```
 1  using System;
 2  using System.Collections.Generic;
 3  using System.ComponentModel;
 4  using System.Data;
 5  using System.Data.SqlClient;
 6  using System.Drawing;
 7  using System.IO;
 8  using System.Linq;
 9  using System.Reflection;
10  using System.Runtime.InteropServices;
11  using System.Text;
12  using System.Threading;
13  using System.Threading.Tasks;
14  using System.Windows.Forms;
15  using AppStatusControl;
16  using BSGlobals;
17
18  namespace AppStatusMonitor
19  {
20      public partial class frmMain : Form
21      {
22
23          int NumActivitiesPerMonitor = 15;
24          int LookbackInDays = 90;
25          const string JobName = "App Status Monitor";
26          List<string> AppNameList = new List<string>();
27          List<AppStatusUserControl> StatusMonitorList = new
              List<AppStatusUserControl>();
28          int TimerUpdateIntervalInMsec = 10000; // A default value for the timer
               update interval, which is read from config on startup
29          DateTime StartupTime = DateTime.Now;
30          Size MonitorSize = new Size(0, 0);
31          bool SingleLineMode = true;
32
33          public frmMain()
34          {
35              InitializeComponent();
36
37              // Get the refresh interval in seconds and convert to msec.
38              bool activitycountokay = int.TryParse
                  (Config.GetConfigurationKeyValue("AppStatusMonitor",
                  "NumActivitiesPerMonitor"), out NumActivitiesPerMonitor);
39              bool lookbackokay = int.TryParse(Config.GetConfigurationKeyValue
                  ("AppStatusMonitor", "LookbackInDays"), out LookbackInDays);
40              bool success = int.TryParse(Config.GetConfigurationKeyValue
                  ("AppStatusMonitor", "UpdateIntervalInSecs"), out int result);
41              if (success)
42              {
43                  TimerUpdateIntervalInMsec = 1000 * result;
```

```
44              }
45              timUpdateStatus.Enabled = true;
46
47              SetPanelSize();
48              DataIO.WriteToJobLog(BSGlobals.Enums.JobLogMessageType.INFO, "Job
                  starting", JobName);
49          }
50
51          private void SetPanelSize()
52          {
53              pnlMonitors.Size = new Size(this.ClientSize.Width - 8,
                  this.ClientSize.Height - 8);
54          }
55
56          private void timUpdateStatus_Tick(object sender, EventArgs e)
57          {
58              // First time in:  Initial timer value is 1 msec so that we get a
                  fast initial data load.
59              //   Afterward, set the timer update interval to whatever was read
                  from the config file
60              if (timUpdateStatus.Interval != TimerUpdateIntervalInMsec)
61              {
62                  timUpdateStatus.Interval = TimerUpdateIntervalInMsec;
63              }
64
65              // Get the names of all apps in the log table (for the last N days)
66              GetAppNames();
67
68              // For each app, get the last N cycles and check for errors
                  originating from within the app.
69              int NumCycles = NumActivitiesPerMonitor;
70
71              SqlParameter[] ActivityParams = new SqlParameter[3];
72              for (int i = 0; i < AppNameList.Count; i++)
73              {
74                  //  command.Parameters.Add(new SqlParameter("@MessageType",
                      type.ToString("d")));
75                  ActivityParams[0] = new SqlParameter("@pvchrJobName",
                      AppNameList[i]);
76                  ActivityParams[1] = new SqlParameter("@pvintLookbackInDays",
                      LookbackInDays);
77                  ActivityParams[2] = new SqlParameter("@pvintNumCycles",
                      NumCycles);
78                  SqlDataReader rdr = DataIO.ExecuteQuery
                      (Enums.DatabaseConnectionStringNames.EventLogs,
                      CommandType.StoredProcedure, "Proc_Select_Last_N_Activities",
                       ActivityParams); // A misnamed sproc.  Should be N, not 5
79                  UpdateMonitor(rdr, StatusMonitorList[i]);
80              }
```

```
 81          }
 82
 83
 84          private void GetAppNames()
 85          {
 86              try
 87              {
 88                  bool MonitorsNeedRecreating = false;
 89
 90                  // Get a list of all apps in the event log that have run in the
                        past N days
 91                  // Syntax:
 92                  //   Results is a list of dictionary entries of type
                       <string>,<object> as required by ExecuteSQL.
