```
 1  using static RabbitMQ_SendClient.General_Classes.ModbusConfig;
 2  using static RabbitMQ_SendClient.GlobalRabbitMqServerFunctions;
 3  using static RabbitMQ_SendClient.GlobalSerialFunctions;
 4  using static RabbitMQ_SendClient.SystemVariables;
 5
 6  namespace RabbitMQ_SendClient
 7  {
 8      using System;
 9      using System.Collections.Generic;
10      using System.Collections.ObjectModel;
11      using System.ComponentModel;
12      using System.Data;
13      using System.Data.SqlClient;
14      using System.Diagnostics;
15      using System.IO;
16      using System.IO.Ports;
17      using System.Linq;
18      using System.Text;
19      using System.Threading;
20      using System.Windows;
21      using System.Windows.Controls;
22      using System.Windows.Controls.DataVisualization.Charting;
23      using System.Windows.Forms;
24      using System.Windows.Media.Animation;
25      using System.Windows.Threading;
26      using EasyModbus.Exceptions;
27      using Newtonsoft.Json;
28      using RabbitMQ.Client;
29      using RabbitMQ.Client.Exceptions;
30      using UI;
31      using CheckBox = System.Windows.Controls.CheckBox;
32      using MessageBox = System.Windows.Forms.MessageBox;
33
34      /// <summary>
35      /// Main UI for RabbitMQ Client
36      /// </summary>
37      public partial class MainWindow : IDisposable
38      {
39          /// <summary>
40          /// RabbitMQ Server Information for setup
41          /// </summary>
42          protected internal static readonly ObservableCollection<CheckListItem>
              AvailableModbusSerialPorts =
43              new ObservableCollection<CheckListItem>();
44
45          protected internal static readonly ObservableCollection<CheckListItem> AvailableSerialPorts =
46              new ObservableCollection<CheckListItem>();
47
48          internal static ObservableCollection<MessageDataHistory>[] MessagesSentDataPair =
49              new ObservableCollection<MessageDataHistory>[0];
50
51          protected internal static ModbusControl[] ModbusControls = new ModbusControl[0];
52
53          private static readonly StackTrace StackTracing = new StackTrace();
54
55          private static readonly string DatabaseLoc =
56              AppDomain.CurrentDomain.BaseDirectory + "Database\\MessageData.mdf";
57
58          internal static double[] MessagesPerSecond = new double[0];
59
```

```
60          private readonly string _connString =
61              $"Data Source=(LocalDB)\\MSSQLLocalDB;AttachDbFilename=\"{DatabaseLoc}\";Integrated
                Security=True";
62
63          private readonly Dictionary<DispatcherTimer, Guid> _modbusTimerId = new
              Dictionary<DispatcherTimer, Guid>();
64
65          private readonly DateTime _previousTime = new DateTime();
66          private readonly DispatcherTimer _systemTimer = new DispatcherTimer();
67
68          /// <summary>
69          /// Mainline Executable to the RabbitMQ Client
70          /// </summary>
71          public MainWindow()
72          {
73              InitializeSerialPortCheckBoxes();
74              InitializeComponent();
75
76              GetFriendlyDeviceNames();
77
78              InitializeSerialPorts();
79              InitializeHeartBeatTimer();
80
81              _systemTimer.Start();
82          }
83
84          public static LineSeries[] Lineseries { get; set; } = new LineSeries[0];
85
86          public static string DeviceName { get; } = Environment.MachineName;
87
88          /// <summary>
89          /// Close all open channels and serial ports before system closing
90          /// </summary>
91          public void Dispose()
92          {
93              for (var i = 0; i < SerialPort.GetPortNames().Length; i++)
94              {
95                  while (SerialPorts[i].IsOpen)
96                  {
97                      SerialPorts[i].Close();
98                  }
99                  SerialPorts[i].Dispose();
100                 CloseSerialPortUnexpectedly(i);
101             }
102
103             //Disposes of timer in a threadsafe manner
104             if (_systemTimer.IsEnabled)
105                 _systemTimer.Stop();
106         }
107
108         private void GetFriendlyDeviceNames()
109         {
110             var loaded = GenerateFriendlies();
111
112             if (!loaded)
113             {
114                 MessageBox.Show(
115                     Properties.Resources.MainWindow_MainWindow_FatalError_ConfigurationFile,
116                     @"Fatal Error - Configuration Failure", MessageBoxButtons.OK,
                        MessageBoxIcon.Error);
```

```
117                 CloseSafely();
118                 Close();
119             }
120         }
121 
122         /// <summary>
123         /// Publishes Message to RabbitMQ
124         /// </summary>
125         /// <param name="message">
126         /// JSON/Plain-Text Message. HAS TO BE PREFORMATTED for Json Serialization
127         /// </param>
128         /// <param name="index">
129         /// Index for Dynamic Server Allocation
130         /// </param>
131         /// <param name="uidGuid">
132         /// </param>
133         /// <returns>
134         /// Message success state
135         /// </returns>
136         private bool PublishMessage(string message, int index, Guid uidGuid)
137         {
138             try
139             {
140                 var properties = FactoryChannel[index].CreateBasicProperties();
141                 if (SerialCommunications[index].MessageType == "JSON")
142                     try
143                     {
144                         CalculateNpChart(index);
145                         JsonConvert.DeserializeObject<Messages[]>(message);
146                         properties.ContentType = "jsonObject";
147                     }
148                     catch (JsonException ex)
149                     {
