Polisy Data Collection

Contents

Sometimes it is necessary to record data points in your home controller system in order to determine the best way to resolve something. For example, you might want to track your air conditioner usage over time to see when it was being used most so that you change when it turns on in order to save electricity.

In this document I will describe the steps that I took to accomplish this goal. Although I use specific thermostat references, it should be clear that this approach can easily be adjusted to any sort of data collection.

I use the TimeData and Climacell (weather) nodes in this example as well as a Trane Z-Wave thermostat. I also use Thunderbird as an email client – if you use another client, your data may vary slightly.

Note in this example, I describe the cooling data however there are several other data points that are also being collected.

Recording the Data

A simple routine to record how long the A/C ran can be seen below. This accumulates the total run time in the variable $CoolMinutesTotalMain.

Program Content for 'Track Main Cooling'

```
If
        'Thermostat Main Area' Heat/Cool State is Cooling

Then
        $CoolStartTimeMain  = 'TimeData' Minutes YTD

Else
        $CoolMinutesMain  = 'TimeData' Minutes YTD
        $CoolMinutesMain -= $CoolStartTimeMain
        $CoolMinutesTotalMain += $CoolMinutesMain
        $CoolStartTimeMain  = 'TimeData' Minutes YTD


idle -> cooling -> pending cooling -> idle
```

Reporting the Data

I was not able to find a way to store arrays of values in Polisy. The way I chose to accomplish this was using Polisy's custom email notifications. By recording the data and then sending it to myself via email once per hour I am able to collect 24 data points every day. The logic to do so is shown below.

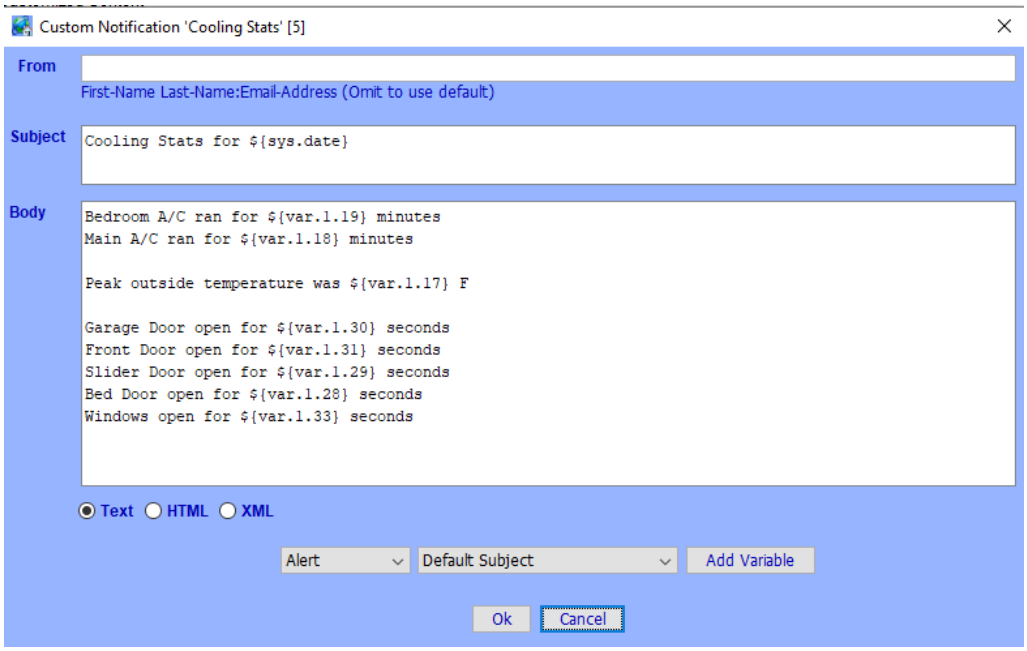
```
Program Content for 'Report Cooling Stats'
If
        'TimeData' Minute is 0 minute

Then
        $PeakOutsideTemperature  = 'Climacell Weather' Temperature °F
        $ACBedMinutes  = $CoolMinutesTotalBeds
        $ACMainMinutes  = $CoolMinutesTotalMain
        $ACBedSeconds  = $DoorBedOpenSeconds
        $ACFrontSeconds  = $DoorFrontOpenSeconds
        $ACGarageSeconds  = $DoorGarageOpenSeconds
        $ACSliderSeconds  = $DoorSliderOpenSeconds
        $ACWindowsSeconds  = $WindowsOpenSeconds
        $CoolMinutesTotalBeds  = 0
        $CoolMinutesTotalMain  = 0
        $DoorBedOpenSeconds  = 0
        $DoorFrontOpenSeconds  = 0
        $DoorGarageOpenSeconds  = 0
        $DoorSliderOpenSeconds  = 0
        $WindowsOpenSeconds  = 0
        Send Notification to 'Alec and Marilyn' content 'Cooling Stats'

Else
   - No Actions - (To add one, press 'Action')
```