 93                  //   For each dictionary entry,
 94                  //        <string> will contain the field name
 95                  //        <object> will contrain the value for that field
                       (which must be explictly typed later)
 96                  //   Each entry in the list represents a single row from the
                       stored procedure.
 97                  List<Dictionary<string, object>> results =
 98                      DataIO.ExecuteSQL
                        (Enums.DatabaseConnectionStringNames.EventLogs,
 99                      "dbo.Proc_Select_List_Of_All_Apps",
100                      new SqlParameter("@pvintLookbackInDays",
                        Config.GetConfigurationKeyValue("AppStatusMonitor",
                        "LookbackInDays")));
101
102                  foreach (Dictionary<string, object> entry in results)
103                  {                    // <object> will be the AppName, once it's
                        converted to a string.
104                      string appname = ((string)entry["JobName"]);
105
106                      // Check if this name is already on the app list.  If not,
107                      //    Add it to the list
108                      //    Mark that monitors need to be recreated.
109                      if (!AppNameList.Contains(appname))
110                      {
111                          AppNameList.Add(appname);
112                          MonitorsNeedRecreating = true;
113                      }
114                  }
115
116                  // If any monitor needs to be created, then
117                  //    Delete all existing monitors
118                  //    Recreate the monitor list in sort order
119
120                  if (MonitorsNeedRecreating)
121                  {
```

```
122                     DeleteAllMonitors();
123                     AppNameList.Sort();
124                     foreach (string name in AppNameList)
125                     {
126                         CreateMonitor(name);
127                     }
128                     ArrangeMonitors();
129                 }
130             }
131             catch (Exception ex)
132             {
133                 DataIO.WriteToJobLog(Enums.JobLogMessageType.ERROR, "Fix the darn program", JobName);
134             }
135         }
136 
137         private void ArrangeMonitors()
138         {
139             // Arrange the monitors to fit within the frame (with scrolling if necessary)
140             // What's the width of the panel interior and the monitor control?
141             //   And how many controls can we fit within the panel's width?
142             int panelx = pnlMonitors.Width;
143             int numpanelsacross = panelx / MonitorSize.Width;
144             if (numpanelsacross == 0)
145             {
146                 numpanelsacross = 1;
147             }
148 
149             // Separate the panels vertically as well
150             int numpanelsdown = (StatusMonitorList.Count + (numpanelsacross - 1)) / numpanelsacross;
151             for (int i = 0; i < numpanelsdown; i++)
152             {
153                 for (int j = 0; j < numpanelsacross; j++)
154                 {
155                     if (i * numpanelsacross + j < StatusMonitorList.Count)
156                     {
157                         AppStatusUserControl uc = StatusMonitorList[j + i * numpanelsacross];
158                         uc.Left = j * MonitorSize.Width;
159                         uc.Top = i * MonitorSize.Height;
160                     }
161                 }
162             }
163         }
164 
165         private void CreateMonitor(string name)
166         {
```

```
167             // Create an application monitor, and render it visible
168             AppStatusUserControl uc = new AppStatusUserControl
                  (NumActivitiesPerMonitor, SingleLineMode)
169             {
170                 AppName = name,
171                 Visible = true
172             };
173             StatusMonitorList.Add(uc);
174             this.pnlMonitors.Controls.Add(uc);
175
176             // Monitor size is the same for all monitors. Save it for later
                  use.
177             MonitorSize = new Size(uc.Width, uc.Height);
178
179             // Add a mouseclick event handler so we can use it to toggle
                  between display modes
180             //uc.ucMouse_Click += new EventHandler((sender, e) => ucMouse_Click
                  (sender, e));
181             uc.ucMouse_Click += ucMouse_Click;
182
183         }
184
185         private void DeleteAllMonitors()
186         {
187             // Destroy all monitors
188
189             //foreach (AppStatusUserControl uc in pnlMonitors.Controls)  Can't
                  use this approach because we're deleting controls and will skip
                  some as the control list compresses
190             for (int i = pnlMonitors.Controls.Count -1; i >= 0; i--)
191             {
192                 if (pnlMonitors.Controls[i] is AppStatusUserControl)
193                 {
194                     AppStatusUserControl uc = (AppStatusUserControl)
                          pnlMonitors.Controls[i];
195                     pnlMonitors.Controls.Remove(uc);
196                     //uc.Dispose(); // Is this needed?
