```
1
2   File : \ConnectDialog.cs : Last SVN Checkout 23-Apr-21 8:52:07 AM
3   //Submission code : 1202_CMPE2800_MDClient
4   //Ezekiel Enns
5   //connection dialog
6   //allows user to connect to a remote or local server
7   using System;
8   using System.Collections.Generic;
9   using System.ComponentModel;
10  using System.Data;
11  using System.Drawing;
12  using System.Linq;
13  using System.Net.Sockets;
14  using System.Text;
15  using System.Threading.Tasks;
16  using System.Windows.Forms;
17
18  namespace DrawClientEzekielEnns
19  {
20      public partial class ConnectDialog : Form
21      {
22          //connetion socket object
23          public Socket connection;
24
25          public ConnectDialog()
26          {
27              InitializeComponent();
28              _HostName.SelectedIndex = 0;
29              _connect.Click += _connect_Click;
30              _cancle.Click += _cancle_Click;
31          }
32
33          /// <summary>
34          /// cancels dialog
35          /// </summary>
36          /// <param name="sender"></param>
37          /// <param name="e"></param>
38          private void _cancle_Click(object sender, EventArgs e)
39          {
40              DialogResult = DialogResult.Cancel;
41          }
42
43          /// <summary>
44          /// atempts to connect to server
45          /// </summary>
46          /// <param name="sender"></param>
47          /// <param name="e"></param>
48          private void _connect_Click(object sender, EventArgs e)
49          {
50              try
```

94

Doc Ded -5

Good.

```
51              {
52                  //creating socekt and trying to connect
53                  _connect.Enabled = false;
54                  connection = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);
55                  connection.BeginConnect(_HostName.Text, (int)_port.Value, connecting, null);
56                  _connect.Text = "Connecting";
57              }
58              catch (Exception er)
59              {
60                  DialogResult result =
61                      MessageBox.Show($"{er.Message}", "Error", MessageBoxButtons.OK);
62                  _connect.Enabled = true;
63                  _connect.Text = "Connect";
64              }
65          }
66
67          /// <summary>
68          /// establishes connection
69          /// </summary>
70          /// <param name="ar"></param>
71          private void connecting(IAsyncResult ar)
72          {
73              try
74              {
75                  //ending connection and returning result
76                  connection.EndConnect(ar);
77                  connection.NoDelay = true;
78                  if (InvokeRequired)
79                      Invoke(new Action(() => DialogResult = DialogResult.OK));
80                  else
81                      DialogResult = DialogResult.OK;
82              }
83
84              //connection failed let user try another server
85              catch (Exception e)
86              {
87                  DialogResult result =
88                      MessageBox.Show($"{e.Message}", "Error", MessageBoxButtons.OK);
89                  if (InvokeRequired)
90                      Invoke(new Action(() => { _connect.Enabled = true; _connect.Text = "Connect"; }));
91                  else
92                  {
93                      _connect.Enabled = true;
94                      _connect.Text = "Connect";
95                  }
96              }
97          }
98      }
99  }
100
```

```
File : \Form1.cs : Last SVN Checkout 27-Apr-21 4:26:51 PM
//Submission code : 1202_CMPE2800_MDClient
//Ezekiel Enns
//form allows user to create lines on a server
//and draws lines from a server
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using mdtypes;
using GenSox;

namespace DrawClientEzekielEnns
{
    public partial class Form1 : Form
    {
        GenSocket _connection;          //generic connection
        LineSegment _line;              //the current line being sent
        Color _color = Color.Red;       //line color
        ushort _thickness = 20;         //line thickness
        byte _Alpha = 255;              //line alpha
        bool chgAlpha = false;          //determins weather alpha should be changed on scroll
        Timer update = new Timer();     //for updating data

        public Form1()
        {
            InitializeComponent();

            //set up timer
            update.Enabled = true;
            update.Interval = 25;
            update.Tick += Update_Tick;

            //event listeners
            MouseDown += StartLine;
            MouseMove += SendLine;
            _menuStrip.Items["Connection"].Click += ConnectionBtn;
            _menuStrip.Items["Colour"].Click += ColourDialog;
            KeyDown += Form1_KeyDWN;
            KeyUp += Form1_KeyUp;
            MouseWheel += Form1_MouseWheel;

            //gui setup
            _menuStrip.Items["Thickness"].BackColor = Color.Black;
            _menuStrip.Items["Thickness"].ForeColor = Color.White;
```

```
151             _menuStrip.Items["Colour"].ForeColor = Color.Red;
152         }
153
154         /// <summary>
155         /// upadates manue as to not bog down other processes
156         /// </summary>
157         /// <param name="sender"></param>
158         /// <param name="e"></param>
159         private void Update_Tick(object sender, EventArgs e)
160         {
161             //dont update if not connected
162             if (_connection == null) return;
163
164             //update menue
165             if (_menuStrip.InvokeRequired)
166             {
167                 _menuStrip.Invoke(new Action(() =>
168                 {
169                     _menuStrip.Items["FrameRecv"].Text = $"Frames RX'ed : {_connection.Stats.Frames}";
170                     _menuStrip.Items["Fragments"].Text = $"Fragments : {_connection.Stats.frags}";
171                     _menuStrip.Items["DestackAvg"].Text = $"Destack Avg : {_connection.Stats.deAvg:0.00}";
172                     _menuStrip.Items["BytesRX"].Text = $"Bytes RXed : {_connection.Stats.byt}";
173                 }));
174             }
175             else
176             {
177                 _menuStrip.Items["FrameRecv"].Text = $"Frames RX'ed : {_connection.Stats.Frames}";
178                 _menuStrip.Items["Fragments"].Text = $"Fragments : {_connection.Stats.frags}";
179                 _menuStrip.Items["DestackAvg"].Text = $"Destack Avg : {_connection.Stats.deAvg:0.00}";
180                 _menuStrip.Items["BytesRX"].Text = $"Bytes RXed : {_connection.Stats.byt}";
181             }
182         }
183
184         /// <summary>
185         /// sets the mousewheel event to change thickness
186         /// </summary>
187         /// <param name="sender"></param>
188         /// <param name="e"></param>
189         private void Form1_KeyUp(object sender, KeyEventArgs e)
190         {
191
192         }
193
194         /// <summary>
195         /// sets mousewheel event to change alpha
196         /// </summary>
197         /// <param name="sender"></param>
198         /// <param name="e"></param>
199         private void Form1_KeyDWN(object sender, KeyEventArgs e)
200         {
```