150                         if (OutOfControl(index))
151                         {
152                             var sf = StackTracing.GetFrame(0);
153                             LogError(ex, LogLevel.Critical, sf);
154                             CloseSerialPortUnexpectedly(index);
155                         }
156                     }
157                 else
158                     properties.ContentType = "plain-text";
159 
160                 properties.Persistent = true;
161 
162                 var address = new PublicationAddress(ExchangeType.Direct,
163                     ServerInformation[index].ExchangeName, "");
164                 FactoryChannel[index].BasicPublish(address, properties, Encoding.UTF8.GetBytes
                      (message));
165                 FactoryChannel[index].BasicAcks += (sender, args) =>
166                 {
167                     const string sqlString = "DELETE FROM[dbo].[MessageData] WHERE [DeliveryTag] =
                          @uuid";
168 
169                     using (var conn = new SqlConnection(_connString))
170                     {
171                         try
172                         {
173                             var command = new SqlCommand(sqlString, conn) { CommandType =
                              CommandType.Text };
```

```
174                             command.Parameters.AddWithValue("@uuid", uidGuid);
175                             conn.Open();
176                             command.ExecuteNonQuery();
177                             conn.Close();
178                         }
179                         catch (Exception ex)
180                         {
181                             if (conn.State == ConnectionState.Open)
182                                 conn.Close();
183                             var sf = StackTracing.GetFrame(0);
184                             LogError(ex, LogLevel.Critical, sf);
185                         }
186                     }
187                 };
188                 return true;
189             }
190             catch (AlreadyClosedException ex)
191             {
192                 var indexOf = ex.Message.IndexOf("\"", StringComparison.Ordinal);
193                 var indexOff = ex.Message.IndexOf("\"", indexOf + 1, StringComparison.Ordinal);
194                 var errmessage = ex.Message.Substring(indexOf + 1, indexOff - indexOf - 1);
195                 MessageBox.Show(errmessage, @"Connection Already Closed", MessageBoxButtons.OK,
196                     MessageBoxIcon.Asterisk);
197 
198                 return false;
199             }
200             catch (Exception ex)
201             {
202                 //Log Message
203                 var sf = StackTracing.GetFrame(0);
204                 LogError(ex, LogLevel.Critical, sf);
205                 return false;
206             }
207         }
208 
209         private void CloseSafely()
210         {
211             TabMessageSettings.IsEnabled = false;
212             _systemTimer.Stop();
213 
214             foreach (var port in SerialPorts)
215             {
216                 while (port.IsOpen)
217                 {
218                     port.Close();
219                 }
220             }
221 
222             foreach (var model in FactoryChannel)
223             {
224                 while (model.IsOpen)
225                 {
226                     model.Close();
227                 }
228             }
229         }
230 
231         /// <summary>
232         /// Serial Commnuication Event Handler.
233         /// </summary>
```

```
234         /// <param name="sender">
235         /// COM Port Data Receveived Object
236         /// </param>
237         /// <param name="e">
238         /// Data Received
239         /// </param>
240         private void DataReceivedHandler(object sender, SerialDataReceivedEventArgs e)
241         {
242             try
243             {
244                 var sp = (SerialPort)sender;
245                 var spdata = sp.ReadLine();
246 
247                 var i = AvailableSerialPorts.TakeWhile(serialPort => serialPort.Content !=
                      sp.PortName).Count();
248                 var index = GetIndex<SerialCommunication>(Guid.Parse(AvailableSerialPorts[i].Uid));
249 
250                 SerialCommunications[index].TotalInformationReceived++;
251                 CalculateNpChart(index);
252 
253                 ProtectData(Guid.Parse(AvailableSerialPorts[index].Uid), spdata, sp.PortName);
254 
255                 while (!PublishMessage(spdata, index, Guid.Parse(AvailableSerialPorts[i].Uid)))
256                 {
257                     //Retry
258                 }
259             }
260             catch (Exception ex)
261             {
262                 //Log Message
263                 var sf = StackTracing.GetFrame(0);
264                 LogError(ex, LogLevel.Critical, sf);
265             }
266         }
267 
268         private void ProtectData(Guid uidGuid, string message, string deviceName)
269         {
270             var index = GetIndex<RabbitServerInformation>(uidGuid);
271 
272             var datInfo = new DataBaseInfo
273             {
274                 Message = message,
275                 TimeStamp = DateTime.Now,
276                 FriendlyName = FriendlyName[index],
277                 Channel = ServerInformation[index].ChannelName,
278                 Exchange = ServerInformation[index].ExchangeName,
279                 ServerAddress = ServerInformation[index].ServerAddress.ToString(),
280                 DeliveryTag = Guid.NewGuid(),
281                 DeviceType = deviceName
282             };
283 
284             const string sqlString =
285                 "INSERT [dbo].[MessageData]
                      (Message,TimeStamp,FriendlyName,Channel,Exchange,ServerAddress,DeliveryTag,DeviceTyp
                      e)VALUES"