197                 }
198             }
199             StatusMonitorList.Clear();
200             pnlMonitors.Refresh();
201         }
202
203         private void UpdateMonitor(SqlDataReader rdr, AppStatusUserControl
              appStatusUserControl)
204         {
205             // Update the selected data monitor
206
207             // The SQL data reader passed into this routine should have 3
```

```
                datasets attached to it:
208             //   - A list of the last N dates (or fewer) of this app's
                acitivity that was other than "started/completed"
209             //   - A list of all warnings and errors from the jobs that ran
                during any of those dates
210             //   - The app's very last starting/completed message to determine
                if the app is still running
211
212             try
213             {
214                 // First result:  The list of the last N activity dates
215                 List<DateTime> ActivityDates = new List<DateTime>();
216                 while (rdr.Read())
217                 {
218                     ActivityDates.Add((DateTime)rdr["LogDate"]);
219                 }
220
221                 // Second result:  The list of all warnings and errors
                    (containing LogDate, MessageType and Message)
222                 List<IssuesType> IssuesList = new List<IssuesType>();
223                 rdr.NextResult();
224                 while (rdr.Read())
225                 {
226                     IssuesType issue = new IssuesType
227                     {
228                         LogDate = (DateTime)rdr["LogDate"],
229                         MessageType = (int)rdr["MessageType"],
230                         Message = rdr["Message"].ToString()
231                     };
232                     IssuesList.Add(issue);
233                 }
234
235                 // Third result:  The app's last starting or completed message.
                        This will be either zero or one record in length
236                 bool AppIsRunning = false;
237                 DateTime LastExecutionTime = new DateTime(1900, 01, 01);
238                 rdr.NextResult();
239                 while (rdr.Read())
240                 {
241                     AppIsRunning = (rdr["Message"].ToString() == "Job
                        starting") ? true : false;
242                     LastExecutionTime = (DateTime)rdr["LogDate"];
243                 }
244                 rdr.Close();
245
246                 Color color = (AppIsRunning) ? Color.White : Color.Blue;
247                 appStatusUserControl.SetLEDColor
                    (AppStatusUserControl.LEDs.LEDActivity, 0, color);
248
```

```
249                 // Determine which activities had errors or warnings
250                 List<LEDStatusesType> LEDStatuses = ComputeLEDStatuses
                      (appStatusUserControl, ActivityDates, IssuesList);
251
252                 // and light the appropriate leds the appropriate color.
253                 for (int i = 0; i < LEDStatuses.Count; i++)
254                 {
255                     appStatusUserControl.SetLEDColor
                        (AppStatusUserControl.LEDs.LEDStatus, i, LEDStatuses
                        [i].LEDColor);
256                 }
257                 appStatusUserControl.ClearLEDs(LEDStatuses.Count); // This
                      clears (turns off) any remaining LEDs.
258
259                 // Set the current runtime value to the last execution time in
                      the log
260
261                 appStatusUserControl.RunTime = LastExecutionTime;
262             }
263             catch (Exception ex)
264             {
265                 DataIO.WriteToJobLog(Enums.JobLogMessageType.ERROR, "Failed to
                      correctly update monitor " + appStatusUserControl.AppName +
                      ": " + ex.ToString(), appStatusUserControl.AppName);
266             }
267
268         }
269
270         private List<LEDStatusesType> ComputeLEDStatuses(AppStatusUserControl
              appStatusUserControl, List<DateTime> activityDates, List<IssuesType>
              issuesList)
271         {
272             // Take the list of issues and bounce them across the list of
                  activity dates to determine in which date range the issue arose.
273             //   Return the appropriate LED color for each activity date
274
275             List<LEDStatusesType> LEDStatusList = new List<LEDStatusesType>();
276             for (int i = 0; i < activityDates.Count; i++)
277             {
278                 LEDStatusList.Add(new LEDStatusesType());
279             }
280
281             try
282             {
283                 // Find out which activity this issue belongs to
284                 for (int j = 0; j < issuesList.Count; j++)
285                 {
286                     IssuesType issue = issuesList[j];
287                     for (int i = 0; i < activityDates.Count - 1; i++)
```

```
288                     {
289                         // is it activity [i]?
290                         string Messages = "";
291                         if ((issue.LogDate <= activityDates[i]) &&
                        (issue.LogDate >= activityDates[i + 1]))
292                         {
293                             // Why yes it is!  Set the activity's LED to either
                         yellow (if it was green) or red (unconditionally) based on
                         the message type.