? Should it, unless invoked across threads, Timers are UI threads

Ghost Code?

```
201             if (e.KeyCode == Keys.A)
202             {
203                 if (chgAlpha)
204                 {
205                     chgAlpha = false;
206                     _menuStrip.Items["Alpha"].BackColor = BackColor;
207                     _menuStrip.Items["Alpha"].ForeColor = Color.Black;
208 
209                     _menuStrip.Items["Thickness"].BackColor = Color.Black;
210                     _menuStrip.Items["Thickness"].ForeColor = Color.White;
211                 }
212                 else
213                 {
214                     chgAlpha = true;
215                     _menuStrip.Items["Alpha"].BackColor = Color.Black;
216                     _menuStrip.Items["Alpha"].ForeColor = Color.White;
217 
218                     _menuStrip.Items["Thickness"].BackColor = BackColor;
219                     _menuStrip.Items["Thickness"].ForeColor = Color.Black;
220                 }
221             }
222 
223         }
224 
225         /// <summary>
226         /// changes either alpha or thickness
227         /// </summary>
228         /// <param name="sender"></param>
229         /// <param name="e"></param>
230         private void Form1_MouseWheel(object sender, MouseEventArgs e)
231         {
232             if (chgAlpha)
233             {
234                 //checking to see if new alpha is in bounds
235                 byte data = (byte)(_Alpha+(e.Delta / 100));
236                 if (data >= 1 && data <= 255)
237                     _Alpha = data;
238 
239                 //wrap around for alpha
240                 else if (data == 0 && e.Delta > 0)
241                     _Alpha = 1;
242                 else if (data == 0 && e.Delta < 0)
243                     _Alpha = 255;
244 
245                 //updating menu
246                 _menuStrip.Items["Alpha"].Text = $"Alpha : {_Alpha}";
247             }
248             else
249             {
250                 //checking if new thickness is in bounds
```

```
251                 ushort data = (ushort)(_thickness + (e.Delta / 100));
252                 if (data >= 1)
253                     _thickness = data;
254 
255                 //updating menu
256                 _menuStrip.Items["Thickness"].Text = $"Thickness : {_thickness}";
257             }
258         }
259 
260         /// <summary>
261         /// upadates line color and color of menu item
262         /// </summary>
263         /// <param name="sender"></param>
264         /// <param name="e"></param>
265         private void ColourDialog(object sender, EventArgs e)
266         {
267             if (sender is ToolStripItem tsi)
268             {
269                 ColorDialog cd = new ColorDialog();
270                 if (cd.ShowDialog() == DialogResult.OK)
271                 {
272                     _color = cd.Color;
273                     tsi.ForeColor = _color;
274                 }
275             }
276         }
277 
278         /// <summary>
279         /// opens a connection dialog and connects gen socket
280         /// </summary>
281         /// <param name="sender"></param>
282         /// <param name="e"></param>
283         private void ConnectionBtn(object sender, EventArgs e)
284         {
285             if (sender is ToolStripItem tsi)
286             {
287                 //when connection is a;ready established
288                 if(_connection != null)
289                 {
290                     if (_connection.Connected)
291                     {
292                         //clear screen and close the connection
293                         Graphics g = CreateGraphics();
294                         _connection.Connected = false;
295                         tsi.Text = "Dissconnected";
296                         tsi.ForeColor = Color.Red;
297                         g.Clear(BackColor);
298 
299                         return;
300                     }
```

```
301             }
302
303             //when a connection isnt already made get user to pick
304             ConnectDialog connect = new ConnectDialog();
305             if (DialogResult.OK == connect.ShowDialog())
306             {
307                 //create new connection and link events
308                 _connection = new GenSocket(connect.connection);
309                 _connection.dataReady += _connection_dataReady;
310                 _connection.SocketError += _connection_SocketError;
311                 tsi.Text = "Connected";
312                 tsi.ForeColor = Color.Green;
313             }
314             else