Note that Polisy does not protect the variables used in Notifications so you have to copy the data to a temporary variable if it needs to be zeroed out as above. In this case the variables that start with "$AC…" are the temporary variables used by the notification, whereas the others are actual data collection points.

The 'Cooling Stats' notification is a simple email that sends the cooling data points (stored in the temporary variables) to my email account.



Getting and Storing the Email Data

In my email client I have created a filter that picks out the emails based upon seeing the words “Cooling Stats” in the subject line and places them in a specific folder.

I then select all of the Cooling Stats emails in the folder and use my email clients “Save As” function to save them as individual files on my hard drive. The data from these emails includes lots of header information that is not needed. Shown below is an example email file.

```
X-Mozilla-Status: 0001
X-Mozilla-Status2: 00000000
X-Mozilla-Keys:
Delivered-To: **********@gmail.com
Received: by 2002:a05:6520:4d01:b0:1be:f098:e203 with SMTP id cy1csp2410727lkb;
      Mon, 16 May 2022 10:00:30 -0700 (PDT)

   **** LOTS OF HEADER DATA REMOVED FOR CLARITY ****

Date: Mon, 16 May 2022 10:00:29 -0700
Subject: Cooling Stats for 2022/05/16
MIME-Version: 1.0
Content-Type: text/plain; charset=utf-8
X-Antivirus: AVG (VPS 220516-8, 5/16/2022), Inbound message
X-Antivirus-Status: Clean

Bedroom A/C ran for 14 minutes
Main A/C ran for 14 minutes

Peak outside temperature was 85 F
```

As can be seen, this data needs to be parsed before it can be of any practical use.

