Artistic Telemetry: A Hybrid Approach

Robert Plates

How to write as much software telemetry as you want with no perceived application performance degradation.

```
11111111111111111111111111111111111111111111111111111111111111111111111111111
11001100110011001100110011001100110011001100110011001111111111111111111111111
11111111111111111111111112211111111111111111111111111111111111100011111111111111
11111111111111111111111112211111111111111111111111111111111110000000111111111111
11100000001111111111111112211111111100111111111111111111111110000000111111111111
11100000001111111111111122211111111100001111111111111111111111100011111111111111
11100000001111111111111122211111110000000111111111111111111111111111111111110000
11100000001111111111112222221110000000000000011111111111111111111111111111000000
11100000001111111111122222222222200000000000000001111111111111111111111111000000000
11100000001111111122222211102222220000000000000000000111111111111111111000000000000
11100000001111112222222111001022222222200000000000000000011111111111000000000000000
11100000001111222222211100000000002222222200000000000000000011111110000000000000000
11100000001222222222000000000000000002222222220000000000000000011111000000000000000000
11100000222222222000000000000000000000000222222220000000000000011100000000000000000000
11102222222220000000000111000000000000000000222222220000000001110000000000000000000
11222222222000000000111111000000000000000000000222222000000001110000000000000000000
22222222000000000000111111000000000000000000000000222220000000110000000000000000000
22222000000000000000001110000000000000000000000000002222200001000000000000000000000
22220000000000000000000000000000000000000000000000002222220000000000000000000000
```

Artistic Telemetry: A Hybrid Approach

Dear Reader: My name is Rob Plates. I am a seasoned software developer and I have been using a hybrid telemetry storage technique that practically eliminates the performance hit when applying telemetry "creatively" inside a software application. I hope you don't mind I share this hybrid telemetry approach in the form of a short story. One with which I hope you can relate to and find what I share in this article interesting and useful.

I think I have a love affair with telemetry. I see beauty or art in how software developers implement telemetry within the software it is reporting on. When I first considered software telemetry, I spoke of it in terms of "logging". I often use the word *logging* interchangeably with the word *telemetry*. I just see the word *telemetry* as being *cooler* at this time in my life.

Metamorphosis in Writing Telemetry

I started my "telemetry" journey by developing a *log file/telemetry writer* for myself that takes care of all the things I wanted in writing telemetry to a text file and/or a database. The telemetry I was writing at the time was simply line by line, play by play, all the time. I hated it. It slowed down my app. But… it worked. I was correcting issues quickly. Everyone was happy. I just could not accept how much slower the app was performing after the telemetry was added. Still, without the telemetry, I could not fix problems as quickly. So, in reality, telemetry and me… …kind of needed each other. *(I think I needed telemetry more than telemetry needed me LOL)*

When considering how to lessen the performance hit associated with writing good telemetry, I considered throttling how much telemetry was being written as an option to getting back some of the performance. I added a "verbosity level" to my telemetry writer so I could raise and lower how much telemetry was being written in real time. That helped but…meh!

Then a reactive element, one that reacts to "error" log entries, was added to my telemetry writer. The plan was; if the verbosity level was *low* or *off* at the time an error or exception log entry was written, the programming would automatically raise the verbosity level. Thus, outputting the additional telemetry intended to help with debugging. This worked at capturing telemetry when needed and was very helpful in keeping telemetry resources (storage) from being gobbled up with unwanted data.

After more than a while, technology changed. Solid State Drives came into being. The Internet was now a pipeline for Audio Visual media as well as raw data. The underlying hardware/firmware architecture changed from being memory stingy, to allowing use of sizeable chunks of memory. When I returned to thinking about improving telemetry output performance, I realized there was another way I could speed up the application while keeping the amount of telemetry data at a useful level. I started to closely examine *the process of storing a telemetry entry (see below).* As a result, I started using computer memory as a temporary storage location for telemetry data before I actually write it to a file or database. There is sizable performance gain for doing that without losing the amount of telemetry data being written.

I then added writing to memory storage buffers to my telemetry writer. Rather than write the data to the permanent storage medium in the middle of the application process, we store the telemetry data in an application memory buffer. After the application process is over, we then write the telemetry data in the application memory buffer to the permanent storage. By writing all telemetry data to permanent storage once, we get a significant performance gain even with a larger data payload. *(See the Test Results section below.)*

My telemetry design seriously changed for me after doing that. At this point in my telemetry analysis journey, I was using combinations of memory to file, memory to database, even memory to debug console.