315             {
316                 //on fail show there is no connection
317                 tsi.Text = "Dissconnected";
318                 tsi.ForeColor = Color.Red;
319             }
320
321         }
322     }
323
324     /// <summary>
325     /// event that triggers on errors genrated by the gensoxcket class
326     /// </summary>
327     /// <param name="sender"></param>
328     /// <param name="e"></param>
329     private void _connection_SocketError(object sender, EventArgs e)
330     {
331         //showing there was a disco error
332         DialogResult result =
333             MessageBox.Show($"you were Discoed", "Error", MessageBoxButtons.OK);
334         //updating menu
335         if (_menuStrip.InvokeRequired)
336         {
337             _menuStrip.Invoke(new Action(() => {
338                 _menuStrip.Items["Connection"].Text = "Dissconnected";
339                 _menuStrip.Items["Connection"].ForeColor = Color.Red;
340             }));
341         }
342         else
343         {
344             _menuStrip.Items["Connection"].Text = "Dissconnected";
345             _menuStrip.Items["Connection"].ForeColor = Color.Red;
346         }
347
348         //clearing graphics
349         if (InvokeRequired)
350         {
```

```
351                 Invoke(new Action(() =>
352                 {
353                     Graphics g = CreateGraphics();
354                     g.Clear(BackColor);
355 
356                 }));
357             }
358             else
359             {
360                 Graphics g = CreateGraphics();
361                 g.Clear(BackColor);
362 
363             }
364         }
365 
366         /// <summary>
367         /// event that fires when ever data is recived and processed by genSocket
368         /// </summary>
369         /// <param name="obj"></param>
370         /// <param name="s"></param>
371         private void _connection_dataReady(object obj, GenSocket s)
372         {
373             //casting data
374             if (obj is LineSegment ls)
375             {
376                 //rendering data
377                 Graphics g = CreateGraphics();
378                 ls.Render(g);
379 
380 
381             }
382 
383         }
384 
385         /// <summary>
386         /// sends and upadtes current line while mouse is moved and left button is clicked
387         /// </summary>
388         /// <param name="sender"></param>
389         /// <param name="e"></param>
390         private void SendLine(object sender, MouseEventArgs e)
391         {
392             if (e.Button == MouseButtons.Left && _connection != null)
393             {
394                 //only when connected and the line has been init by startLine
395                 if (_line != null && _connection.Connected)
396                 {
397                     //updating line and sending
398                     _line.End = e.Location;
399                     _connection.Send(_line);
400 
```

```
401                     //making new line for next call
402                     _line = new LineSegment
403                     { Start = e.Location, Colour = _color, Thickness = _thickness, Alpha = _Alpha };
404                 }
405             }
406         }
407 
408         /// <summary>
409         /// on mouse click initalizes a new line for SendLine
410         /// </summary>
411         /// <param name="sender"></param>
412         /// <param name="e"></param>
413         private void StartLine(object sender, MouseEventArgs e)=>
414             _line = new LineSegment { Start = e.Location, Colour = _color, Thickness = _thickness, Alpha = _Alpha };
415 
416 
417 
418 
419     }
420 }
421 
422 File : \GenSox\GenSocket.cs : Last SVN Checkout 27-Apr-21 4:26:51 PM
423 //Submission code : 1202_CMPE2800_MDClient
424 //Ezekiel Enns
425 //A generic socket that sends data and recvs and processes data
426 using System;
427 using System.Collections.Generic;
428 using System.IO;
429 using System.Linq;
430 using System.Net.Sockets;
431 using System.Runtime.Serialization;
432 using System.Runtime.Serialization.Formatters.Binary;
433 using System.Text;
434 using System.Threading;
435 using System.Threading.Tasks;
436 using mdtypes;
437 
438 namespace GenSox
439 {
440     public class GenSocket
441     {
442         //stores recv data
443         private Queue<object> _RecvData = new Queue<object>();
444 
445         //stored sending data
446         private Queue<object> _SendData = new Queue<object>();
447 
448 
449         //deleafte for event
450         public delegate void delVoidLS(object obj, GenSocket s);
```

