutMultiplier;
     public int WriteTotalTimeoutConstant;
}

private struct OVERLAPPED
{
     //Serial Overlapped Block
     public int  Internal;
     public int  InternalHigh;
     public int  Offset;
     public int  OffsetHigh;
     public int    hEvent;
}
```

```
//IMPORT WIN32 API METHODS

[DllImport("coredll.dll")]
private static extern uint GetLastError();
[DllImport("coredll.dll")]
private static extern int CreateFile(
     string lpFileName,          // file name ↙
     uint dwDesiredAccess,       // access mode ↙
     uint dwShareMode,           // share mode ↙
     int lpSecurityAttributes,   // SD ↙
     uint dwCreationDisposition, // how to create ↙
     uint dwFlagsAndAttributes,  // file attributes ↙
     int hTemplateFile           // handle to template file ↙
     );
[DllImport("coredll.dll")]
private static extern bool GetCommState(
     int hFile,         // handle to communications device ↙
     ref DCB lpDCB      // device-control block ↙
     );
[DllImport("coredll.dll")]
private static extern bool BuildCommDCB(
     string lpDef,      // device-control string ↙
     ref DCB lpDCB      // device-control bloc ↙
     );
[DllImport("coredll.dll")]
private static extern bool SetCommState(
     int hFile,         // handle to communications device ↙
     ref DCB lpDCB      // device-control block ↙
     );
[DllImport("coredll.dll")]
private static extern bool GetCommTimeouts(
     int hFile,                      // handle to comm device ↙
     ref COMMTIMEOUTS lpCommTimeouts  // time-out values ↙
     );
[DllImport("coredll.dll")]
private static extern bool SetCommTimeouts(
     int hFile,                      // handle to comm device ↙
     ref COMMTIMEOUTS lpCommTimeouts  // time-out values ↙
     );
[DllImport("coredll.dll")]
public static extern bool ReadFile(
     int handle, [In,Out] byte[] buffer, uint bytesToRead, ↙
     out uint bytesRead, IntPtr overlapped ) ↙
     ;
[DllImport("coredll.dll")]
private static extern bool WriteFile(
```

```
        int hFile,                          // handle to file ↙
        byte[] lpBuffer,                    // data buffer ↙
        int nNumberOfBytesToWrite,          // number of bytes to ↙
        write
        ref int lpNumberOfBytesWritten,     // number of bytes ↙
        written
        ref OVERLAPPED lpOverlapped         // overlapped buffer ↙
        );
    [DllImport("coredll.dll")]
    private static extern bool CloseHandle(
        int hObject                         // handle to object ↙
        );
    [DllImport("coredll.dll")]
    private static extern bool SetCommBreak(
        int hFile                           // handle to object ↙
        );
    [DllImport("coredll.dll")]
    private static extern bool ClearCommBreak(
        int hFile                           // handle to object ↙
        );
    [DllImport("coredll.dll")]
    private static extern bool EscapeCommFunction(
        int hFile,                          // handle to object ↙
        byte dwFunc                         // pin command ↙
        );

    [DllImport("coredll.dll", EntryPoint="ReadFile", SetLastError = ↙
    true)]
    private static extern int CEReadFile(IntPtr hFile, byte[]↙
    lpBuffer, Int32 nNumberOfBytesToRead, ref Int32 ↙
    lpNumberOfBytesRead, IntPtr lpOverlapped);

    [DllImport("coredll.dll", EntryPoint="SetCommMask", ↙
    SetLastError = true)]
    private static extern int CESetCommMask(IntPtr handle, ↙
    CommEventFlags dwEvtMask); //För att kunna lyssna

    [DllImport("coredll.dll", EntryPoint="WaitForSingleObject", ↙
    SetLastError = true)]
    private static extern int CEWaitForSingleObject(IntPtr hHandle, ↙
    uint dwMilliseconds);

    [DllImport("coredll.dll", EntryPoint="EventModify", ↙
    SetLastError = true)]
    private static extern int CEEventModify(IntPtr hEvent, uint ↙
    function);
```