294                             Messages = activityDates[i].ToShortDateString() +  "
                         " + activityDates[i].ToShortTimeString();
295                             issue.LEDNum = i;
296                             issuesList[j] = issue;  // Save this; we'll use it
                        when hovering over a LED
297                             LEDStatusesType ledstatus = LEDStatusList[i];
298                             switch (issue.MessageType)
299                             {
300                                 case 1:
301                                     ledstatus.LEDColor = Color.Green;
302                                     break;
303                                 case 2:
304                                     if (ledstatus.LEDColor == Color.Green)
305                                     {
306                                         ledstatus.LEDColor = Color.Yellow;
307                                     }
308                                     break;
309                                 case 3:
310                                     ledstatus.LEDColor = Color.Red;
311                                     break;
312                                 default:
313                                     break;
314                             }
315
316                             // Save the message as well...
317                             Messages += "\r\n" + issue.Message;  //
                        TBD .Messages is Unnecessary, get rid of it in the class
318                             // And save the status message back to the control
                        for later tool tipping
319                             LEDStatusList[i] = ledstatus;  // TBD Unnecessary
320                             appStatusUserControl.SetLEDMessage(i, Messages);
321                             break;
322                         }
323                     }
324                 }
325             }
326             catch (Exception ex)
327             {
328                 throw new Exception("Error trying to update LED status: " +
                      ex.ToString());
```

```
329             }
330             return (LEDStatusList);
331         }
332 
333         private class IssuesType
334         {
335             public DateTime LogDate { get; set; }
336             public int MessageType { get; set; }
337             public string Message { get; set; }
338             public int LEDNum { get; set; }
339 
340             public IssuesType()
341             {
342                 LogDate = DateTime.Now;
343                 MessageType = 0;
344                 Message = "";
345                 LEDNum = -1;
346             }
347         }
348 
349         private class LEDStatusesType
350         {
351             public Color LEDColor { get; set; }
352             public LEDStatusesType()
353             {
354                 LEDColor = Color.Green;
355             }
356         }
357 
358         private void frmMain_FormClosing(object sender, FormClosingEventArgs e)
359         {
360             DateTime StopTime = DateTime.Now;
361             double ElapsedTime = ((TimeSpan)(StopTime -
                  StartupTime)).TotalSeconds;
362             TimeSpan t = TimeSpan.FromSeconds(ElapsedTime);
363             string result = string.Format("{0:D2}h:{1:D2}m:{2:D2}.{3:D3}s",
                  t.Hours, t.Minutes, t.Seconds, t.Milliseconds);
364             DataIO.WriteToJobLog(BSGlobals.Enums.JobLogMessageType.INFO,
                  "Runtime: " + result, JobName);
365             DataIO.WriteToJobLog(BSGlobals.Enums.JobLogMessageType.INFO, "Job
                  completed", JobName);
366         }
367 
368         private void frmMain_ResizeEnd(object sender, EventArgs e)
369         {
370             // At the end of a form resize, redistribute the existing monitors
371             SetPanelSize();
372             ArrangeMonitors();
373         }
```

```
374
375         private void AppStatusMonitor_Hover(object sender, EventArgs e)
376         {
377             // TBD THIS IS OBSOLETE (NEVER HIT AND NOT NEEDED)
378             // Mouse just hovered over a LED.  Get the LED's index and the name
                of the app that triggered this event
379             AppStatusUserControl uc = (AppStatusUserControl)sender;
380             int lednum = uc.LEDNum;
381             string appname = uc.AppName;
382             string msg = uc.GetLEDMessage(lednum);
383         }
384
385         private void ucMouse_Click(object sender, EventArgs e)
386         {
387             ToggleDisplay();
388         }
389
390         private void pnlMonitors_Click(object sender, EventArgs e)
391         {
392             ToggleDisplay();
393         }
394
395         private void ToggleDisplay()
396         {
397             // Toggle the display between single line and multiline
398 #if false
399             for (int i = 0; i < StatusMonitorList.Count; i++)
400             {
401                 AppStatusUserControl uc = StatusMonitorList[i];
402                 uc.ToggleDisplayMode();
403             }
404 #else
405             SingleLineMode = !SingleLineMode;
406             DeleteAllMonitors();
407             AppNameList.Sort();
408             foreach (string name in AppNameList)
409             {
410                 CreateMonitor(name);
411             }
412             ArrangeMonitors();
413 #endif
414         }
415     }
416 }
417
```