```
1  using System;
2  using System.Diagnostics;
3  using System.Collections.Generic;
4  using System.ComponentModel;
5  using System.Data;
6  using System.Drawing;
7  using System.Text;
8  using System.Windows.Forms;
9  using System.Threading;
10 using System.Runtime.InteropServices;
11 using Microsoft.StylusInput;
12 using Microsoft.StylusInput.PluginData;
13 using Microsoft.Ink;
14
15 /// Kenrick Mock
16 /// Mouse attention-highlighter
17 /// TO DO: Comment and re-factor this messy code
18
19 namespace PenAttention
20 {
21     // The class must implement the IStylusAsyncPlugin interface
22     // to get pen notification calls
23     public partial class frmAttention : Form, IStylusAsyncPlugin
24     {
25         private RealTimeStylus myRealTimeStylus;    // Used to detect pen
26         public static bool pen_detected = false;    // Whether or not pen is nearby
27         public enum Attention_Mode { Highlight, Pencil, Pointer, None };
28         private static Attention_Mode attention_type = Attention_Mode.Highlight;
29         long lMsg;
30         IntPtr myHwnd;
31         PerPixelAlphaForm highlightForm = new PerPixelAlphaForm();
32         PerPixelAlphaForm extendedHighlightForm = new PerPixelAlphaForm();
33         private static Bitmap curBmp = null;
34         private bool circleHighlight = true;    // Default is we are using the circle highlight
35         private bool made_changes = false;      // Whether or not we changed a setting
36
37         // Right click message
38         private static int RCLICKCODE = 32767;
39
40         // Original highlight settings
41         private int origRadius, origOpacity;
42         private Color origColor;
43         private bool origHighlightMouse;
44         private bool origCircleHighlight;
45         private int origHeight, origWidth;
46         private bool origToggleColor;
47
48         // Extended monitor settings
```

```
49          public bool extendedEnabled = false;
50          private double scaleEX = 1;             // Scaling factor for X dimension
51          private double scaleEY = 1;             // Scaling factor for Y dimension
52          private int startEX = 1024;             // Initial X/Y coordinate of extended monitor
53          private int startEY = 0;
54          public int primaryX = 0;               // Coordinates, size of local window
55          public int primaryY = 0;
56          public int primaryHeight = 768;
57          public int primaryWidth = 1024;
58
59          [DllImport("user32.dll", SetLastError = true)]
60          static extern bool ChangeWindowMessageFilter(uint message, uint dwFlag);
61
62          [DllImport("user32.dll")]
63          static extern int SetWindowLong(IntPtr hWnd, int nIndex, int dwNewLong);
64
65          [DllImport("user32.dll")]
66          static extern long GetWindowLong(IntPtr hWnd, int nIndex);
67
68          [DllImport("MouseHook.dll")]
69          static extern bool setMyHook(IntPtr hWnd);
70
71          [DllImport("MouseHook.dll")]
72          static extern bool clearMyHook(IntPtr hWnd);
73
74          [DllImport("MouseHook.dll")]
75          static extern void setFlag(int flag);
76
77          [DllImport("user32.dll", EntryPoint = "FindWindow", SetLastError = true)]
78          static extern System.IntPtr FindWindowByCaption(int ZeroOnly, string lpWindowName);
79
80          [DllImport("user32.dll", SetLastError = true, CharSet = CharSet.Auto)]
81          static extern uint RegisterWindowMessage(string lpString);
82
83          public frmAttention()
84          {
85              InitializeComponent();
86              myRealTimeStylus = new RealTimeStylus(this.pboxPencil, false); // Register stylus, not mouse
87              myRealTimeStylus.AsyncPluginCollection.Add(this);   // Get callbacks on pen activity
88              myRealTimeStylus.Enabled = true;
89              pen_detected = false;
90              this.ControlBox = false;
91          }
92
93          private void frmSettings_Load(object sender, EventArgs e)
94          {
95              //SetWindowLong(this.Handle, -20, (int)GetWindowLong(this.Handle, -20) | 0x00000020);
96
```

```
97              //this.Left = System.Windows.Forms.Control.MousePosition.X -15;
98              //this.Top = System.Windows.Forms.Control.MousePosition.Y - 15;
99
100             myHwnd = FindWindowByCaption(0, this.Text);
101             setMyHook(myHwnd);
102             //Register the message
103             lMsg = RegisterWindowMessage("UWM_MOUSEMOVE_MSG-PENATTEN-44E531B1_14D3_11d5_A025_006067718D04");
104
105             int osversion = System.Environment.OSVersion.Version.Major;
106             if (osversion >= 6)
107             {
108                 // Vista, add message to filter list so we can get it from IE
109                 ChangeWindowMessageFilter(Convert.ToUInt32(lMsg), 1);
110             }
111
112             // Defaults for highlighting
113             btnColor.BackColor = PenAttention.Properties.Settings.Default.highlightColor;
114             cbHighlightMouse.Checked = PenAttention.Properties.Settings.Default.highlightMouse;
115             tbRadius.Value = PenAttention.Properties.Settings.Default.highlightRadius;
116             tbOpacity.Value = PenAttention.Properties.Settings.Default.highlightOpacity;
117             circleHighlight = PenAttention.Properties.Settings.Default.circleHighlight;
118             tbHeight.Value = PenAttention.Properties.Settings.Default.highlightHeight;
119             tbWidth.Value = PenAttention.Properties.Settings.Default.highlightWidth;
120             cbToggleColor.Checked = PenAttention.Properties.Settings.Default.toggleColor;
121
122             // Extended display settings
123             extendedEnabled = PenAttention.Properties.Settings.Default.extendedDisplay;
124             enableExtendedMonitorHighlightToolStripMenuItem.Checked = extendedEnabled;