```
[DllImport("coredll.dll", EntryPoint="CreateEvent", ↙
SetLastError = true)]
private static extern IntPtr CECreateEvent(IntPtr ↙
lpEventAttributes, int bManualReset, int bInitialState, string ↙
lpName);

[DllImport("coredll.dll", EntryPoint="GetCommModemStatus", ↙
SetLastError = true)]
private static extern int CEGetCommModemStatus(IntPtr hFile, ↙
ref uint lpModemStat);

[DllImport("coredll.dll", EntryPoint="ClearCommError", ↙
SetLastError = true)]
private static extern int CEClearCommError(IntPtr hFile, ref ↙
CommErrorFlags lpErrors, CommStat lpStat);

[DllImport("coredll.dll", EntryPoint="WaitCommEvent", ↙
SetLastError = true)]
private static extern int CEWaitCommEvent(IntPtr hFile, ref ↙
CommEventFlags lpEvtMask, IntPtr lpOverlapped);

public  bool ReadFile2(IntPtr hPort, byte[] buffer, int ↙
cbToRead, ref Int32 cbRead, IntPtr lpOverlapped)
{
     return Convert.ToBoolean(CEReadFile(hPort, buffer, ↙
     cbToRead, ref cbRead, IntPtr.Zero));
}

public delegate void CommEvent();     // Raised on all enabled ↙
communication events
public delegate void CommChangeEvent(bool NewState);
// Raised when the communication state changes
public delegate void CommErrorEvent(string Description);
// Raised during any communication error
public event CommErrorEvent OnError;  //  A communication error ↙
has occurred
public event CommEvent DataReceived;  // Serial data has been ↙
received
public event CommEvent TxDone;        // Transmit complete
public event CommEvent FlagCharReceived;   // Set flag ↙
character was in the receive stream
public event CommEvent PowerEvent;    // Power change event has ↙
occurred
public event CommEvent HighWater;     // Serial buffer's high- ↙
water level has been exceeded
```

```
public event CommChangeEvent DSRChange;    // DSR state has ↙
changed
public event CommChangeEvent RingChange;   // Ring signal has ↙
been detected
public event CommChangeEvent CTSChange;    // CTS state has ↙
changed
public event CommChangeEvent RLSDChange;   // RLSD state has ↙
changed

public PortManager()
{
     closeEventName = "CloseEvent";
     closeEvent = CECreateEvent(IntPtr.Zero, Convert. ↙
     ToInt32(true), Convert.ToInt32(false), closeEventName);
     rxFIFO = new Queue(rxBufferSize);
     txBuffer = new byte[txBufferSize];
}

public bool PortOpen(string port)
{
     //Configure/Prepare to Open
     PortClose();  //Close port if already open
     DCB dcbCommPort = new DCB();     //Call DCB struct
     COMMTIMEOUTS ctoCommPort = new COMMTIMEOUTS(); //Call ↙
     COMMTIMEOUTS struct
     COMMPROP commprop = new COMMPROP();   //Call COMMPROP ↙
     struct

     //Open the COM port
     hComm = CreateFile (
          port,     //Pointer to the name of the port
          GENERIC_READ | GENERIC_WRITE,//Access (read-write) mode
          0,        //Share mode
          0,        //Pointer to the security attribute
          OPEN_EXISTING,     //How to open the serial port
          0,        //Port attributes
          0);       //Handle to port with attribute to copy
     // If Port Can't be Opened, Return
     if(hComm == INVALID_HANDLE_VALUE) return false;
     //Return, update user
     if(hComm == INVALID_HANDLE_VALUE)
     {
          throw(new ApplicationException("Comm Port Can Not Be ↙
          Opened"));
     }
```

```
// Set the COMM Port Type
commprop.dwProvSubType = 0x00000001;