286                 +
287 
                  "(@message,@timestamp,@friendlyname,@channel,@exchange,@serveraddress,@deliverytag,@devicetype)";
288 
289             using (var conn = new SqlConnection(_connString))
```

```
290             {
291                 var command = new SqlCommand(sqlString, conn)
292                 {
293                     CommandType = CommandType.Text
294                 };
295                 command.Parameters.AddWithValue("@message", datInfo.Message);
296                 command.Parameters.AddWithValue("@timestamp", datInfo.TimeStamp);
297                 command.Parameters.AddWithValue("@friendlyname", datInfo.FriendlyName);
298                 command.Parameters.AddWithValue("@channel", datInfo.Channel);
299                 command.Parameters.AddWithValue("@exchange", datInfo.Exchange);
300                 command.Parameters.AddWithValue("@serveraddress", datInfo.ServerAddress);
301                 command.Parameters.AddWithValue("@deliverytag", datInfo.DeliveryTag);
302                 command.Parameters.AddWithValue("@devicetype", datInfo.DeviceType);
303                 conn.Open();
304                 command.ExecuteNonQuery();
305                 conn.Close();
306             }
307         }
308 
309         /// <summary>
310         /// RabbitMQ Heartbeat Timer. Adjusts value of system information in scrollbar on tick
311         /// </summary>
312         private void InitializeHeartBeatTimer()
313         {
314             try
315             {
316                 _systemTimer.Tick += SystemTimerOnTick;
317                 _systemTimer.Interval = TimeSpan.FromMilliseconds(100);
318             }
319             catch (Exception e)
320             {
321                 var message = e.Message + "\nError in Timer Initialization";
322                 MessageBox.Show(message, e.Source, MessageBoxButtons.OK, MessageBoxIcon.Error);
323             }
324         }
325 
326         /// <summary>
327         /// Initializes SerialPort checkboxes for both Serial and ModBus communication
328         /// </summary>
329         private static void InitializeSerialPortCheckBoxes()
330         {
331             AvailableSerialPorts.Clear();
332             AvailableModbusSerialPorts.Clear();
333 
334             var ports = SerialPort.GetPortNames();
335             foreach (var t in ports)
336             {
337                 var serialPortCheck = new CheckListItem
338                 {
339                     Content = t,
340                     IsChecked = false,
341                     Name = t + "Serial",
342                     Uid = Guid.NewGuid().ToString()
343                 };
344                 AvailableSerialPorts.Add(serialPortCheck);
345 
346                 var serialModbusCheck = new CheckListItem
347                 {
348                     Content = t,
349                     IsChecked = false,
```

```
350                     Name = t + "Modbus",
351                     Uid = Guid.NewGuid().ToString()
352                 };
353                 AvailableModbusSerialPorts.Add(serialModbusCheck);
354             }
355         }
356 
357         /// <summary>
358         /// Provides initializing access to the serial ports
359         /// </summary>
360         private void InitializeSerialPorts()
361         {
362             var ports = SerialPort.GetPortNames();
363             if (ports.Length == 0)
364             {
365                 LstSerial.Items.Add("No Ports Available");
366                 LstModbusSerial.Items.Add("No Ports Available");
367             }
368             else
369             {
370                 try
371                 {
372                     LstSerial.ItemsSource = AvailableSerialPorts;
373                     LstModbusSerial.ItemsSource = AvailableModbusSerialPorts;
374                 }
375                 catch (Exception e)
376                 {
377                     var message = e.Message + "\nError in Port Enumeration";
378                     MessageBox.Show(message, e.Source, MessageBoxButtons.OK, MessageBoxIcon.Error);
379                 }
380             }
381         }
382 
383         /// <summary>
384         /// Provides the required closing processes. Allows for safe shutdown of the program
385         /// </summary>
386         /// <param name="sender">
387         /// </param>
388         /// <param name="e">
389         /// </param>
390         private void MainWindow_OnClosing(object sender, CancelEventArgs e)
391         {
392             foreach (var serialPort in SerialPorts)
393             {
394                 if (serialPort.IsOpen)
395                     serialPort.Close();
396             }
397 
398             foreach (var model in FactoryChannel)
399             {
400                 while (model != null && model.IsOpen)
401                 {
402                     model.Close();
403                 }
404             }
405         }
406 
407         private void MainWindow_OnLoaded(object sender, RoutedEventArgs e)
408         {
409             var doubleAnimation = new DoubleAnimation
```

```
410             {
411                 From = -tbmarquee.ActualWidth,
412                 To = canMain.ActualWidth,
413                 RepeatBehavior = RepeatBehavior.Forever,
414                 Duration = new Duration(TimeSpan.Parse("0:0:10"))
415             };
416             tbmarquee.BeginAnimation(Canvas.LeftProperty, doubleAnimation);
417             doubleAnimation.BeginAnimation(Canvas.LeftProperty, doubleAnimation);
418         }
419 
420         /// <summary>
421         /// TODO allow for dynamic updates of serial ports in serial port selection while keeping
422         /// current states
423         /// </summary>
424         private static void ResizeSerialSelection()
425         {
426             var ports = SerialPort.GetPortNames();
427             for (var i = 0; i < ports.Length; i++)