The technique itself is easy.
In terms of .NET Programming: Instead of writing to the "logger" during the middle of a process I want monitored, I write the telemetry to a .NET StringBuilder object declared inside the process. *(Or within the scope of what you want to*

implement.) At the end of the process, I write a single entry to the "Logger" with all the telemetry data inside the StringBuilder object. In this scenario the StringBuilder object acted as the memory storage medium.

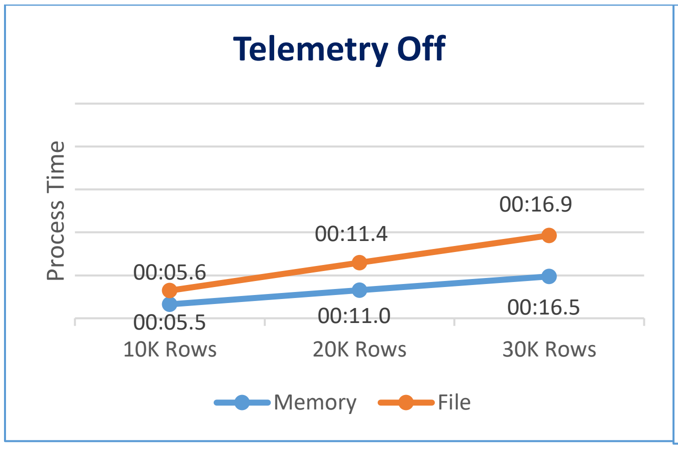
Test Results

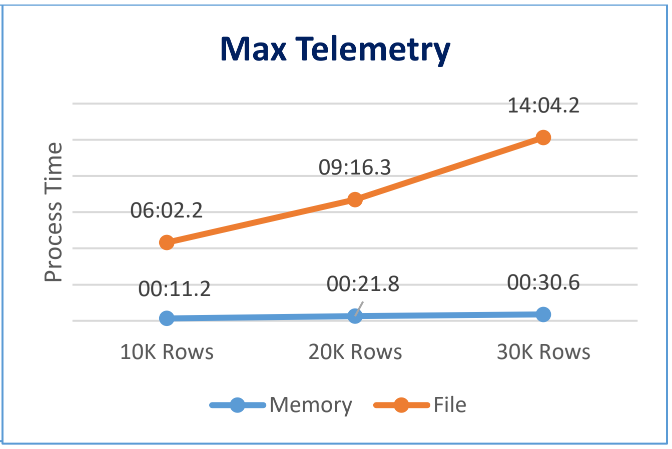
When I started using the technique, solid state disk drives (SSD's) were just coming into the mainstream PC builds. One would "think" that writing to memory-based disk drives would eliminate the performance hit associated with writing to a disk file.

I wrote a test program that would read a significant number of rows from a database and wrote telemetry to monitor the process. For this article I performed 3 tests. **Test 1** reading 10K rows of data. **Test 2** reading 20K rows of data. **Test 3** reading 30K rows of data. The table being read contained 128 columns of data. (No Images or blob fields) The test was performed on up to date modern computer equipment in a networked environment. Tests 1, 2, and 3 were performed with telemetry turned off. Then the tests were run again with telemetry at maximum output. The test program reported the time it started and the time it ended on the test app window for me to collect and add to a spreadsheet that compiled all the data for each individual test.

The respective sizes of telemetry data files were: Test 1 10K resulted in a 14.5MB file. Test 2 20K resulted in a 174.2MB file., Test 3 30K resulted in 261.3MB data file.

While performing the tests, I realized that the performance benefit was significant. Especially when test3 was performed. I realized *how much* of a benefit, when I compiled the test results. The charts below summarize the compiled test data.





Value Format: mm:ss.s (minutes/seconds)

Chart Legend

Process Time: The overall time it took the application to read the records and save the telemetry data.
File: Telemetry written line by line directly to permanent storage.
Memory: Telemetry written using the hybrid memory technique.

What can we conclude from the results?

A 13-minute performance improvement using the hybrid technique is significant.

It would be interesting to explore the performance relationship within the Hybrid memory telemetry technique for the possibility of a logarithmic scale. Furthermore, to answer questions like: Does using faster memory in SSD's (or a better class of SSD) close the gap in the results from these tests?

The processes of storing a telemetry entry

Of course, I am overlooking “true” radio telemetry (think motorsports…). Radio telemetry breaks down data encoded in electronic format to software values for later use in the information chain. For this article I am focusing on the software development storage side of telemetry.

The processes I am referring to, focus on what the computer does to perform the actual storage of the data to the selected storage location. When you consider what happens in the computer memory just to save one or more characters to a file, you start to appreciate that a lot of steps are taken by the computer to perform the save process.

Let's just focus on writing a text file log entry. From the software developer’s perspective, we are “simply” writing a string of characters to a file. (Typically, 40 to 80 characters but occasionally much longer) We don’t consider that there is a process respecting the transfer of the telemetry data to storage. We’re not supposed to.