// Set the COMM Timeouts
GetCommTimeouts(hComm,ref ctoCommPort);
ctoCommPort.ReadTotalTimeoutConstant = ReadTimeout;
ctoCommPort.ReadTotalTimeoutMultiplier = 0;
ctoCommPort.WriteTotalTimeoutMultiplier = 0;
ctoCommPort.WriteTotalTimeoutConstant = 0;
SetCommTimeouts(hComm,ref ctoCommPort);

// Set BAUD RATE, PARITY, WORD SIZE, AND STOP BITS. ↙
(From Above Values)
GetCommState(hComm, ref dcbCommPort);
dcbCommPort.BaudRate=BaudRate;
dcbCommPort.flags=0;
dcbCommPort.flags|=1;
if (Parity>0)
{
     dcbCommPort.flags|=2;
}
dcbCommPort.Parity=Parity;
dcbCommPort.ByteSize=ByteSize;
dcbCommPort.StopBits=StopBits;

//Setup VCC Pins for PersonalECG Device - ENABLE COMMANDS
dcbCommPort.fDtrControl = 0x01;
//DTR_CONTROL_ENABLE    Enable control over DTR/RTS
dcbCommPort.fRtsControl = 0x01;
//RTS_CONTROL_ENABLE

if (!SetCommState(hComm, ref dcbCommPort))     //Throw ↙
failure exception if needed
{
     //uint ErrorNum=GetLastError();
     throw(new ApplicationException("Comm Port Can Not Be ↙
     Opened"));
}

//Set VCC Pins for PersonalECG Device to 'HIGH'
EscapeCommFunction(hComm, SETRTS);    //Set RTS high
EscapeCommFunction(hComm, SETDTR);    //Set DTR high
Opened = true;     //Port open

// Start the receive thread
eventThread = new Thread(new ThreadStart(CommEventThread));
```

```
            eventThread.Start();
            threadStarted.WaitOne();

            return true;  //Return bool

        }

        public void PortClose()
        //Close port
        {
            if (hComm!=INVALID_HANDLE_VALUE)
            {
                CloseHandle(hComm);
                //Terminate handle
                hComm = INVALID_HANDLE_VALUE;
                //Reset handle
            }
        }

        public byte PortRead()
        //Read one byte from serial
        {
            uint byteRead;
            //Declare buffer
            byte[] BufByte = new byte[1];
            bool result = ReadFile( hComm, BufByte, 1, out byteRead, ↙
            IntPtr.Zero );//Read byte from port
            if (result == true)
                //If successful, return byte
                return byteRead == 0 ? (byte)0 : BufByte[0];
            else
            {
                //If it failed, update UI
                return (byte)0;
                //Return 0's
            }
        }

        public void PortWrite(byte[] WriteBytes)
        {
            if (hComm!=INVALID_HANDLE_VALUE)
            //If port opened correctly, write
            {
                OVERLAPPED ovlCommPort = new OVERLAPPED();
                int BytesWritten = 0;
                WriteFile(hComm,WriteBytes,WriteBytes.Length,ref ↙
```

```
                BytesWritten,ref ovlCommPort);   //Write byte
        }
        else
        {
                throw(new ApplicationException("Comm Port Not Open"));
                //Else, return/update UI
        }
}

private void CommEventThread()
{
        byte[] readbuffer = new Byte[rxBufferSize];
        CommEventFlags eventFlags = new CommEventFlags();
        int bytesRead = 0;
        AutoResetEvent rxEvent = new AutoResetEvent(false);
        CESetCommMask((IntPtr)hComm, CommEventFlags.ALLCE);

        try
        {
                threadStarted.Set();

                #region >>>> thread loop <<<<
                while(hComm != (int)INVALID_HANDLE_VALUE)
                {
                        // wait for a Comm event
                        if(!Convert.ToBoolean( ↙
                        CEWaitCommEvent((IntPtr)hComm, ↙
                        ref eventFlags, IntPtr.Zero)))
                        {
                                int e = Marshal.GetLastWin32Error();

                                if(e == (int)APIErrors.ERROR_IO_PENDING)
                                {
                                        // IO pending so just wait and try again
                                        rxEvent.WaitOne();
                                        Thread.Sleep(0);
                                        continue;
                                }

                                if(e == (int)APIErrors.ERROR_INVALID_HANDLE)
                                {
                                        // Calling Port.Close() causes hPort to ↙
                                        become invalid
                                        // Since Thread.Abort() is unsupported ↙
                                        in the CF, we must
                                        // accept that calling Close will throw ↙
```

```
                an error here.

