```
using MySql.Data.MySqlClient;
using PlotterDataGH.Properties;
using System.Data;
using System.Diagnostics;
using System.IO;
using System.Reflection;
using System.Windows;
using System.Windows.Controls;
using Newtonsoft.Json;
using PlotterDataGH;
using System.Threading;
using System.Net;
using System.Collections.Generic;
using Microsoft.Data.Sqlite;
using System.Text;
using System.Net.NetworkInformation;
using System.Windows.Media;
using System.Threading.Tasks;
using System;
using Microsoft.Win32.TaskScheduler;
using System.Net.Http;

namespace WpfApp2
{
    /// <summary>
    /// Interaction logic for UserControl1.xaml
    /// </summary>
    ///
    public partial class UserControl1 : UserControl
    {
        bool cartridgeHidden = true;
        public int plotterId;
        public string plotterIp;
        public string serialnm;
        AutoResetEvent waiter = new AutoResetEvent(false);

        private static readonly HttpClient client = new HttpClient();

        public MainWindow ParentForm { get; set; }

        public UserControl1()
        {
            InitializeComponent();
            btnScan.IsEnabled = false;
        }

        private void SetTimer()
        {
            //Set a timer between pings
            System.Windows.Threading.DispatcherTimer dispatcherTimer = new 
              System.Windows.Threading.DispatcherTimer();
            dispatcherTimer.Tick += dispatcherTimer_Tick;
            dispatcherTimer.Interval = new TimeSpan(0, 10, 0);
            dispatcherTimer.Start();
        }

        public void loadData()
        {
            //Load cartridge data and put it in a class called cartridgeControl
            DataTable dataTable = new DataTable();
            SqliteConnection cnn;
            SqliteCommand cmd = null;
            cnn = new SqliteConnection("Data Source=plotterData.db;");
            cnn.Open();

            string query = string.Format("SELECT * FROM `cartridge_reading` where `parent_id` = {0}", plotterId);
            cmd = new SqliteCommand(query, cnn);

            SqliteDataReader reader = cmd.ExecuteReader();
            dataTable.Load(reader);

            foreach (DataRow row in dataTable.Rows)
            {
                RowDefinition rd = new RowDefinition();
                rd.Height = GridLength.Auto;
                plotterControl.RowDefinitions.Add(rd);
                cartridgeControl cartridgeControls = new cartridgeControl();
                cartridgeControls.lblCartridgeNaam.Content = row["cartridge_model"].ToString();
                cartridgeControls.lblCatridgeVolume.Content = row["volume"].ToString();
                plotterControl.Children.Add(cartridgeControls);
                Grid.SetRow(cartridgeControls, plotterControl.RowDefinitions.Count);
            }
            RowDefinition last = new RowDefinition();
            plotterControl.RowDefinitions.Add(last);

            for (int x = plotterControl.RowDefinitions.Count - 1; x > 0; x--)
            {
                plotterControl.RowDefinitions[x].Height = new GridLength(0);
            }

            SetTimer();
            SendPing();
        }

        #region Check status
        private void dispatcherTimer_Tick(object sender, EventArgs e)
        {
            //Send a ping to the plotter and if it responds turn the light green
            SendPing();
        }

        private void SendPing()
        {
            Ping pingSender = new Ping();

            // When the PingCompleted event is raised,
            // the PingCompletedCallback method is called.
            pingSender.PingCompleted += new PingCompletedEventHandler
              (PingCompletedCallback);

            // Create a buffer of 32 bytes of data to be transmitted.
            string data = "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa";
            byte[] buffer = Encoding.ASCII.GetBytes(data);

            // Wait 12 seconds for a reply.
            int timeout = 12000;

            // Set options for transmission:
            // The data can go through 64 gateways or routers
            // before it is destroyed, and the data packet
            // cannot be fragmented.
            PingOptions options = new PingOptions(64, true);

            pingSender.SendAsync(plotterIp, timeout, buffer, options, waiter);
        }

        private void PingCompletedCallback(object sender,
          PingCompletedEventArgs e)
        {
            // If the operation was canceled, display a message to the user.
            if (e.Cancelled)
            {
                ellStatus.Fill = new SolidColorBrush(Colors.Red);
                btnScan.IsEnabled = false;

                // Let the main thread resume.