When we think about how the computer moves the data to and from the memory locations and registers the CPU uses to transfer data to the external storage, there are a LOT of steps. The telemetry data is copied into and out of Memory buffers and CPU registers multiple times before it gets to its destination in the file storage.

Saving a row to a database I would estimate carries double to triple the processing steps as the text file consideration above.

A consideration like this makes one appreciate how truly fast the computers we work with are nowadays.

Benefits of the Hybrid Telemetry Technique

There are 3 benefits that I can think of worth putting in this article. As follows:

Get significantly less of a performance hit on the application.

As discussed in the previous section, using the hybrid technique in a well strategized manner relieves a majority of the performance hit associated with telemetry writing.

Flexibility to implement "temporary" telemetry for resolving difficult to find software anomalies.

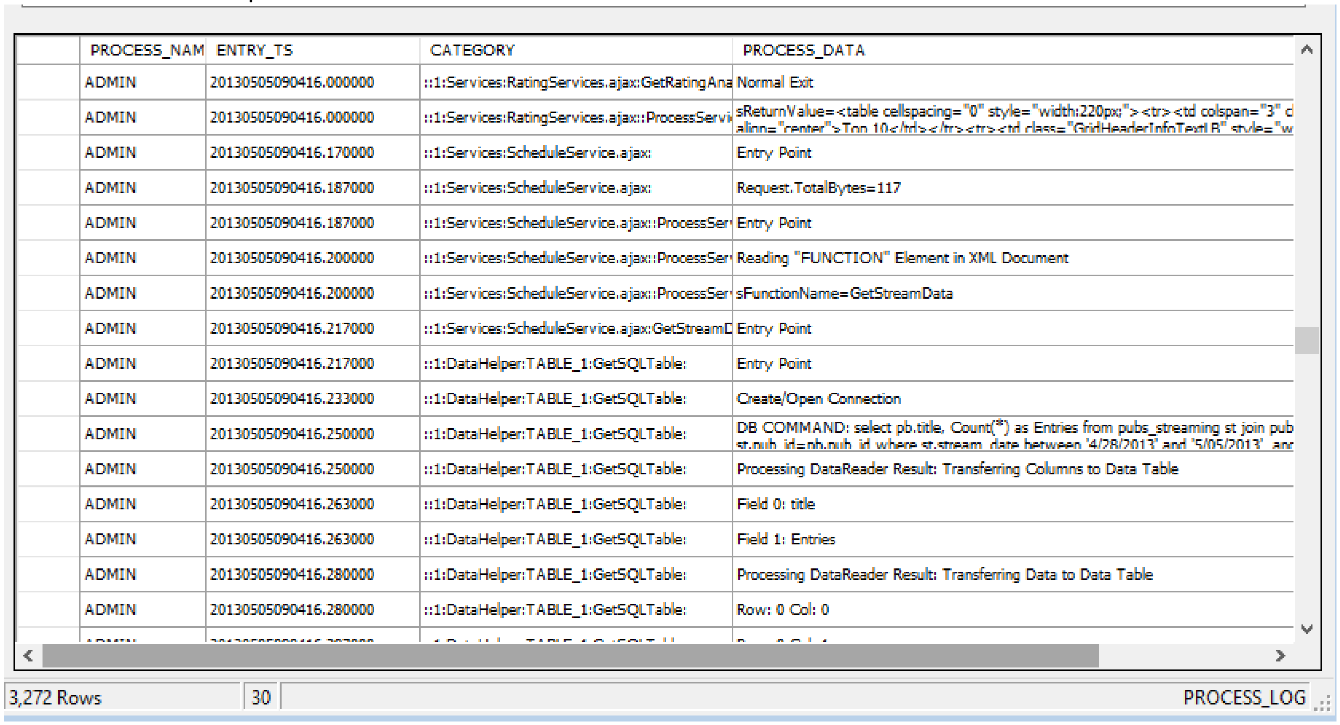
There have been many times I have added "temporary" telemetry using this hybrid technique when tracking down issues that cannot be or are much too difficult to setup in a software debugging environment. Once the bug was found and repaired, I shelved the telemetry writing source code.

Easier searching and viewing of specific telemetry data

After I found the telemetry data I was looking for, it was far easier to view and analyze as a unit. The telemetry data saved using this technique eliminates the need to "re-assemble" the telemetry. When telemetry data is written line by line to a database for storage, it has to be re-assembled "line by line" when you need to debug an issue. *(Considering splunk, the same is now true of text files!)*

Database Telemetry Analysis

The screenshot below is showing the result-set of data from an application that wrote to a database log table line by line as the software process executed. The telemetry data sits in the PROCESS_DATA column. As you can see, most of the entries are short and simple.