                // Close signals the closeEvent, so ↙
                wait on it
                // We wait 1 second, though Close ↙
                should happen much sooner
                int eventResult = ↙
                CEWaitForSingleObject(closeEvent, 1000);

                if(eventResult == ↙
                (int)APIConstants.WAIT_OBJECT_0)
                {
                     // the event was set so close was ↙
                     called
                     hComm = INVALID_HANDLE_VALUE;

                     // reset our ResetEvent for the ↙
                     next call to Open
                     threadStarted.Reset();

                     return;
                }
           }

           // WaitCommEvent failed
           // 995 means an exit was requested (thread ↙
           killed)
           if(e == 995)
           {
                return;
           }
           else
           {
                string error = String.Format("Wait ↙
                Failed: {0}", e);
                //throw new CommPortException(error);
           }
      }

      // Re-specify the set of events to be monitored ↙
      for the port.
      CESetCommMask((IntPtr)hComm, CommEventFlags.ALLCE);

      // check the event for errors
      #region >>>> error checking <<<<
      if(((uint)eventFlags & (uint)CommEventFlags.ERR) ↙
```

```
!= 0)
{
     CommErrorFlags errorFlags = new ↙
     CommErrorFlags();
     CommStat commStat = new CommStat();

     if(((uint)errorFlags & ↙
     (uint)CommErrorFlags.BREAK) != 0)
     {
          // BREAK can set an error, so make sure ↙
          the BREAK bit is set an continue
          eventFlags |= CommEventFlags.BREAK;
     }
     else
     {
          // we have an error. Build a meaningful ↙
          string and throw an exception
          StringBuilder s = new StringBuilder( ↙
          "UART Error: ", 80);
          if ((errorFlags & CommErrorFlags.FRAME) ↙
          != 0)
          { s = s.Append("Framing,"); }
          if ((errorFlags & CommErrorFlags.IOE) ↙
          != 0)
          { s = s.Append("IO,"); }
          if ((errorFlags & CommErrorFlags.↙
          OVERRUN)!= 0)
          { s = s.Append("Overrun,"); }
          if ((errorFlags & CommErrorFlags.↙
          RXOVER) != 0)
          { s = s.Append("Receive Overflow,"); }
          if ((errorFlags & CommErrorFlags. ↙
          RXPARITY) != 0)
          { s = s.Append("Parity,"); }
          if ((errorFlags & CommErrorFlags. ↙
          TXFULL) != 0)
          { s = s.Append("Transmit Overflow,"); }

          // no known bits are set
          if(s.Length == 12)
          { s = s.Append("Unknown"); }

          // raise an error event
          if(OnError != null)
               OnError(s.ToString());
```

```
                continue;
            }
        } // if(((uint)eventFlags & ↙
        (uint)CommEventFlags.ERR) != 0)
        #endregion

        #region >>>> line status checking <<<<
        // check for status changes
        uint status = 0;
        CEGetCommModemStatus((IntPtr)hComm, ref status);

        // check the CTS
        if(((uint)eventFlags & (uint)CommEventFlags.CTS) ↙
        != 0)
        {
            if(CTSChange != null)
                CTSChange((status & ↙
                (uint)CommModemStatusFlags.MS_CTS_ON) ↙
                != 0);
        }

        // check the DSR
        if(((uint)eventFlags & (uint)CommEventFlags.DSR) ↙
        != 0)
        {
            if(DSRChange != null)
                DSRChange((status & ↙
                (uint)CommModemStatusFlags.MS_DSR_ON) ↙
                != 0);
        }