                // UserToken is the AutoResetEvent object that the main thread
                // is waiting for.
                ((AutoResetEvent)e.UserState).Set();
            }

            // If an error occurred, display the exception to the user.
            if (e.Error != null)
            {
                ellStatus.Fill = new SolidColorBrush(Colors.Red);
                btnScan.IsEnabled = false;

                // Let the main thread resume.
                ((AutoResetEvent)e.UserState).Set();
            }

            PingReply reply = e.Reply;

            DisplayReply(reply);

            // Let the main thread resume.
            ((AutoResetEvent)e.UserState).Set();
        }

        public void DisplayReply(PingReply reply)
        {
            if (reply == null)
            {
                return;
            }


            if (reply.Status == IPStatus.Success)
            {
                ellStatus.Fill = new SolidColorBrush(Colors.Green);
                btnScan.IsEnabled = true;
            }
            else
            {
                ellStatus.Fill = new SolidColorBrush(Colors.Red);
                btnScan.IsEnabled = false;
            }
        }

        #endregion

        private void btnExpand_Click(object sender, RoutedEventArgs e)
        {
            //Expand the control which will show the cartridges
            if(cartridgeHidden)
            {
                for(int x = 0; x <= plotterControl.RowDefinitions.Count - 1; x++)
                {
                    plotterControl.RowDefinitions[x].Height = GridLength.Auto;
                }

                cartridgeHidden = false;

                Button button = sender as Button;
                MahApps.Metro.IconPacks.PackIconFontAwesome fe = button.Content as MahApps.Metro.IconPacks.PackIconFontAwesome;
                fe.Kind = MahApps.Metro.IconPacks.PackIconFontAwesomeKind.AngleDoubleUpSolid;
            }
            else
            {
                for(int x = plotterControl.RowDefinitions.Count - 1; x > 0; x--)
                {
                    plotterControl.RowDefinitions[x].Height = new GridLength(0);
                }
                cartridgeHidden = true;

                Button button = sender as Button;
                MahApps.Metro.IconPacks.PackIconFontAwesome fe = button.Content as MahApps.Metro.IconPacks.PackIconFontAwesome;
                fe.Kind = MahApps.Metro.IconPacks.PackIconFontAwesomeKind.AngleDoubleD
```

```
                      ownSolid;
                }
            }

            private void Button_Click(object sender, RoutedEventArgs e)
            {
                AddPlotter addPlotter = new AddPlotter();
                addPlotter.ParentForm = ParentForm;
                addPlotter.editForm(plotterId);
                addPlotter.editingMode = true;
                addPlotter.Show();
            }

            #region Scanning

            private void Button_Click_1(object sender, RoutedEventArgs e)
            {
                //Run a scan of this specific plotter
                RunScan(lblMerk.Uid.ToString(), plotterIp, 
                  lblNaam.Content.ToString());
                MahApps.Metro.IconPacks.PackIconFontAwesome fe = btnScan.Content 
                  as MahApps.Metro.IconPacks.PackIconFontAwesome;
                fe.Kind = 
                  MahApps.Metro.IconPacks.PackIconFontAwesomeKind.SyncSolid;
                btnScan.IsEnabled = false;
            }

            public void RunScan(string Merk, string IP, string Naam)
            {
                var debugField = Path.GetDirectoryName(
        Assembly.GetExecutingAssembly().GetName().CodeBase);

                debugField = debugField.Substring(6);

                var filename = debugField + @"/ghWebscraper.exe";

                //Start the Converted python file and pass the paramater
                string arguments = string.Format(@"{0} {1} {2} {3}", Merk, IP, 
                  Settings.Default.sendData, Naam);

                ellStatus.Fill = new SolidColorBrush(Colors.Orange);

                //Process myProcess = new Process();
                //myProcess.Exited += new EventHandler(myProcess_Exited);
                //myProcess.StartInfo.FileName = filename;
                //myProcess.StartInfo.Arguments = arguments;
                //myProcess.Start();

                doStuff(filename, arguments);
            }

            async System.Threading.Tasks.Task doStuff(string fileName, string 
              args)
            {
                ParentForm.DisableWhileScanning();
                await RunProcessAsync(fileName, args);

                //MahApps.Metro.IconPacks.PackIconFontAwesome fe = btnScan.Content as MahApps.Metro.IconPacks.PackIconFontAwesome;
                //fe.Kind = MahApps.Metro.IconPacks.PackIconFontAwesomeKind.BinocularsSolid;
                //btnScan.IsEnabled = true;