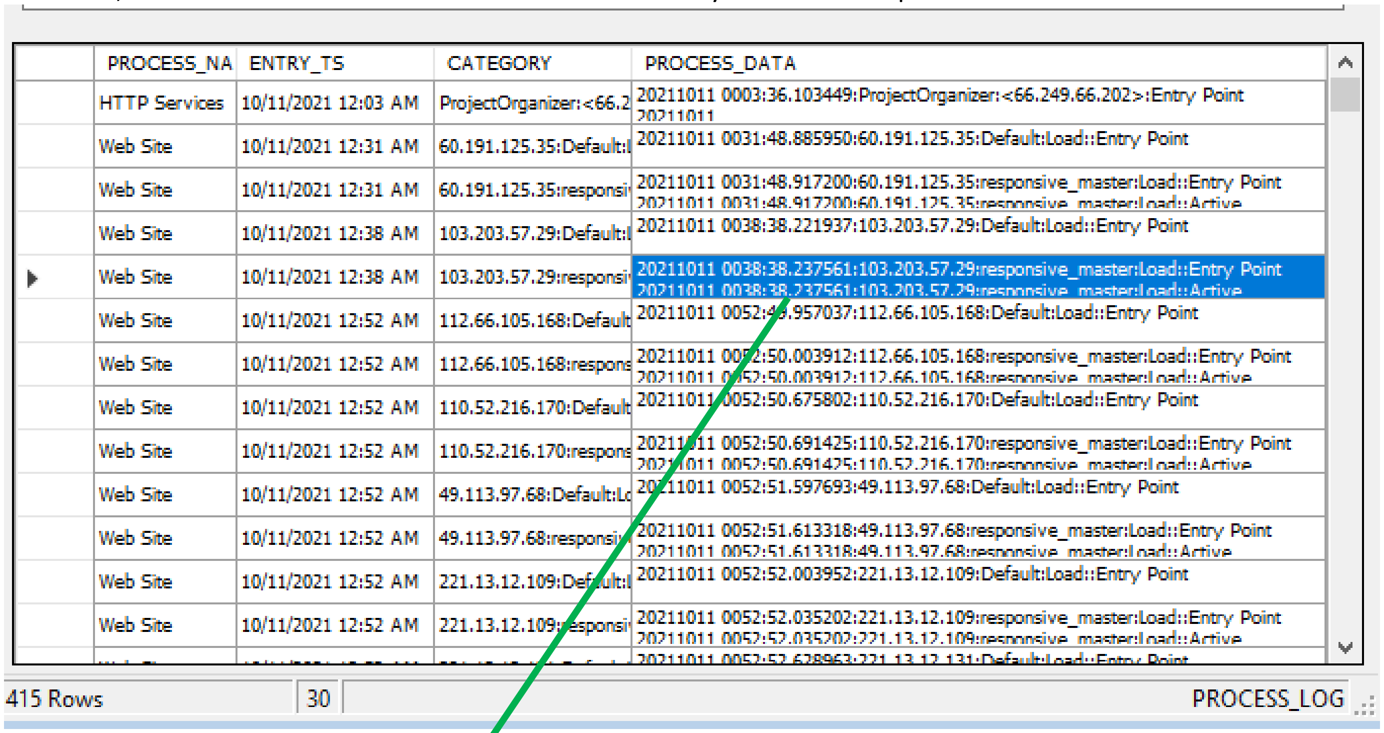
	PROCESS_NAM	ENTRY_TS	CATEGORY	PROCESS_DATA
	ADMIN	20130505090416.000000	::1:Services:RatingServices.ajax:GetRatingAna	Normal Exit
	ADMIN	20130505090416.000000	::1:Services:RatingServices.ajax::ProcessServi	sReturnValue=<table cellspacing="0" style="width:220px;"><tr><td colspan="3" c align="center">Top 10</td></tr><tr><td class="GridHeaderInfoTextLB" style="w
	ADMIN	20130505090416.170000	::1:Services:ScheduleService.ajax:	Entry Point
	ADMIN	20130505090416.187000	::1:Services:ScheduleService.ajax:	Request.TotalBytes=117
	ADMIN	20130505090416.187000	::1:Services:ScheduleService.ajax::ProcessSer	Entry Point
	ADMIN	20130505090416.200000	::1:Services:ScheduleService.ajax::ProcessSer	Reading "FUNCTION" Element in XML Document
	ADMIN	20130505090416.200000	::1:Services:ScheduleService.ajax::ProcessSer	sFunctionName=GetStreamData
	ADMIN	20130505090416.217000	::1:Services:ScheduleService.ajax:GetStreamD	Entry Point
	ADMIN	20130505090416.217000	::1:DataHelper:TABLE_1:GetSQLTable:	Entry Point
	ADMIN	20130505090416.233000	::1:DataHelper:TABLE_1:GetSQLTable:	Create/Open Connection
	ADMIN	20130505090416.250000	::1:DataHelper:TABLE_1:GetSQLTable:	DB COMMAND: select pb.title, Count(*) as Entries from pubs_streaming st join pub st.pub_id=pb.pub_id where st.stream_date between '4/28/2013' and '5/05/2013' anc
	ADMIN	20130505090416.250000	::1:DataHelper:TABLE_1:GetSQLTable:	Processing DataReader Result: Transferring Columns to Data Table
	ADMIN	20130505090416.263000	::1:DataHelper:TABLE_1:GetSQLTable:	Field 0: title
	ADMIN	20130505090416.263000	::1:DataHelper:TABLE_1:GetSQLTable:	Field 1: Entries
	ADMIN	20130505090416.280000	::1:DataHelper:TABLE_1:GetSQLTable:	Processing DataReader Result: Transferring Data to Data Table
	ADMIN	20130505090416.280000	::1:DataHelper:TABLE_1:GetSQLTable:	Row: 0 Col: 0