        // check for a RING
        if(((uint)eventFlags & (uint)CommEventFlags.↙
        RING) != 0)
        {
            if(RingChange != null)
                RingChange((status & ↙
                (uint)CommModemStatusFlags.MS_RING_ON) ↙
                != 0);
        }

        // check for a RLSD
        if(((uint)eventFlags & (uint)CommEventFlags.↙
        RLSD) != 0)
        {
            if(RLSDChange != null)
```

```
                RLSDChange((status & ↙
                (uint)CommModemStatusFlags.MS_RLSD_ON) ↙
                != 0);
        }

        // check for TXEMPTY
        if(((uint)eventFlags & ↙
        (uint)CommEventFlags.TXEMPTY) != 0)
             if(TxDone != null) { TxDone(); }

        // check for RXFLAG
        if(((uint)eventFlags & (uint)CommEventFlags. ↙
        RXFLAG) != 0)
             if(FlagCharReceived != null) { ↙
             FlagCharReceived(); }

        // check for POWER
        if(((uint)eventFlags & (uint)CommEventFlags. ↙
        POWER) != 0)
             if(PowerEvent != null) { PowerEvent(); }

        // check for high-water state
        if((eventFlags & CommEventFlags.RX80FULL) != 0)
             if(HighWater != null) { HighWater(); }
        #endregion

        #region >>>> Receive data subsection <<<<
        // check for RXCHAR
        if((eventFlags & CommEventFlags.RXCHAR) != 0)
        {
             do
             {
                  // make sure the port handle is valid
                  if(hComm == INVALID_HANDLE_VALUE)
                  {
                       bytesRead = 0;
                       break;
                  }

                  // data came in, put it in the buffer ↙
                  and set the event
                  if (!ReadFile2((IntPtr)hComm, ↙
                  readbuffer, rxBufferSize, ref ↙
                  bytesRead, rxOverlapped))
                  {
                       string errString = String.Format( ↙
```

```
                    "ReadFile Failed: {0}", Marshal. ↙
                    GetLastWin32Error());
                    if(OnError != null)
                        OnError(errString);

                    return;
                }
                if (bytesRead >= 1)
                {
                    // take the mutex
                    rxBufferBusy.WaitOne();

                    // put the data into the fifo
                    // this *may* be a perf problem ↙
                    and needs testing
                    for(int b = 0 ; b < bytesRead ; b++)
                        rxFIFO.Enqueue(readbuffer[b]);

                    // get the FIFO length
                    int fifoLength = rxFIFO.Count;

                    // release the mutex
                    rxBufferBusy.ReleaseMutex();

                    // fire the DataReceived event ↙
                    every RThreshold bytes
                    if((DataReceived != null) && ↙
                    (rthreshold != 0) && (fifoLength ↙
                    >= rthreshold))
                    {
                        DataReceived();
                    }
                }
            } while (bytesRead > 0);
        } // if((eventFlags & CommEventFlags.RXCHAR) != 0)
        #endregion
    } // while(true)
    #endregion
} // try
catch(Exception e)
{
    if(OnError != null)
        OnError(e.Message);

    return;
}
```

```
        }

        public byte[] Input
        {
            get
            {
                int dequeueLength = 0;

                // lock the rx FIFO while reading
                rxBufferBusy.WaitOne();

                // how much data are we *actually* going to return ↙
                from the call?
                if(inputLength == 0)
                    dequeueLength = rxFIFO.Count;  // pull the ↙
                    entire buffer
                else
                    dequeueLength = (inputLength < rxFIFO.Count) ? ↙
                    inputLength : rxFIFO.Count;

                byte[] data = new byte[dequeueLength];

                // dequeue the data
                for(int p = 0 ; p < dequeueLength ; p++)
                    data[p] = (byte)rxFIFO.Dequeue();