428             {
429                 var portName = ports[i];
430                 if (portName == AvailableSerialPorts[i].Content)
431                 {
432                 }
433             }
434         }
435 
436         /// <summary>
437         /// Clears relevant modbus related serial port and enables serial port
438         /// </summary>
439         /// <param name="sender">
440         /// </param>
441         /// <param name="e">
442         /// </param>
443         private void SerialEnabled_CheckboxChecked(object sender, RoutedEventArgs e)
444         {
445             if (!this.IsInitialized) return;
446 
447             var cb = (CheckBox)sender;
448             var uidGuid = Guid.Parse(cb.Uid);
449 
450             var index = GetIndex<CheckListItem>(uidGuid);
451 
452             var cbo = (CheckListItem)LstModbusSerial.Items[index];
453             if (cb.IsChecked != null) cbo.IsChecked = false;
454 
455             //disables Modbus COM Port
456             AvailableModbusSerialPorts.RemoveAt(index);
457             AvailableModbusSerialPorts.Insert(index, cbo);
458 
459             //Enable Port for serial communications
460             SetupSerial(uidGuid);
461             ResizeLineSeries(AvailableSerialPorts[index].Name);
462 
463             var setupSerialForm = new SerialPortSetup(uidGuid);
464             var activate = setupSerialForm.ShowDialog(); //Confirm Settings
465 
466             switch (activate)
467             {
468                 case true:
469 
```

```
470                     var friendlyNameForm = new VariableConfigure(uidGuid);
471                     var nameSet = friendlyNameForm.ShowDialog();
472                     if ((nameSet != null) && !nameSet.Value)
473                         goto case null;
474 
475                     SetupFactory(uidGuid);
476                     ServerInformation[ServerInformation.Length - 1] = SetDefaultSettings(uidGuid);
477 
478                     var configureServer = new SetupServer(uidGuid);
479                     var serverConfigured = configureServer.ShowDialog();
480 
481                     if ((serverConfigured != null) && !serverConfigured.Value)
482                     {
483                         //Canceled
484                         Array.Resize(ref ServerInformation, ServerInformation.Length - 1);
485                         goto case null;
486                     }
487 
488                     SerialPorts[SerialPorts.Length - 1].DataReceived += DataReceivedHandler;
489                     var init = SerialPortInitialize(SerialPorts.Length - 1, this.IsInitialized);
490                     tbmarquee.Text += $"Serial Port {cb.Name} Active";
491                     if (init) return;
492 
493                     //Initialzation of port failed. Closing port and unchecking it
494                     cb.IsChecked = false;
495                     AvailableSerialPorts.RemoveAt(index);
496                     var cliT = new CheckListItem
497                     {
498                         Name = cb.Name,
499                         Uid = cb.Uid,
500                         Content = cbo.Content,
501                         IsChecked = false
502                     };
503                     AvailableSerialPorts.Insert(index, cliT);
504                     tbmarquee.Text.Replace($"Serial Port {cb.Name} Active", "");
505                     break;
506 
507                 case null:
508                 default: //incl case false
509                     cb.IsChecked = false;
510                     AvailableSerialPorts.RemoveAt(index);
511                     var cliF = new CheckListItem
512                     {
513                         Name = cb.Name,
514                         Uid = cb.Uid,
515                         Content = cbo.Content,
516                         IsChecked = false
517                     };
518                     AvailableSerialPorts.Insert(index, cliF);
519                     Array.Resize(ref SerialPorts, SerialPorts.Length - 1); //Removes last
                        initialization of Serial Port
520                     break;
521             }
522         }
523 
524         private void SerialEnabled_CheckboxUnchecked(object sender, RoutedEventArgs e)
525         {
526             if (!this.IsInitialized) return;
527 
528             var cb = (CheckBox)sender;
```

```
529             var port = SerialPorts.FirstOrDefault(sp => sp.PortName == cb.Content.ToString());
530
531             if ((port != null) && port.IsOpen)
532                 port.Close();
533         }
534
535         /// <summary>
536         /// Clears related Serial Port that is not
537         /// </summary>
538         /// <param name="sender">
539         /// </param>
540         /// <param name="e">
541         /// </param>
542         private void SerialModbusEnabled_CheckboxChecked(object sender, RoutedEventArgs e)
543         {
544             if (!this.IsInitialized) return;
545
546             var cb = (CheckBox)sender;
547             var uidGuid = Guid.Parse(cb.Uid);
548
549             var index = GetIndex<CheckListItem>(uidGuid);
550
551             cb.Name = AvailableModbusSerialPorts[index].Name;
552
553             var cbo = (CheckListItem)LstSerial.Items[index];
554             if (cb.IsChecked != null) cbo.IsChecked = false;
555
556             //disables Modbus COM Port
557             AvailableSerialPorts.RemoveAt(index);
558             AvailableSerialPorts.Insert(index, cbo);
559
560             switch (true)
561             {
562                 case true:
563                     //Enable Port
564                     SetupModbusSerial(uidGuid);
565
566                     var setupSerialForm = new SerialPortSetup(uidGuid);
567                     setupSerialForm.cboMessageType.SelectedIndex = 1;
568                     var activate = setupSerialForm.ShowDialog();