Parsing the Email Data

A simple program can be written to parse the data and extract the needed data in comma separated format that is usable in Excel. I chose to use C#, but it could be written in any language. My program is shown below for those who wish to use C#. This parses all of the *.eml files (generated by Thunderbird in the previous step) stripping out the required information and stores the data from each file as a line in the CoolingStats.txt output file that is generated in the same folder.

```
//#define CONSOLE_OUTPUT

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;

namespace CoolingStats
{
    // This will parse all of the *.eml Thunderbird email files in the current directory
    //    and generate a text file referenced by rawFileName
    //
    // For this to work the following must be true:
    //   a) the email subject line must contain date and time (ex: My random subject
YY/MM/DD HH:MM)
    //   b) the email body must contain one line for each data point in the following
format
    //        data point description [f9]: 9999999
    //
    //      where: "data point description" is a free form description of this data point
    //             the 9 in [f9]: is a number from 0-9 (only 10 data points are currently
supported)
    //             9999999 is an integer representation of the data points value
    //
    //   c) you can include lines that do not contain the "[f9]:" prefix and they will be
ignored by the parsing
    //      this is often useful for other data that is rarely used


    class Program
    {
        // prefix format in data files parsed:  prefix title string[f9]: value

        enum WhichDataElementPiece
        {
            Name,           // ex: "Peak Outside Temp"
            Token,          // ex: "[f0]"
            Value           // ex: "89"
        }

        static void Main(string[] args)
        {
            //var directory = @"C:\Users\Alec\Desktop\Home Automation\Cooling
Stats\Cooling Stats";
            var directory = @".";
            var rawFileName = directory + @"\RawEmailData.txt";
```

```
            ParseEmailFiles(directory, rawFileName);
        }


        // parse .\*.eml files and generate raw data file
        static void ParseEmailFiles(string directory, string rawDataFileName){

            using (StreamWriter outFile = new StreamWriter(rawDataFileName))
            {
                string[] allFiles = Directory.GetFiles(directory, "*.eml");
                string Title = "Date,Time";
                Boolean TitleDataWritten = false;

                // parse each data file, extract the data and write a single line in the
raw data output file with this data
                foreach (string fileName in allFiles)
                {
                    IEnumerable<string> lines = File.ReadLines(fileName);

                    string subject = "Subject: ";
                    string CSVOutput = "";
                    // it is assumed that the date and time will be embedded in the email
"subject" line in the following formats
                    string timeString = "HH:MM", dateString = "YY/MM/DD";
                    Boolean EndOfHeaderFound = false;

                    foreach (string line in lines)
                    {

                        if (line.StartsWith(subject))
                        {
                            // pick out the date and time from the "subject" line
                            dateString = line.Substring(line.IndexOf('/', subject.Length)
- 2, 8);
                            timeString = line.Substring(line.IndexOf(':', subject.Length)
- 2, 5);

                            CSVOutput = dateString + "," + timeString;
                        }
                        else
                        {
                            if (EndOfHeaderFound)
                            {
                                // if the title data has not been written out yet, then
collect title data from this line
                                if (!TitleDataWritten && (line.Length > 0))
                                {
                                    foreach (string dataValue in
GetDataElementsFromLine(line, WhichDataElementPiece.Name))
                                    {
                                        if (dataValue != null)
                                        {
                                            Title += "," + dataValue;
                                        }
                                    }
                                }
```

```
                                // process all of the data values present on this line
                                foreach (string dataValue in
GetDataElementsFromLine(line, WhichDataElementPiece.Value))
                                {
                                    if (dataValue != null)
                                    {
                                        CSVOutput += "," + dataValue;
                                    }
                                }
                            }
                        }

                        // this needs to be AFTER the parsing
                        if (line.StartsWith("X-Antivirus-Status: Clean"))
                        {
                            // once the email binary file line "just above" the user data
is found
                            // we can start processing the data.  It is risky before this
as "]:"
                            // could exist in the header data
                            EndOfHeaderFound = true;
                        }
                    }

                    // if the title data has not been written out yet then do so
                    if (!TitleDataWritten)
                    {
                        outFile.WriteLine(Title);
                        TitleDataWritten = true;
                    }

                    // write this email files data out to the raw data file
                    outFile.WriteLine(CSVOutput);
                }
            }
        }

        // pick out ALL of the data elements specified by 'which' in this line
        //
        // This is more complicated than it would seem necessary.  The original logic
        // was much simpler but it relied upon the fact that each data element would be
on a seperate line.
        // This worked for quite a while but then a BUG was introduced into the eISY
hardware (May 2024)
        // which caused all of the data elements to show up on the SAME line (essentially
the CR/LF were lost.)
        //
        // Universal Devices doesn't consider this a high priority because if you just
want to "SEE" the
        // data you can switch from 'text' mode to 'HTML' mode and then it will look like
the data is on
        // seperate lines.  HOWEVER, if you are parsing the email file, in both cases the
data is still
        // on ONE line.
        //
        // This logic overcomes that problem and handles the data regardless of line
breaks.
        //
```

```
        static string[] GetDataElementsFromLine(string line, WhichDataElementPiece which)
        {
            const int MAX_ELEMENTS_PER_LINE = 10;
            string[] dataElements = new string[MAX_ELEMENTS_PER_LINE];
            int dataElementIdx = 0;

            int lineLength = line.Length;
            int idx = 0;

            while (idx < lineLength && dataElementIdx < dataElements.Length)
            {
                // only pick out the fields with '[f9]:' in them (ignore optional fields)
                int bracketIdx = line.IndexOf('[', idx);
                int colonIdx = line.IndexOf("]:", idx);
                if (bracketIdx != -1 && colonIdx != -1)
                {
                    // pick out the 'name' part (ex: "Peak Outside Temperature") of the
data element
                    int nameLength = (bracketIdx - idx);
                    var nameString = line.Substring(idx, nameLength);

                    // skip past 'name'
                    idx += nameLength;

                    // pick out the 'token' part (ex: "[f0]") of the data element
                    int tokenLength = colonIdx - bracketIdx + 1;
                    var tokenString = line.Substring(idx, tokenLength);

                    // skip past 'token', ':' and seperating space character
                    idx += tokenLength + 2;

                    // pick out the 'value' part (ex: 89) of the data element
                    int valueEndIdx = line.IndexOf(' ', idx);
                    if (valueEndIdx == -1)
                    {
                        valueEndIdx = lineLength;
                    }
                    var valueString = line.Substring(idx, valueEndIdx - idx);

                    // skip past 'value' and seperating space character
                    idx += (valueEndIdx - idx) + 1;

                    switch (which)
                    {
                        case WhichDataElementPiece.Name:
                            dataElements[dataElementIdx++] = nameString;
                            break;
                        case WhichDataElementPiece.Token:
                            dataElements[dataElementIdx++] = tokenString;
                            break;
                        case WhichDataElementPiece.Value:
                            dataElements[dataElementIdx++] = valueString;
                            break;
                    }
                }
                else
                {
                    // no more required fields found - break out of loop
```

```
                    break;
                }
            }

            return dataElements;
        }
    }
}
```