                // release the mutex so the Rx thread can continue
                rxBufferBusy.ReleaseMutex();

                return data;
            }
        }
    }
}
```

# Recorder class

```
using System;
using System.Windows.Forms;
using System.Runtime.InteropServices;

namespace RFID
{
     public enum RecorderFlags : uint
     {
          VRS_NO_NOTIFY = 0x0002,           // No parent notifcation
          VRS_NO_MOVE = 0x0040              // Grip is removed and the ↙
          control cannot be moved around by the user
     }

     public class Recorder
     {
          //API Declares
          // Define calls to Recorder_Create wrapper
          [DllImport("voicerecorder.dll", SetLastError = true)]
          private static extern IntPtr InitRecorder(RecorderFlags ↙
          dwStyle, IntPtr hwndParent, int xPos, int yPos, String ↙
          lpszRecordFileName);
          [DllImport("voicerecorder.dll", SetLastError = true)]
          private static extern void RecordVoice(IntPtr hwndRecorder);
          [DllImport("voicerecorder.dll", SetLastError = true)]
          private static extern void CloseRecorder(IntPtr hwndRecorder);
          //Creates a recorder with the predefined settings and returns ↙
          an IntPtr that is used in the other two methods
          public IntPtr Create(RecorderFlags dwStyle, IntPtr hwndParent, ↙
          int xPos, int yPos, String Filename)
          {
               IntPtr handle = new IntPtr();
               try
               {
                    handle = InitRecorder(dwStyle,hwndParent,xPos,yPos, ↙
                    Filename);
               }
               catch (Exception err)
               {
                    MessageBox.Show(err.Message);
                    return IntPtr.Zero;
               }
               return handle;
          }
```

```
        //Begins the recording and creates the file Filename in the ↙
        'hwndRecord' created in Create
        public void Record(IntPtr hwndRecord)
        {
            try
            {
                RecordVoice(hwndRecord);
            }
            catch (Exception err)
            {
                MessageBox.Show(err.Message);
                return;
            }
        }

        //Ends the recording and saves it to the file Filename in the ↙
        'hwndRecord' created in Create
        public void Close(IntPtr hwndRecord)
        {
            try
            {
                CloseRecorder(hwndRecord);
            }
            catch (Exception err)
            {
                MessageBox.Show(err.Message);
                return;
            }
        }
    }

    //Keeps track on what window is used and gives the info to the ↙
    Recorder, which will appear in the active window
    public class WindowInfo
    {
        [DllImport("coredll.dll")]
        private static extern IntPtr FindWindow(string ClassName, ↙
        string WindowName);

        public IntPtr hWnd(string ClassName, string WindowName)
        {
            return FindWindow(ClassName, WindowName);
        }
    }
}
```

# Tag class

```
using System;

namespace RFID
{
     public class Tag
     {
          private string theTagNumber;
          private string theTagSound;
          private string theAttribute;
          private string theTagSound2;

          //Constructor to create a plain tag without attribute or sounds
          public Tag()
          {
          }

          //Constructor to create a tag with the ID i, sound1 str and ↙
          attribute attr
          public Tag(string i, string str, string attr)
          {
               theTagNumber = i;
               theTagSound = str;
               theAttribute = attr;
               theTagSound2 = "";
          }

          //Constructor to create a tag with the ID i, sound1 str, sound2 ↙
          str2 and attribute attr
          public Tag(string i, string str, string str2, string attr)
          {
               theTagNumber = i;
               theTagSound = str;
               theTagSound2 = str2;
               theAttribute = attr;
          }

          public string tagNumber
          {
               get { return theTagNumber; }
          }

          public string tagSound
          {
```

```
            get { return theTagSound; }
        }

        public string attribute
        {
            get { return theAttribute; }
        }

        public string tagSound2
        {
            get { return theTagSound2; }
        }

        public void changeTagSound2(string s)
        {
            theTagSound2 = s;
        }
        public void changeTagSound(string s)
        {
            theTagSound = s;
        }
    }
}
```