569
570                     if ((activate == null) || !activate.Value)
571                     {
572                         CloseModbusSerial(uidGuid);
573                         RemoveAtIndex<SerialCommunication>(SerialCommunications.Length - 1,
                              SerialCommunications);
574                         ResetCheckBox(AvailableModbusSerialPorts[index]);
575                         return;
576                     }
577
578                     index = SerialCommunications.Length - 1;
579                     SerialCommunications[index].X(SerialCommunications[index].MaximumErrors);
580
581                     var port = new SerialPort
582                     {
583                         PortName = SerialCommunications[index].ComPort,
584                         BaudRate = (int)SerialCommunications[index].BaudRate,
585                         Parity = SerialCommunications[index].SerialParity,
586                         StopBits = SerialCommunications[index].SerialStopBits,
587                         DataBits = SerialCommunications[index].SerialBits,
```

```
588                         Handshake = SerialCommunications[index].FlowControl,
589                         RtsEnable = SerialCommunications[index].RtsEnable,
590                         ReadTimeout = SerialCommunications[index].ReadTimeout
591                     };
592
593                     Array.Resize(ref ModbusControls, ModbusControls.Length + 1);
594                     ModbusControls[ModbusControls.Length - 1].UidGuid = uidGuid;
595
596                     ModbusControls[ModbusControls.Length - 1].ModbusTimers = new DispatcherTimer
597                     {
598                         Interval = TimeSpan.FromMilliseconds(1000),
599                         IsEnabled = false
600                     };
601                     ModbusControls[ModbusControls.Length - 1].ModbusTimers.Tick += ModbusTimerOnTick;
602                     _modbusTimerId.Add(ModbusControls[ModbusControls.Length - 1].ModbusTimers,
                          uidGuid);
603
604                     ModbusControls[ModbusControls.Length - 1].ModbusAddressList =
605                         new List<Tuple<bool, bool, bool, bool, int>>();
606                     index = GetIndex<CheckListItem>(uidGuid);
607
608                     var modbusSelection = new ModbusSelection
609                     {
610                         DeviceAddress = port.PortName,
611                         DeviceName = AvailableModbusSerialPorts[index].Name,
612                         IsAbsolute = true
613                     };
614
615                     var addressesInitialized = modbusSelection.ShowDialog();
616
617                     if ((addressesInitialized == null) || !addressesInitialized.Value)
618                     {
619                         MessageBox.Show(@"Failed to create Modbus Configuration", @"ERROR",
                              MessageBoxButtons.OK,
620                                 MessageBoxIcon.Error);
621                         goto case false;
622                     }
623
624                     SetupFactory(uidGuid);
625
626                     var configureServer = new SetupServer(uidGuid);
627                     var serverConfigured = configureServer.ShowDialog();
628
629                     if ((serverConfigured != null) && !serverConfigured.Value)
630                     {
631                         Array.Resize(ref ServerInformation, ServerInformation.Length - 1);
632                         Array.Resize(ref ModbusControls, ModbusControls.Length - 1);
633                         MessageBox.Show(@"Failed to open RabbitMQ Connection", @"ERROR",
                              MessageBoxButtons.OK,
634                                 MessageBoxIcon.Error);
635                         break;
636                     }
637                     InitializeModbusClient(port);
638                     tbmarquee.Text += $"Serial Port {cb.Name} Active";
639
640                     if (ModbusClients[ModbusClients.Length - 1].Connected)
641                     {
642                         ResizeLineSeries(cb.Name);
643                         ModbusControls[ModbusControls.Length - 1].ModbusTimers.IsEnabled = true;
644                         InitializeRead(uidGuid);
```

```
645                         break;
646                     }
647                     MessageBox.Show(@"Failed to open Serial Port", @"ERROR", MessageBoxButtons.OK,
648                         MessageBoxIcon.Error);
649                     goto case false; //unsuccessful
650 
651                 case false:
652                     cb.IsChecked = false;
653                     AvailableModbusSerialPorts.RemoveAt(index);
654                     RemoveAtIndex<RabbitServerInformation>(ServerInformation.Length - 1,
                          ServerInformation);
655 
656                     var cliT = new CheckListItem
657                     {
658                         Name = cb.Name,
659                         Uid = cb.Uid,
660                         Content = cbo.Content,
661                         IsChecked = false
662                     };
663                     AvailableModbusSerialPorts.Insert(index, cliT);
664                     tbmarquee.Text.Replace($"Serial Port {cb.Name} Active", "");
665                     break;
666             }
667         }
668 
669         private void InitializeRead(Guid uidGuid)
670         {
671             var index = ModbusControls.Select((val, ind) =>
672                     new { ind, val })
673                 .First(e => e.val.UidGuid == uidGuid)
674                 .ind;
675             var modbusItem = ModbusControls.FirstOrDefault(e => e.UidGuid == uidGuid);
676             var message = "";
677 
678             foreach (var modbusAddress in modbusItem.ModbusAddressList)
679             {
680                 var address = modbusAddress.Item5;
