

```
1 using System;
2 using System.Collections.Generic;
3 using System.Drawing;
4 using System.Windows.Forms;
5 using System.Runtime.InteropServices;
6 using System.IO;
7
8 //-----\\
9 // Author: Nikola Nejedlý \\
10 // Kybersoft (c) 2017 \\
11 // Web: http://nejedniko.tk \\
12 // TUL | FM:IL - Project \\
13 // .NET 3.5 - support winXP \\
14 //-----\\
15
16
17 namespace GifSlider
18 {
19     public partial class Slideshow : Form
20     {
21         double scale = 1;
22         public Form TheParent;
23         string filenow = "";
24         int count = 0;
25         public int time = 3000;
26         int id = 0;
27         public bool prevent_idle = false;
28
29         Image temp;
30
31         public bool random = false;
32         public bool repeatrand = false;
33
34         public bool topm = false;
35
36         public bool playsong = false;
37         public string song;
38         public bool loopsong = true;
39         List<bool> random_order_checkup = new List<bool>();
40
41         public bool nextontime = true;
42         public int startat = 0;
43         public Int16 imagefit = 0; // 0 - stretch; 1 - fit; 2 - 1:1
44
45         public int vol = 0;
46
47         WMPLib.WindowsMediaPlayer wplayer = new WMPLib.WindowsMediaPlayer();
48
49         public Slideshow()
50         {
51             InitializeComponent();
52             timer.Enabled = false;
53         }
54
55         public void playMp3(string file, int vol, bool repeat)
56         {
```

```
57         if (!File.Exists(file))
58             return;
59
60         wplayer.URL = file;
61         wplayer.settings.volume = vol;
62
63         if (repeat)
64             wplayer.settings.setMode("loop", true);
65         else
66             wplayer.settings.setMode("loop", false);
67
68         wplayer.controls.play();
69     }
70
71     public void stopMp3()
72     {
73         wplayer.controls.stop();
74     }
75
76     private void Slideshow_Load(object sender, EventArgs e)
77     {
78         playMp3(song, vol, loopsong);
79
80         if (prevent_idle)
81         {
82             uint fPreviousExecutionState =
83                 NativeMethods.SetThreadExecutionState(
84                     NativeMethods.ES_CONTINUOUS | NativeMethods.ES_SYSTEM_REQUIRED);
85             if (fPreviousExecutionState == 0)
86             {
87                 MessageBox.Show("Something went wrong. Can't prevent
88                     Windows IDLE state. \r\nSlideshow will continue without
89                     it.", "ERROR", MessageBoxButtons.OK, MessageBoxIcon.Error);
90             }
91         }
92
93         int htemp = Screen.PrimaryScreen.WorkingArea.Size.Height;
94         int wtemp = Screen.PrimaryScreen.WorkingArea.Size.Width;
95         this.SetDesktopLocation(wtemp / 2 - this.Width / 2, htemp / 2 -
96             this.Height / 2);
97
98         if (topm)
99             SetWindowPos(this.Handle, HWND_TOPMOST, 100, 100, 300, 300,
100                 TOPMOST_FLAGS);
101
102         foreach(var item in Form1.imglist)
103             random_order_checkup.Add(false);
104
105         if (startat >= 0)
106             id = startat;
107         else
108             id = 0;
109
110         random_order_checkup[id] = true;
111
112         Point p = new Point(0, 0);
```

```
108         pictureBox1.Location = p;
109         pictureBox1.Width = this.Width;
110         pictureBox1.Height = this.Height;
111
112         timer.Interval = time;
113         count = Form1.imglist.Count;
114
115         updatePB();
116     }
117
118     private Point fit(Image origin, int wi, int he)
119     {
120         int sourceWidth = origin.Width;
121         int sourceHeight = origin.Height;
122         double destX = 0;
123         double destY = 0;
124
125         double nScale = 0;
126         double nScaleW = 0;
127         double nScaleH = 0;
128
129         nScaleW = ((double)wi / (double)sourceWidth);
130         nScaleH = ((double)he / (double)sourceHeight);
131         nScale = Math.Min(nScaleH, nScaleW);
132         destY = (he - sourceHeight * nScale) / 2;
133         destX = (wi - sourceWidth * nScale) / 2;
134
135         int destWidth = (int)Math.Round(sourceWidth * nScale);
136         int destHeight = (int)Math.Round(sourceHeight * nScale);