3,272 Rows | 30 | PROCESS_LOG

Continued next page

Pictured below is the log of an application writing telemetry using this hybrid technique!

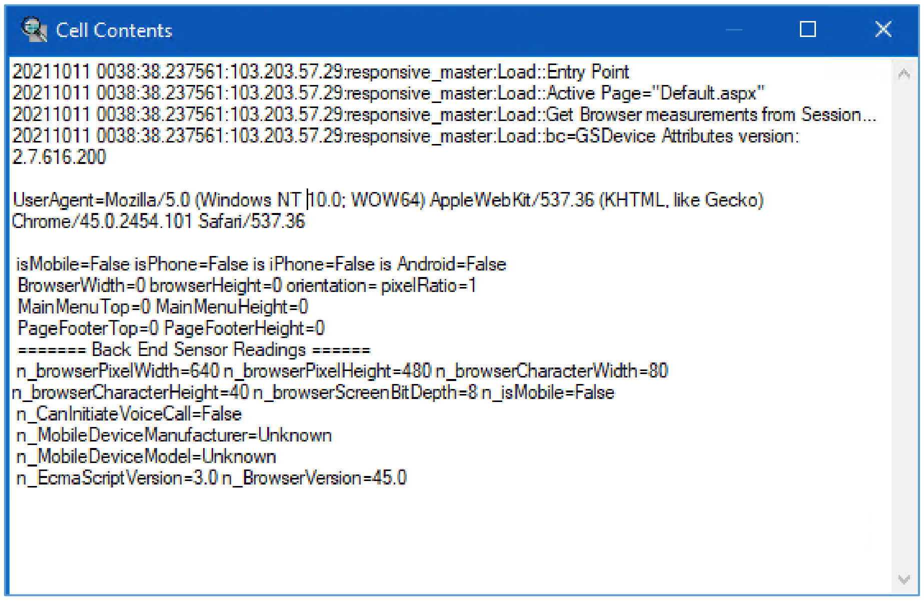
Just like the line by line example above, the telemetry data is in the PROCESS_DATA column. However, unlike the line by line method, the column for each row now holds the telemetry for the whole process.