681 
682                 if (modbusAddress.Item1)
683                     try
684                     {
685                         SerialCommunications[index].TotalInformationReceived++;
686                         CalculateNpChart(index);
687                         while (true)
688                         {
689                             try
690                             {
691                                 var readCoil = ModbusClients[index].ReadCoils(address, 1);
692                                 message =
693                                     readCoil.Aggregate(message, (current, b) => current + b.ToString()
                                  + "\n");
694                             }
695                             catch (IOException)
696                             {
697                                 Thread.Sleep(100);
698 
699                                 continue;
700                             }
701                             break;
702                         }
```

```
703
704                         ProtectData(uidGuid, message, ModbusClients[index].IPAddress + "1" +
                              modbusAddress.Item5);
705
706                         PublishMessage(message, index, uidGuid);
707
708                         UpdateGraph(uidGuid,
709                             AvailableModbusSerialPorts[
710                                     AvailableModbusSerialPorts.Select((val, ind) => new { ind, val })
711                                         .First(e => e.val.Uid == uidGuid.ToString())
712                                         .ind]
713                                 .Name);
714                     }
715                     catch (CRCCheckFailedException crcCheckFailedException)
716                     {
717                         if (OutOfControl(index))
718                         {
719                             var sf = StackTracing.GetFrame(0);
720                             LogError(crcCheckFailedException, LogLevel.Critical, sf);
721                             MessageBox.Show("CRC Failure. Please check settings and connection",
722                                 "CRC Error Check Failure", MessageBoxButtons.OK,
                                  MessageBoxIcon.Error);
723                             CloseModbusUnexpectedly(uidGuid);
724                             ResetCheckBox(AvailableModbusSerialPorts[index]);
725                             return;
726                         }
727                     }
728
729                 if (modbusAddress.Item2)
730                     try
731                     {
732                         SerialCommunications[index].TotalInformationReceived++;
733                         CalculateNpChart(index);
734
735                         while (true)
736                         {
737                             try
738                             {
739                                 var readDiscrete = ModbusClients[index].ReadDiscreteInputs(address,
                                  2);
740                                 message =
741                                     readDiscrete.Aggregate(message,
742                                         (current, b) => current + b.ToString() + "\n");
743                             }
744                             catch (IOException)
745                             {
746                                 Thread.Sleep(100);
747
748                                 continue;
749                             }
750                             break;
751                         }
752
753                         ProtectData(uidGuid, message, ModbusClients[index].IPAddress + "1" +
                              modbusAddress.Item5);
754
755                         PublishMessage(message, index, uidGuid);
756
757                         UpdateGraph(uidGuid,
758                             AvailableModbusSerialPorts[
```

```
                                    AvailableModbusSerialPorts.Select((val, ind) => new { ind, val })
                                        .First(e => e.val.Uid == uidGuid.ToString())
                                        .ind]
                                .Name);
                    }
                    catch (CRCCheckFailedException crcCheckFailedException)
                    {
                        if (OutOfControl(index))
                        {
                            var sf = StackTracing.GetFrame(0);
                            LogError(crcCheckFailedException, LogLevel.Critical, sf);
                            MessageBox.Show("CRC Failure. Please check settings and connection",
                                "CRC Error Check Failure", MessageBoxButtons.OK,
                             MessageBoxIcon.Error);
                            CloseModbusUnexpectedly(uidGuid);
                            ResetCheckBox(AvailableModbusSerialPorts[index]);
                            return;
                        }
                    }

                if (modbusAddress.Item3)
                    try
                    {
                        SerialCommunications[index].TotalInformationReceived++;
                        CalculateNpChart(index);

                        while (true)
                        {
                            try
                            {
                                var readRegister = ModbusClients[index].ReadHoldingRegisters(address,
                             3);
                                message =
                                    readRegister.Aggregate(message,
                                        (current, i) => current + i.ToString() + "\n");
                            }
                            catch (IOException)
                            {
                                Thread.Sleep(100);

                                continue;
                            }