137         return new Point(destWidth, destHeight);
138     }
139
140     private void updatePB()
141     {
142
143         //imagefit | 0 - stretch; 1 - fit; 2 - 1:1
144         int Wid = this.ClientSize.Width;
145         int Hei = this.ClientSize.Height;
146         Point p = new Point(0, 0);
147
148         if (!(Form1.imglist.Count > 0))
149             return;
150
151         filenow = Form1.imglist[id];
152
153         if (temp != null)
154             temp.Dispose();
155         try
156         {
157             temp = Image.FromFile(filenow);
158         }
159         catch
160         {
161             temp = null;
162         }
163     }
```

```
164         if (temp == null)
165             return;
166
167         if (imagefit == 0)
168         {
169             pictureBox1.Width = (int)(Wid * scale);
170             pictureBox1.Height = (int)(Hei * scale);
171             if (scale != 1)
172                 p = new Point(Wid / 2 - pictureBox1.Width/2, Hei/2 - pictureBox1.Height/2);
173             pictureBox1.Location = p;
174         }
175         else if (imagefit == 1)
176         {
177             Point wh = fit(temp, Wid, Hei);
178
179             pictureBox1.Height = (int)(wh.Y * scale);
180             pictureBox1.Width = (int)(wh.X * scale);
181             p = new Point(Wid / 2 - pictureBox1.Width / 2 , Hei / 2 - pictureBox1.Height / 2);
182             pictureBox1.Location = p;
183
184         }
185         else if (imagefit == 2)
186         {
187             int img_w = (int)(temp.Width * scale);
188             int img_h = (int)(temp.Height * scale);
189
190             p = new Point(Wid / 2 - img_w / 2, Hei / 2 - img_h / 2);
191             pictureBox1.Location = p;
192
193             pictureBox1.Width = img_w;
194             pictureBox1.Height = img_h;
195
196         }
197         pictureBox1.Image = Image.FromFile(filenow);
198     }
199
200     private void Slideshow_ResizeEnd(object sender, EventArgs e)
201     {
202         updatePB();
203     }
204
205     private void timer1_Tick(object sender, EventArgs e)
206     {
207         if (!nextontime)
208             return;
209
210         if(!random)
211         {
212             if (id < count - 1)
213                 id++;
214             else
215                 id = 0;
216         }
217     }
```

```
218         if (random && repeatrand)
219         {
220             Random rr = new Random(Environment.TickCount);
221             id = (int)Math.Floor((double)rr.Next(0, count));
222         }
223         else
224         if (random && !repeatrand)
225         {
226             Random rr = new Random(Environment.TickCount);
227             id = (int)Math.Floor((double)rr.Next(0, count));
228
229             while (random_order_checkup[id] == true)
230                 id = (int)Math.Floor((double)rr.Next(0, count));
231
232             random_order_checkup[id] = true;
233
234             bool alldone = true;
235             foreach (var item in random_order_checkup)
236             {
237                 if (item == false)
238                     alldone = false;
239             }
240             if (alldone)
241             {
242                 random_order_checkup.Clear();
243                 foreach (var item in Form1.imglist)
244                 {
245                     random_order_checkup.Add(false);
246                 }
247             }
248         }
249
250         updatePB();
251     }
252
253     private void Slideshow_FormClosed(object sender, FormClosedEventArgs e)
254     {
255         Form1.slides = new Slideshow();
256         stopMp3();
257         TheParent.Enabled = true;
258         TheParent.Show();
259     }
260
261     private void Slideshow_KeyPress(object sender, KeyPressEventArgs e)
262     {
263         if (e.KeyChar == (char)Keys.Return || e.KeyChar == (char)Keys.Enter)
264         {
265             e.Handled = true;
266         }
267
268         if (e.KeyChar == (char)Keys.Escape)
269         {
270             this.Close();
271             e.Handled = true;
272         }
273     }
```

```
272     }
273 }
274
275 private void Slideshow_KeyDown(object sender, KeyEventArgs e)
276 {
277     if (e.KeyCode == Keys.Left)
278     {
279         if (id > 0)