Running this on my Thunderbird email files generates the following CoolingStats.txt file.

2022/05/15,05:46,58,0,92,0,0,0,0,0
2022/05/15,12:00,338,33,96,0,0,0,0,0
2022/05/15,13:00,22,24,96,0,0,0,0,0
2022/05/15,14:00,0,0,95,0,0,0,0,0

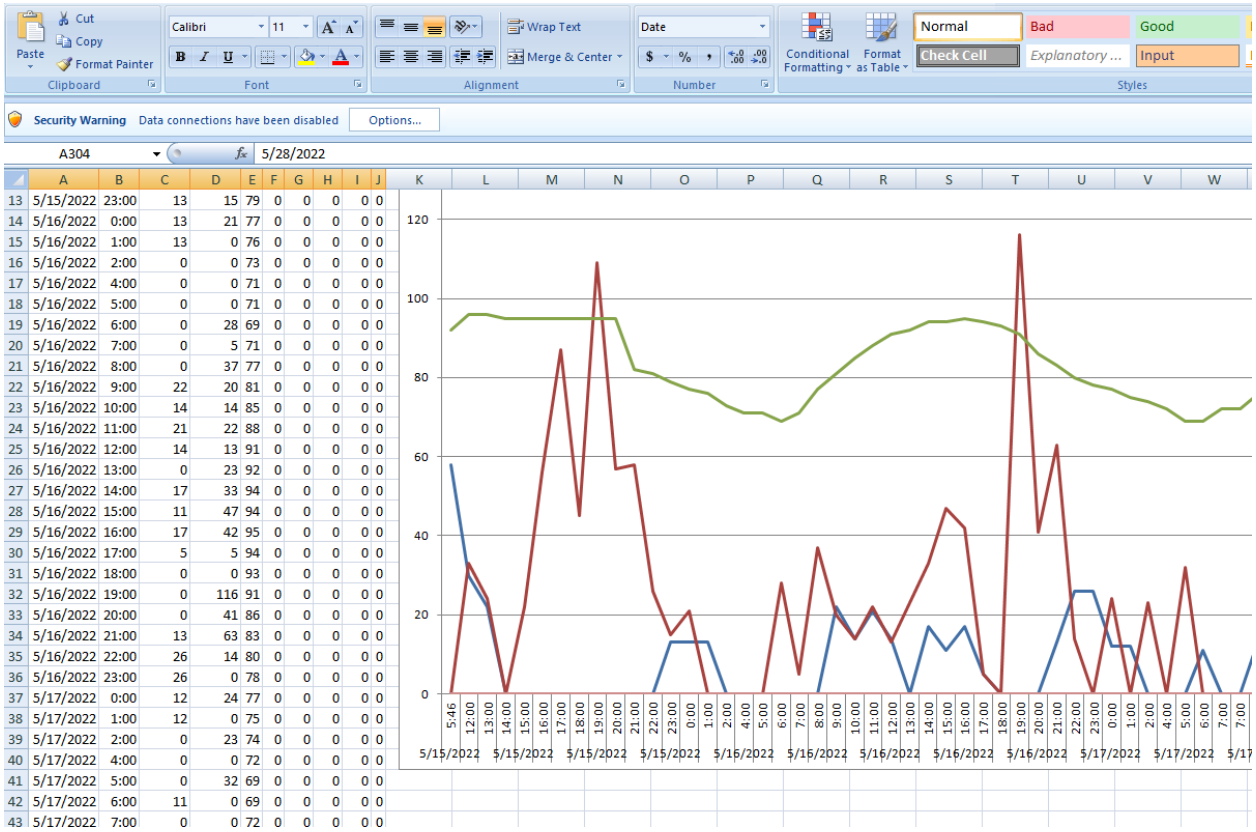
**** LOTS OF DATA REMOVED FOR CLARITY ****