                            break;
                        }

                        ProtectData(uidGuid, message, ModbusClients[index].IPAddress + "1" +
                          modbusAddress.Item5);

                        PublishMessage(message, index, uidGuid);

                        UpdateGraph(uidGuid,
                            AvailableModbusSerialPorts[
                                    AvailableModbusSerialPorts.Select((val, ind) => new { ind, val })
                                        .First(e => e.val.Uid == uidGuid.ToString())
                                        .ind]
                                .Name);
                    }
                    catch (CRCCheckFailedException crcCheckFailedException)
                    {
                        if (OutOfControl(index))
```

```
816                         {
817                             var sf = StackTracing.GetFrame(0);
818                             LogError(crcCheckFailedException, LogLevel.Critical, sf);
819                             MessageBox.Show("CRC Failure. Please check settings and connection",
820                                 "CRC Error Check Failure", MessageBoxButtons.OK,
                                MessageBoxIcon.Error);
821                             CloseModbusUnexpectedly(uidGuid);
822                             ResetCheckBox(AvailableModbusSerialPorts[index]);
823                             return;
824                         }
825                     }
826 
827                 if (modbusAddress.Item4)
828                     try
829                     {
830                         SerialCommunications[index].TotalInformationReceived++;
831                         CalculateNpChart(index);
832 
833                         while (true)
834                         {
835                             try
836                             {
837                                 var readInputRegister = ModbusClients[index].ReadInputRegisters
                                (address, 4);
838                                     message =
839                                         readInputRegister.Aggregate(message,
840                                             (current, i) => current + i.ToString() + "\n");
841                             }
842                             catch (IOException)
843                             {
844                                 Thread.Sleep(100); //attempt to resyncronize
845 
846                                 continue;
847                             }
848                             break;
849                         }
850 
851                         ProtectData(uidGuid, message, ModbusClients[index].IPAddress + "1" +
                            modbusAddress.Item5);
852 
853                         PublishMessage(message, index, uidGuid);
854 
855                         UpdateGraph(uidGuid,
856                             AvailableModbusSerialPorts[
857                                     AvailableModbusSerialPorts.Select((val, ind) => new { ind, val })
858                                         .First(e => e.val.Uid == uidGuid.ToString())
859                                         .ind]
860                                 .Name);
861                     }
862                     catch (CRCCheckFailedException crcCheckFailedException)
863                     {
864                         if (OutOfControl(index))
865                         {
866                             var sf = StackTracing.GetFrame(0);
867                             LogError(crcCheckFailedException, LogLevel.Critical, sf);
868                             MessageBox.Show("CRC Failure. Please check settings and connection",
869                                 "CRC Error Check Failure", MessageBoxButtons.OK,
                                MessageBoxIcon.Error);
870                             CloseModbusUnexpectedly(uidGuid);
871                             ResetCheckBox(AvailableModbusSerialPorts[index]);
```

```
872                             return;
873                         }
874                     }
875             }
876         }
877
878         private void ModbusTimerOnTick(object sender, EventArgs eventArgs)
879         {
880             var uidGuid = Guid.Empty;
881             _modbusTimerId.TryGetValue((DispatcherTimer)sender, out uidGuid);
882
883             if (uidGuid == Guid.Empty) return;
884
885             InitializeRead(uidGuid);
886         }
887
888         private void ResetCheckBox(CheckListItem checkListItem)
889         {
890             var index = GetIndex<CheckListItem>(Guid.Parse(checkListItem.Uid));
891
892             this.Dispatcher.Invoke((MethodInvoker)delegate
893            {
894                 var checkList = new CheckListItem
895                 {
896                     Name = checkListItem.Name,
897                     Uid = checkListItem.Uid,
898                     Content = checkListItem.Content,
899                     IsChecked = false
900                 };
901
902                 if (AvailableSerialPorts.Any(availableSerialPort => checkListItem.Uid ==
                      availableSerialPort.Uid))
903                 {
904                     AvailableSerialPorts.Remove(AvailableSerialPorts[index]);
905                     AvailableSerialPorts.Insert(index, checkList);
906                 }
907
908                 if (AvailableModbusSerialPorts.Any(
909                     availableModbusSerialPort => checkListItem.Uid == availableModbusSerialPort.Uid))
910                 {
911                     AvailableModbusSerialPorts.Remove(AvailableModbusSerialPorts[index]);
912                     AvailableModbusSerialPorts.Insert(index, checkList);
913                 }
914            });
915         }
916
917         /// <summary>
918         /// Updates infomration on Statusbar on what system is exeperiencing.