280             id--;
281         else
282             id = count - 1;
283
284         updatePB();
285         timer.Stop();
286         timer.Start();
287         e.Handled = true;
288     }
289     else
290     if (e.KeyCode == Keys.Right)
291     {
292         if (id < count - 1)
293             id++;
294         else
295             id = 0;
296
297         updatePB();
298         timer.Stop();
299         timer.Start();
300         e.Handled = true;
301     }
302     else
303     if (e.KeyCode == Keys.Up)
304     {
305         if (scale < 2)
306             scale += 0.05;
307         else
308             scale = 2;
309
310         updatePB();
311         e.Handled = true;
312     }
313     else
314     if (e.KeyCode == Keys.Down)
315     {
316         if (scale > 0.1)
317             scale -= 0.05;
318         else
319             scale = 0.1;
320
321         updatePB();
322         e.Handled = true;
323     }
324 }
325
326 private void Slideshow_DoubleClick(object sender, EventArgs e)
327 {
```

```

328         this.Close();
329     }
330
331     private void Slideshow_Resize(object sender, EventArgs e)
332     {
333         updatePB();
334     }
335
336     private void Slideshow_MouseDoubleClick(object sender, MouseEventArgs e)
337     {
338         this.Close();
339     }
340
341
342     // ----- || StayOnTop -----
343
344     private static readonly IntPtr HWND_TOPMOST = new IntPtr(-1);
345     private const UInt32 SWP_NOSIZE = 0x0001;
346     private const UInt32 SWP_NOMOVE = 0x0002;
347     private const UInt32 TOPMOST_FLAGS = SWP_NOMOVE | SWP_NOSIZE;
348
349     [DllImport("user32.dll")]
350     [return: MarshalAs(UnmanagedType.Bool)]
351     public static extern bool SetWindowPos(IntPtr hWnd, IntPtr
352         hWndInsertAfter, int X, int Y, int cx, int cy, uint uFlags);
353
354     protected override void WndProc(ref Message m)
355     {
356         const int RESIZE_HANDLE_SIZE = 10;
357
358         switch (m.Msg)
359         {
360             case 0x0084/*NCHITTEST*/ :
361                 base.WndProc(ref m);
362
363                 if ((int)m.Result == 0x01/*HTCLIENT*/)
364                 {
365                     Point screenPoint = new Point(m.LParam.ToInt32());
366                     Point clientPoint = this.PointToClient(screenPoint);
367                     if (clientPoint.Y <= RESIZE_HANDLE_SIZE)
368                     {
369                         if (clientPoint.X <= RESIZE_HANDLE_SIZE)
370                             m.Result = (IntPtr)13/*HTTOPLEFT*/ ;
371                         else if (clientPoint.X < (Size.Width -
372                             RESIZE_HANDLE_SIZE))
373                             m.Result = (IntPtr)12/*HTTOP*/ ;
374                         else
375                             m.Result = (IntPtr)14/*HTTOPRIGHT*/ ;
376                     }
377                     else if (clientPoint.Y <= (Size.Height -
378                         RESIZE_HANDLE_SIZE))
379                     {
380                         if (clientPoint.X <= RESIZE_HANDLE_SIZE)

```

```

378         m.Result = (IntPtr)10/*HTLEFT*/ ;
379         else if (clientPoint.X < (Size.Width -
RESIZE_HANDLE_SIZE))
380             m.Result = (IntPtr)2/*HTCAPTION*/ ;
381         else
382             m.Result = (IntPtr)11/*HTRIGHT*/ ;
383     }
384     else
385     {
386         if (clientPoint.X <= RESIZE_HANDLE_SIZE)
387             m.Result = (IntPtr)16/*HTBOTTOMLEFT*/ ;
388         else if (clientPoint.X < (Size.Width -
RESIZE_HANDLE_SIZE))
389             m.Result = (IntPtr)15/*HTBOTTOM*/ ;
390         else
391             m.Result = (IntPtr)17/*HTBOTTOMRIGHT*/ ;
392     }
393 }
394 return;
395 }
396 base.WndProc(ref m);
397 }
398
399 protected override CreateParams CreateParams
400 {
401     get
402     {
403         CreateParams cp = base.CreateParams;
404         cp.Style |= 0x20000;
405         return cp;
406     }
407 }
408
409 // -----
----- || Prevent IDLE
410
411 internal static class NativeMethods
412 {
413     [DllImport("kernel32.dll")]
414     public static extern uint SetThreadExecutionState(uint esFlags);
415     public const uint ES_CONTINUOUS = 0x80000000;
416     public const uint ES_SYSTEM_REQUIRED = 0x00000001;
417 }
418
419 }
420 }
421

```