                ParentForm.fillerGrid.RowDefinitions.Clear();
                ParentForm.fillerGrid.Children.Clear();
                ParentForm.LoadData();
            }

            public static async Task<int> RunProcessAsync(string fileName, string args)
            {
                using (var process = new Process
                {
                    StartInfo =
            {
                FileName = fileName, Arguments = args,
                UseShellExecute = false, CreateNoWindow = true,
                RedirectStandardOutput = true, RedirectStandardError = true
            },
                    EnableRaisingEvents = true
                })
                {
                    return await RunProcessAsync(process).ConfigureAwait(false);
                }
            }
            private static Task<int> RunProcessAsync(Process process)
            {
                var tcs = new TaskCompletionSource<int>();

                process.Exited += (s, ea) => tcs.SetResult(process.ExitCode);
                process.OutputDataReceived += (s, ea) => Console.WriteLine(ea.Data);
                process.ErrorDataReceived += (s, ea) => Console.WriteLine("ERR: " + ea.Data);

                bool started = process.Start();
                if (!started)
                {
                    //you may allow for the process to be re-used (started = false)
                    //but I'm not sure about the guarantees of the Exited event in such a case
                    throw new InvalidOperationException("Could not start process: " + process);
                }

                process.BeginOutputReadLine();
                process.BeginErrorReadLine();

                return tcs.Task;

            }

            #endregion

            #region Send Data
            private void btnSend_Click(object sender, RoutedEventArgs e)
            {
                //Send data to Goedhart Group
                Await();
            }

            async System.Threading.Tasks.Task Await()
            {
                var task = SendPlotterAsync();
                int timeout = 1000;
                if (await System.Threading.Tasks.Task.WhenAny(task, 
                  System.Threading.Tasks.Task.Delay(timeout)) == task)
                {
                    // task completed within timeout
                }
                else
                {
                    MessageBox.Show("Verbinding met de Goedhart Servers kon niet 
                      gemaakt worden");
                }
            }

            async System.Threading.Tasks.Task SendPlotterAsync()
            {
                var values = new Dictionary<string, string>
                {
                    { "postType", "Plotter" },
                    { "serial_number", serialnm },
                    { "model_id", lblMerk.Uid.ToString() },
                    { "meters_printed", lblMeterstand.Content.ToString() },
                    { "naam", lblNaam.Content.ToString() },
                    { "IP", plotterIp },
                    { "bedrijfs_Naam", Settings.Default.bedrijfsNaam }
                };

                var content = new FormUrlEncodedContent(values);

                var response = await client.PostAsync("http://10.0.0.125/", 
                  content);

                var responseString = await response.Content.ReadAsStringAsync();

                DataTable dataTable = new DataTable();
                SqliteConnection cnn;
                SqliteCommand cmd = null;
                cnn = new SqliteConnection("Data Source=plotterData.db;");
                cnn.Open();

                string query = string.Format("SELECT * FROM `cartridge_reading` 
                  where `parent_id` = {0}", plotterId);
                cmd = new SqliteCommand(query, cnn);

                SqliteDataReader reader = cmd.ExecuteReader();
                dataTable.Load(reader);

                foreach (DataRow row in dataTable.Rows)
                {
                    SendCartridgeAsync(responseString, row["cartridge_model"].ToString(), row["volume"].ToString(), row["max_volume"].ToString());
                }
            }

            async System.Threading.Tasks.Task SendCartridgeAsync(string parent_id, string cartridge_model, string volume, string max_volume = null)
            {
                var values = new Dictionary<string, string>
                {
                    { "postType", "Cartridge" },
                    { "parent_id", parent_id },
                    { "cartridge_model", cartridge_model },
                    { "volume", volume },
                    { "max_volume", max_volume }
                };

                var content = new FormUrlEncodedContent(values);

                var response = await client.PostAsync("http://10.0.0.125/", content);

                var responseString = await response.Content.ReadAsStringAsync();
            }

            #endregion
        }
    }

```