125             primaryX = PenAttention.Properties.Settings.Default.primaryX;
126             primaryY = PenAttention.Properties.Settings.Default.primaryY;
127             primaryHeight = PenAttention.Properties.Settings.Default.primaryHeight;
128             primaryWidth = PenAttention.Properties.Settings.Default.primaryWidth;
129             // Check if we have an extended display
130             if (Screen.AllScreens.Length <= 1)
131             {
132                 extendedEnabled = false;
133                 enableExtendedMonitorHighlightToolStripMenuItem.Checked = false;
134             }
135             else if (extendedEnabled)
136             {
137                 calcScreenMappings();
138             }
139
140             if (circleHighlight)
141             {
142                 rbCircle.Checked = true;
143             }
144             else
```

```
145             {
146                 rbRectangle.Checked = true;
147             }
148 
149             // Icon or highlight
150             int attentionMode = PenAttention.Properties.Settings.Default.selectedIcon;
151             if (attentionMode == 0)
152             {
153                 attention_type = Attention_Mode.Highlight;
154                 this.showHighlightToolStripMenuItem.Checked = true;
155                 setHighlightForm();
156                 updateHighlightForm();
157             }
158             else if (attentionMode == 1)
159             {
160                 attention_type = Attention_Mode.Pencil;
161                 this.showPencilToolStripMenuItem.Checked = true;
162                 setIconForm(pboxPencil);
163                 updateHighlightForm();
164             }
165             else if (attentionMode == 2)
166             {
167                 attention_type = Attention_Mode.Pointer;
168                 this.showPointerToolStripMenuItem.Checked = true;
169                 setIconForm(pboxPointer);
170                 updateHighlightForm();
171                 //highlightForm.Hide();
172                 //extendedHighlightForm.Hide();
173             }
174             else if (attentionMode == -1)
175             {
176                 attention_type = Attention_Mode.None;
177                 highlightForm.Hide();
178                 extendedHighlightForm.Hide();
179             }
180             //Hide();
181         }
182 
183         protected override void WndProc(ref Message m)
184         {
185             //calling the base first is important, otherwise the values you set later will be lost
186             base.WndProc(ref m);
187 
188             if (m.Msg == lMsg)
189             {
190                 if (m.WParam.ToInt32() == RCLICKCODE)       // Right click
191                 {
192                     toggleHighlightColors();
```

```
193                 }
194                 else                                        // Mouse Move
195                 {
196                     MouseMoved(m.WParam.ToInt32(), m.LParam.ToInt32());
197                 }
198                 //Console.WriteLine(m.WParam + " " + m.LParam + ": " + System.Windows.Forms.Cursor.Position.X + "," + ↙
    System.Windows.Forms.Cursor.Position.Y);
199             }
200         }
201 
202         // Toggle highlight colors if in highlight mode and option checked
203         private void toggleHighlightColors()
204         {
205             if ((attention_type == Attention_Mode.Highlight) && cbToggleColor.Checked && highlightForm.Visible)
206             {
207                 Color c = btnColor.BackColor;
208                 byte r = c.R;
209                 byte g = c.G;
210                 byte b = c.B;
211                 btnColor.BackColor = Color.FromArgb(b, r, g);
212                 setHighlightForm();
213             }
214         }
215 
216         private void setHighlightForm()
217         {
218             if (circleHighlight)
219             {
220                 curBmp = new Bitmap(tbRadius.Value * 2, tbRadius.Value * 2);
221                 Graphics g = Graphics.FromImage(curBmp);
222                 SolidBrush b = new SolidBrush(btnColor.BackColor);
223                 g.FillEllipse(b, 0, 0, tbRadius.Value * 2, tbRadius.Value * 2);
224 
225                 highlightForm.SetBitmap(curBmp, (byte)(tbOpacity.Value * 255 / 100));
226                 highlightForm.Show();
227                 DrawPrimaryForm(System.Windows.Forms.Control.MousePosition.X - tbRadius.Value,
228                                 System.Windows.Forms.Control.MousePosition.Y - tbRadius.Value);
229 
230                 extendedHighlightForm.SetBitmap(curBmp, (byte)(tbOpacity.Value * 255 / 100));
231                 DrawExtendedForm(System.Windows.Forms.Control.MousePosition.X - tbRadius.Value,
232                                  System.Windows.Forms.Control.MousePosition.Y - tbRadius.Value, true);
233             }
234             else
235             {
236                 curBmp = new Bitmap(tbWidth.Value, tbHeight.Value);
237                 Graphics g = Graphics.FromImage(curBmp);
238                 SolidBrush b = new SolidBrush(btnColor.BackColor);
239                 g.FillRectangle(b, 0, 0, tbWidth.Value, tbHeight.Value);
```

```
240
241                 highlightForm.SetBitmap(curBmp, (byte)(tbOpacity.Value * 255 / 100));
242                 highlightForm.Show();
243                 DrawPrimaryForm(System.Windows.Forms.Control.MousePosition.X - (tbWidth.Value / 2),
244                                 System.Windows.Forms.Control.MousePosition.Y - (tbHeight.Value / 2));
245                 extendedHighlightForm.SetBitmap(curBmp, (byte)(tbOpacity.Value * 255 / 100));
246                 DrawExtendedForm(System.Windows.Forms.Control.MousePosition.X - (tbWidth.Value/2),
247                                  System.Windows.Forms.Control.MousePosition.Y - (tbHeight.Value/2), false);
248             }
249         }
```