919         /// </summary>
920         /// <param name="sender">
921         /// System Timer Thread Object
922         /// </param>
923         /// <param name="eventArgs">
924         /// Timer Arguments
925         /// </param>
926         private void SystemTimerOnTick(object sender, EventArgs eventArgs)
927         {
928             //Prevents code from running before intialization
929             if (!this.IsInitialized) return;
930
```

```
931             ResizeSerialSelection();
932             for (var i = 0; i < AvailableSerialPorts.Count; i++)
933             {
934                 if (AvailableSerialPorts[i].IsChecked)
935                     UpdateGraph(Guid.Parse(AvailableSerialPorts[i].Uid), AvailableSerialPorts
                        [i].Name);
936                 if (AvailableModbusSerialPorts[i].IsChecked)
937                     UpdateGraph(Guid.Parse(AvailableSerialPorts[i].Uid), AvailableModbusSerialPorts
                        [i].Name);
938             }
939 
940             for (var i = 0; i < MessagesPerSecond.Length; i++)
941             {
942                 MessagesPerSecond[i] = 0.0;
943             }
944         }
945 
946         private void UpdateGraph(Guid uidGuid, string itemName)
947         {
948             var timeElapsed = DateTime.Now - _previousTime;
949 
950             if (timeElapsed < TimeSpan.FromSeconds(1))
951                 return; //Only update 1ce per second
952 
953             var index = GetIndex<MessageDataHistory>(uidGuid);
954             if (index == -1) return;
955 
956             this.Dispatcher.Invoke((MethodInvoker)delegate
957            {
958                 if (MessagesSentDataPair[index].Count > 60)
959                     MessagesSentDataPair[index].RemoveAt(0);
960                 var timeNow = DateTime.Now.Minute + ":" + DateTime.Now.Second;
961                 MessagesPerSecond[index]++;
962 
963                 if ((Lineseries[index] != null) && ((string)Lineseries[index].Title == itemName))
964                 {
965                     var messageDataHistory = new MessageDataHistory
966                     {
967                         KeyPair = new KeyValuePair<string, double>(timeNow,
968                             MessagesPerSecond[index] / timeElapsed.TotalSeconds),
969                         UidGuid = uidGuid
970                     };
971                     MessagesSentDataPair[index].Add(messageDataHistory);
972                 }
973                 else
974                 {
975                     Lineseries[index] = new LineSeries
976                     {
977                         ItemsSource = MessagesSentDataPair[index],
978                         DependentValuePath = "Value",
979                         IndependentValuePath = "Key",
980                         Title = itemName
981                     };
982                     LineChart.Series.Add(Lineseries[index]);
983                     var messageDataHistory = new MessageDataHistory
984                     {
985                         KeyPair = new KeyValuePair<string, double>(timeNow,
986                             MessagesPerSecond[index] / timeElapsed.TotalSeconds),
987                         UidGuid = uidGuid
988                     };
```

```
989                     MessagesSentDataPair[index].Add(messageDataHistory);
990                 }
991             });
992         }
993 
994         ///TODO add IP address Management
995         private void AddModbusTCP_Click(object sender, RoutedEventArgs e)
996         {
997             //
998         }
999 
1000        protected internal struct CheckListItem
1001        {
1002            public string Content { get; set; }
1003            public bool IsChecked { get; set; }
1004            public string Name { get; set; }
1005            public string Uid { get; set; }
1006        }
1007 
1008        private struct DataBaseInfo
1009        {
1010            internal string Message { get; set; }
1011            internal DateTime TimeStamp { get; set; }
1012            internal string FriendlyName { get; set; }
1013            internal string Channel { get; set; }
1014            internal string Exchange { get; set; }
1015            internal string ServerAddress { get; set; }
1016            internal Guid DeliveryTag { get; set; }
1017            internal string DeviceType { get; set; }
1018        }
1019 
1020        protected internal struct MessageDataHistory
1021        {
1022            internal KeyValuePair<string, double> KeyPair { get; set; }
1023            internal Guid UidGuid { get; set; }
1024        }
1025 
1026        protected internal struct ModbusControl
1027        {
1028            /// <summary>
1029            /// <para>
1030            /// FunctionCode
1031            /// </para>
1032            /// <para>
1033            /// Address
1034            /// </para>
1035            /// </summary>
1036            public List<Tuple<bool, bool, bool, bool, int>> ModbusAddressList { get; set; }
1037 
1038            internal static ObservableCollection<MessageDataHistory> MessagesSentDataPair { get;
                  set; }
1039 
1040            internal DispatcherTimer ModbusTimers { get; set; }
1041 
1042            internal Guid UidGuid { get; set; }
1043        }
1044    }
1045 }
```