	PROCESS_NA	ENTRY_TS	CATEGORY	PROCESS_DATA
	HTTP Services	10/11/2021 12:03 AM	ProjectOrganizer:<66.2	20211011 0003:36.103449:ProjectOrganizer:<66.249.66.202>:Entry Point 20211011
	Web Site	10/11/2021 12:31 AM	60.191.125.35:Default:l	20211011 0031:48.885950:60.191.125.35:Default:Load::Entry Point
	Web Site	10/11/2021 12:31 AM	60.191.125.35:responsi	20211011 0031:48.917200:60.191.125.35:responsive_master:Load::Entry Point 20211011 0031:48.917200:60.191.125.35:responsive_master:Load::Active
	Web Site	10/11/2021 12:38 AM	103.203.57.29:Default:l	20211011 0038:38.221937:103.203.57.29:Default:Load::Entry Point
▶	Web Site	10/11/2021 12:38 AM	103.203.57.29:responsi	20211011 0038:38.237561:103.203.57.29:responsive_master:Load::Entry Point 20211011 0038:38.237561:103.203.57.29:responsive_master:Load::Active
	Web Site	10/11/2021 12:52 AM	112.66.105.168:Default	20211011 0052:49.957037:112.66.105.168:Default:Load::Entry Point
	Web Site	10/11/2021 12:52 AM	112.66.105.168:respons	20211011 0052:50.003912:112.66.105.168:responsive_master:Load::Entry Point 20211011 0052:50.003912:112.66.105.168:responsive_master:Load::Active
	Web Site	10/11/2021 12:52 AM	110.52.216.170:Default	20211011 0052:50.675802:110.52.216.170:Default:Load::Entry Point
	Web Site	10/11/2021 12:52 AM	110.52.216.170:respons	20211011 0052:50.691425:110.52.216.170:responsive_master:Load::Entry Point 20211011 0052:50.691425:110.52.216.170:responsive_master:Load::Active
	Web Site	10/11/2021 12:52 AM	49.113.97.68:Default:Lo	20211011 0052:51.597693:49.113.97.68:Default:Load::Entry Point
	Web Site	10/11/2021 12:52 AM	49.113.97.68:responsi	20211011 0052:51.613318:49.113.97.68:responsive_master:Load::Entry Point 20211011 0052:51.613318:49.113.97.68:responsive_master:Load::Active