what if there isn't one?

-1

what if user changes these?

```
451         public event delVoidLS dataReady;
452         //event handler for errors
453         public event EventHandler SocketError;
454
455
456         //members for dstats
457         int _totalRecvs = 0;    //total recvs done
458         int _frames = 0;        //frames recved
459         uint _bytes = 0;        //byts recved
460         long _frag = 0;         //frame fragments
461
462         //stats for accessing members
463         public (int Frames, long frags, double deAvg, string byt) Stats
464         {
465             get
466             {
467                 //calcing destack avrage
468                 double deavg = _totalRecvs == 0 ? 0 : (double)_frames / (double)_totalRecvs;
469
470                 //counts for size of bytes
471                 int counter = 0;
472
473                 //temporary bytes for calcuatling size
474                 double tmp = _bytes;
475
476                 //a charcter that represents size
477                 char type = ' ';
478
479                 //calc size
480                 while( tmp > 1024)
481                 {
482                     tmp /= 1024.0;
483                     counter++;
484                 }
485
486                 //set type
487                 switch (counter)
488                 {
489                     case 1:
490                         type = 'K';
491                         break;
492                     case 2:
493                         type = 'M';
494                         break;
495                     case 3:
496                         type = 'G';
497                         break;
498                 }
499                 return (_frames, _frag, deavg, $"{tmp:.00}{type}B");
500             }
```