2022/05/29,01:00,12,28,70,0,0,0,0,0
2022/05/29,02:00,0,5,68,0,0,0,0,0
2022/05/29,04:00,0,0,66,0,0,0,0,0
2022/05/29,05:00,0,0,66,0,0,0,0,0
2022/05/29,06:00,0,0,66,0,100,749,568,0
2022/05/29,07:00,213523,213523,68,5,3,2,3,0
2022/05/29,08:00,25,11,71,0,0,61,0,0

Graphing and Using the Email Data

You can then import CoolingStats.txt into Excel and create a graph of the data using well know methods.

- green line is the outdoor temperature
- red line is run time in minutes for the bed room A/C
- blue line is run time in minutes for the main room A/C



	A	B	C	D	E	F	G	H	I	J
13	5/15/2022	23:00	13	15	79	0	0	0	0	0
14	5/16/2022	0:00	13	21	77	0	0	0	0	0
15	5/16/2022	1:00	13	0	76	0	0	0	0	0
16	5/16/2022	2:00	0	0	73	0	0	0	0	0
17	5/16/2022	4:00	0	0	71	0	0	0	0	0
18	5/16/2022	5:00	0	0	71	0	0	0	0	0
19	5/16/2022	6:00	0	28	69	0	0	0	0	0
20	5/16/2022	7:00	0	5	71	0	0	0	0	0
21	5/16/2022	8:00	0	37	77	0	0	0	0	0
22	5/16/2022	9:00	22	20	81	0	0	0	0	0
23	5/16/2022	10:00	14	14	85	0	0	0	0	0
24	5/16/2022	11:00	21	22	88	0	0	0	0	0
25	5/16/2022	12:00	14	13	91	0	0	0	0	0
26	5/16/2022	13:00	0	23	92	0	0	0	0	0
27	5/16/2022	14:00	17	33	94	0	0	0	0	0
28	5/16/2022	15:00	11	47	94	0	0	0	0	0
29	5/16/2022	16:00	17	42	95	0	0	0	0	0
30	5/16/2022	17:00	5	5	94	0	0	0	0	0
31	5/16/2022	18:00	0	0	93	0	0	0	0	0
32	5/16/2022	19:00	0	116	91	0	0	0	0	0
33	5/16/2022	20:00	0	41	86	0	0	0	0	0
34	5/16/2022	21:00	13	63	83	0	0	0	0	0
35	5/16/2022	22:00	26	14	80	0	0	0	0	0
36	5/16/2022	23:00	26	0	78	0	0	0	0	0
37	5/17/2022	0:00	12	24	77	0	0	0	0	0
38	5/17/2022	1:00	12	0	75	0	0	0	0	0
39	5/17/2022	2:00	0	23	74	0	0	0	0	0
40	5/17/2022	4:00	0	0	72	0	0	0	0	0
41	5/17/2022	5:00	0	32	69	0	0	0	0	0
42	5/17/2022	6:00	11	0	69	0	0	0	0	0
43	5/17/2022	7:00	0	0	72	0	0	0	0	0

This allows you to easily view data point trends collected in Polisy to determine corrective action. In my case the data points indicated that (I have two separate A/C units in my house) one of my A/C units (the bed room A/C) was working much harder than the other. The fix for me was to adjust the main room A/C thermostat down one degree so that it would turn on more often. Now they are sharing the load properly.