	Web Site	10/11/2021 12:52 AM	221.13.12.109:Default:l	20211011 0052:52.003952:221.13.12.109:Default:Load::Entry Point
	Web Site	10/11/2021 12:52 AM	221.13.12.109:responsi	20211011 0052:52.035202:221.13.12.109:responsive_master:Load::Entry Point 20211011 0052:52.035202:221.13.12.109:responsive_master:Load::Active
	[illegible]	[illegible]	[illegible]	20211011 0052:52.628963:221.13.12.131:Default:Load::Entry Point

415 Rows | 30 | PROCESS_LOG

Pictured below are the full contents of the PROCESS_DATA column cell of the display grid.



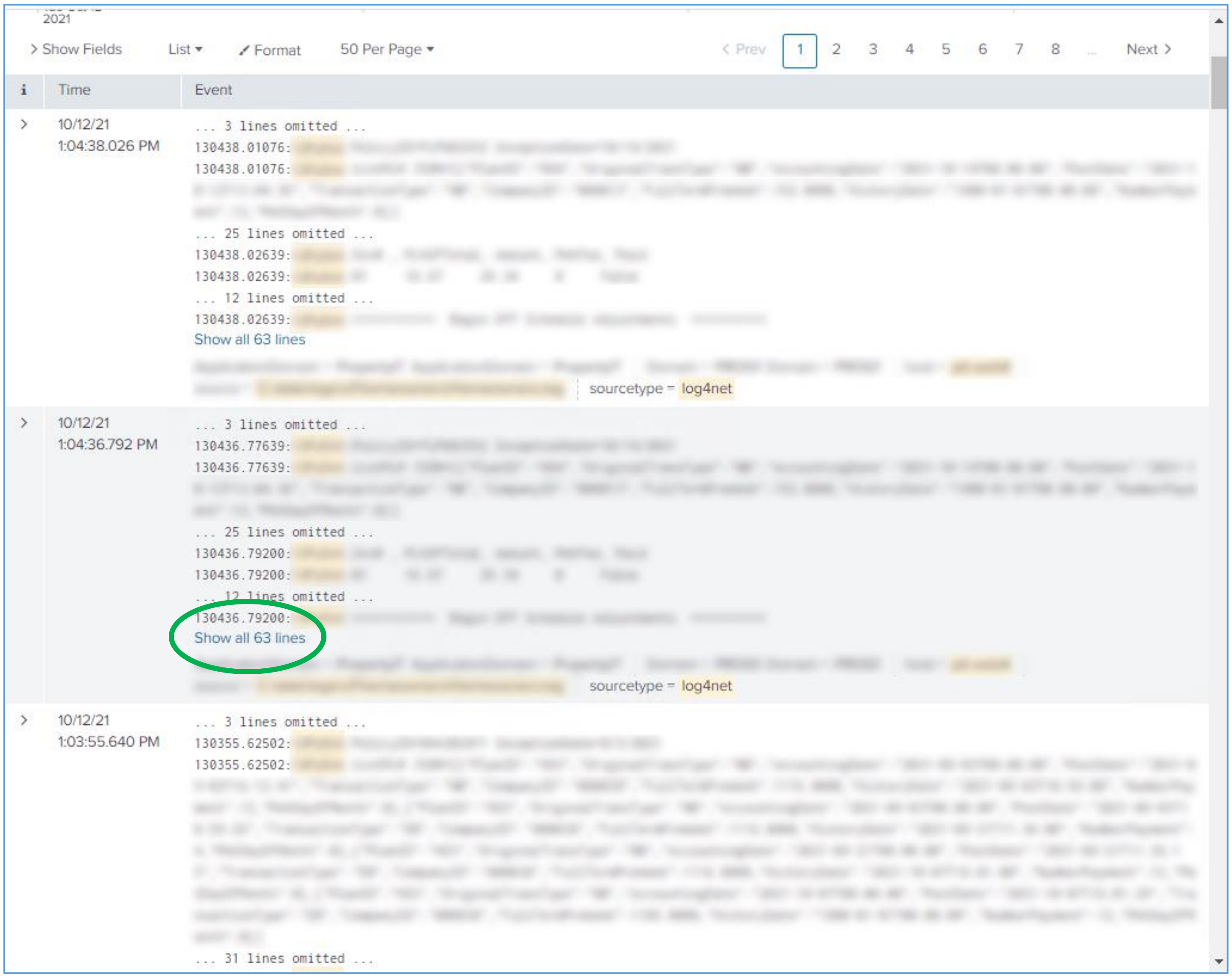
```
20211011 0038:38.237561:103.203.57.29:responsive_master:Load::Entry Point
20211011 0038:38.237561:103.203.57.29:responsive_master:Load::Active Page="Default.aspx"
20211011 0038:38.237561:103.203.57.29:responsive_master:Load::Get Browser measurements from Session...
20211011 0038:38.237561:103.203.57.29:responsive_master:Load::bc=GSDevice Attributes version:
2.7.616.200

UserAgent=Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/45.0.2454.101 Safari/537.36

 isMobile=False isPhone=False is iPhone=False is Android=False
 BrowserWidth=0 browserHeight=0 orientation= pixelRatio=1
 MainMenuTop=0 MainMenuHeight=0
 PageFooterTop=0 PageFooterHeight=0
 ======= Back End Sensor Readings ======
 n_browserPixelWidth=640 n_browserPixelHeight=480 n_browserCharacterWidth=80
n_browserCharacterHeight=40 n_browserScreenBitDepth=8 n_isMobile=False
 n_CanInitiateVoiceCall=False
 n_MobileDeviceManufacturer=Unknown
 n_MobileDeviceModel=Unknown
 n_EcmaScriptVersion=3.0 n_BrowserVersion=45.0
```