Sure, Why not..

```
501         }
502
503         //used to maintain thread syncronization
504         bool _connected = true;
505
506         //represtnets socket connection
507         public bool Connected
508         {
509             get=>_connected;
510             set
511             {
512                 //setting to false will dc the socekt
513                 if (!value)
514                 {
515                     _sox?.Disconnect(false);
516                     _sox?.Close();
517                 }
518                 _connected = value;
519             }
520         }
521
522         Socket _sox;    //connected socket for class
523
524         //dirty
525
526
527         /// <summary>
528         /// starts threads and sets socket up
529         /// </summary>
530         /// <param name="sox"></param>
531         public GenSocket(Socket sox)
532         {
533             _sox = sox;
534             sox.NoDelay = true;
535             Thread RecvTh = new Thread(new ParameterizedThreadStart(RecvThread)) { IsBackground = true };
536             Thread ProsTh = new Thread(ProssThread) { IsBackground = true };
537             Thread SendTh = new Thread(new ParameterizedThreadStart(SendThread)) { IsBackground = true };
538             RecvTh.Start(sox); ProsTh.Start(); SendTh.Start(sox);
539
540
541         }
542
543         /// <summary>
544         /// enques data onto send queue to be sent in data thread
545         /// </summary>
546         /// <param name="data"></param>
547         public void Send(LineSegment data)
548         {
549             //avoid sending null data
550             if (data != null)
```

```
551             {
552                 lock (_SendData)
553                 {
554                     _SendData.Enqueue(data);
555                 }
556             }
557         }
558 
559         #region Threads
560 
561         /// <summary>
562         /// while connected recives frames from socket
563         /// and enques them to recv queue
564         /// </summary>
565         /// <param name="conn"></param>
566         private void RecvThread(object conn)
567         {
568             if (conn is Socket sox)
569             {
570 
571                 byte[] buff = new byte[2048];
572                 MemoryStream ms = new MemoryStream();
573                 BinaryFormatter bf = new BinaryFormatter();
574 
575                 while (Connected)
576                 {
577                     try
578                     {
579                         //receiving data and updating memebers
580                         int iRecv = sox.Receive(buff);
581                         ++_totalRecvs;
582                         _bytes += (uint)iRecv;
583 
584                         //detecting disconnecets
585                         if (iRecv == 0)
586                         {
587                             System.Diagnostics.Trace.WriteLine("#RECV#Disco");
588                             lock (_RecvData)
589                                 _RecvData.Enqueue(this);
590                             //break is important to make sure threads are synced
591                             break;
592                         }
593 
594                         //add recive data to end of recerver  stream
595                         long lPos = ms.Position;
596                         ms.Seek(0, SeekOrigin.End);
597                         ms.Write(buff, 0, iRecv);
598                         ms.Position = lPos;
599 
600                         //attempt extraction
```

```
                    do
                    {
                        long lStart = ms.Position;
                        try
                        {
                            if (bf.Deserialize(ms) is object ls)
                            {
                                lock (_RecvData)
                                {
                                    _RecvData.Enqueue(ls);
                                    ++_frames;
                                }
                            }
                        }
                        catch (SerializationException)
                        {
                            //exit loop due to fragmentation
                            ++_frag;
                            ms.Position = lStart;
                            break;
                        }
                    } while (ms.Position < ms.Length);

                    //clear streem
                    if (ms.Position == ms.Length)
                    {

                        ms.Position = 0;
                        ms.SetLength(0);
                    }

                }
                //most likly thrown due to hard disco
                catch (Exception er)
                {
                    System.Diagnostics.Trace.WriteLine($"#RECV#{er.Message}");
                    lock (_RecvData)
                        _RecvData.Enqueue(this);
                }

                Thread.Sleep(0);
            }
            System.Diagnostics.Trace.WriteLine($"#RECV#TERM");
        }
    }

    /// <summary>
    /// processes recv data and fires events based on dequeued data
    /// </summary>
    private void ProssThread()
```

```
651         {
652             //contains dequeued data
653             object data = new object();
654
655             while (Connected)
656             {
657                 while (_RecvData.Count > 0)
658                 {
659                     lock (_RecvData)
660                         data = _RecvData.Dequeue();
661
662                     //when data is a GenSocket
663                     if (data == this)
664                     {
665                         System.Diagnostics.Trace.WriteLine($"#PROCESS ERROR#");
666                         //raising error event
667                         SocketError?.Invoke(this,new EventArgs());
668                         //clear recv data and shutdown connection
669                         _RecvData.Clear();
670                         Connected = false;
671
672                         //prevents any other errors and terminates thread
673                         break;
674                     }
675                     else if (data is object ls)
676                     {
677                         //when none error data is recved, raise data event
678                         dataReady?.Invoke(ls,this);
679                     }
680                 }
681
682                 Thread.Sleep(0);
683             }
684             System.Diagnostics.Trace.WriteLine($"#PROS#TERM");
685         }
686
687         /// <summary>
688         /// sends data to connection
689         /// </summary>
690         /// <param name="conn"></param>
691         private void SendThread(object conn)
692         {
693             //pull from Tx queue
694             //serilize
695             //send data
696             if (conn is Socket sox)
697             {
698                 BinaryFormatter bf = new BinaryFormatter();
699                 while (Connected)
700                 {
```

```
701
702                     while (_SendData.Count > 0)
703                     {
704                         object t = new object();
705                         MemoryStream ms = new MemoryStream();
706
707                         lock (_SendData)
708                             t = _SendData.Dequeue();
709
710                         //serializing object and sending it
711                         bf.Serialize(ms, t);
712                         try
713                         {
714                             sox.Send(ms.GetBuffer(), (int)ms.Length, SocketFlags.None);
715                         }
716                         catch (Exception e)
717                         {
718                             System.Diagnostics.Trace.WriteLine($"#SEND#{e.Message}");
719                         }
720
721                     }
722
723                     Thread.Sleep(0);
724                 }
725                 System.Diagnostics.Trace.WriteLine($"#SEND#TERM");
726             }
727         }
728
729         #endregion
730
731     }
732 }
733
734
735
736
```