Text File Telemetry Analysis

Splunk is a monitoring software application that excels at compiling quasi synonymous text file logs from disparate locations and presenting them in a single queryable fashion.

One of my challenges when using the splunk tool however, is the limitations of the number of lines in the results returned to the browser. My workspace has a maximum limit of 50 lines. The splunk version I am using IS a web browser application and the underlying data is huge, so the limitation is understandable. However, for some processes that is not enough of a window to view the telemetry for a whole process when writing telemetry line by line. By writing telemetry using this hybrid technique the challenge is addressed.

The screen shot below is from an application process that stored its telemetry using this hybrid technique. The telemetry for the process is embedded in a single line identifying the process for later lookup.

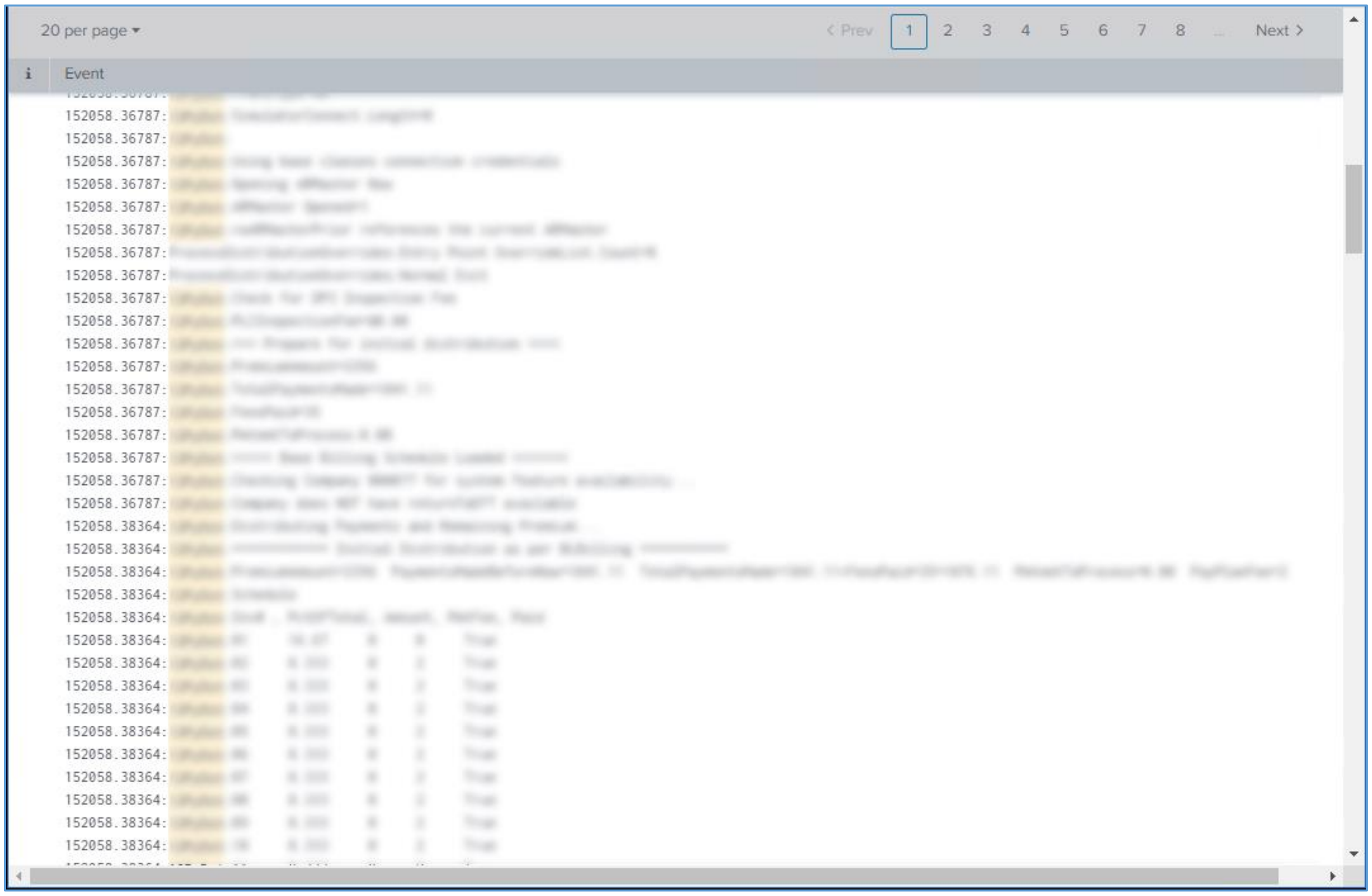


i	Time	Event
>	10/12/21 1:04:38.026 PM	... 3 lines omitted ... 130438.01076: [redacted] 130438.01076: [redacted] ... 25 lines omitted ... 130438.02639: [redacted] 130438.02639: [redacted] ... 12 lines omitted ... 130438.02639: [redacted] Show all 63 lines [redacted] sourcetype = log4net
>	10/12/21 1:04:36.792 PM	... 3 lines omitted ... 130436.77639: [redacted] 130436.77639: [redacted] ... 25 lines omitted ... 130436.79200: [redacted] 130436.79200: [redacted] ... 12 lines omitted ... 130436.79200: [redacted] Show all 63 lines [redacted] sourcetype = log4net
>	10/12/21 1:03:55.640 PM	... 3 lines omitted ... 130355.62502: [redacted] 130355.62502: [redacted] ... 31 lines omitted ...

Each "line" contains the telemetry for the complete process from beginning to end. The splunk application adds the feature of expanding the lines by offering the "Show all 63 lines" link shown in the image above.

The image on the next page shows the screen after expanding the Show All 63 Lines link.

This is convenient way to analyze your application telemetry when you get there.



```
152058.36787: [illegible]
152058.36787: [illegible]
152058.36787: [illegible]
152058.36787: [illegible]
152058.36787: [illegible]
152058.36787: [illegible]
152058.36787: [illegible]
152058.36787: [illegible]
152058.36787: [illegible]
152058.36787: [illegible]
152058.36787: [illegible]
152058.36787: [illegible]
152058.36787: [illegible]
152058.36787: [illegible]
152058.36787: [illegible]
152058.36787: [illegible]
152058.36787: [illegible]
152058.36787: [illegible]
152058.38364: [illegible]
152058.38364: [illegible]
152058.38364: [illegible]
152058.38364: [illegible]
152058.38364: [illegible]
152058.38364: [illegible]
152058.38364: [illegible]
152058.38364: [illegible]
152058.38364: [illegible]
152058.38364: [illegible]
152058.38364: [illegible]
152058.38364: [illegible]
152058.38364: [illegible]
152058.38364: [illegible]
152058.38364: [illegible]
```

I also see a benefit in that I don't have to switch between browser pages to see my telemetry data for the whole process. Especially when I am looking for where a flaw in a process might be. I can now just scroll through the browser page to see the telemetry for the whole process.

Conclusion

I hope that you find this hybrid telemetry technique useful.

Especially when monitoring your own software, appreciating the art of telemetry makes maintaining a software application much more enjoyable. Happy logging.