LabVIEW Simplifies Viewing of Very Large Text Files

Author
Benjamin A. Rayner, Senior Automation Systems Architect, Data Science Automation, Inc.

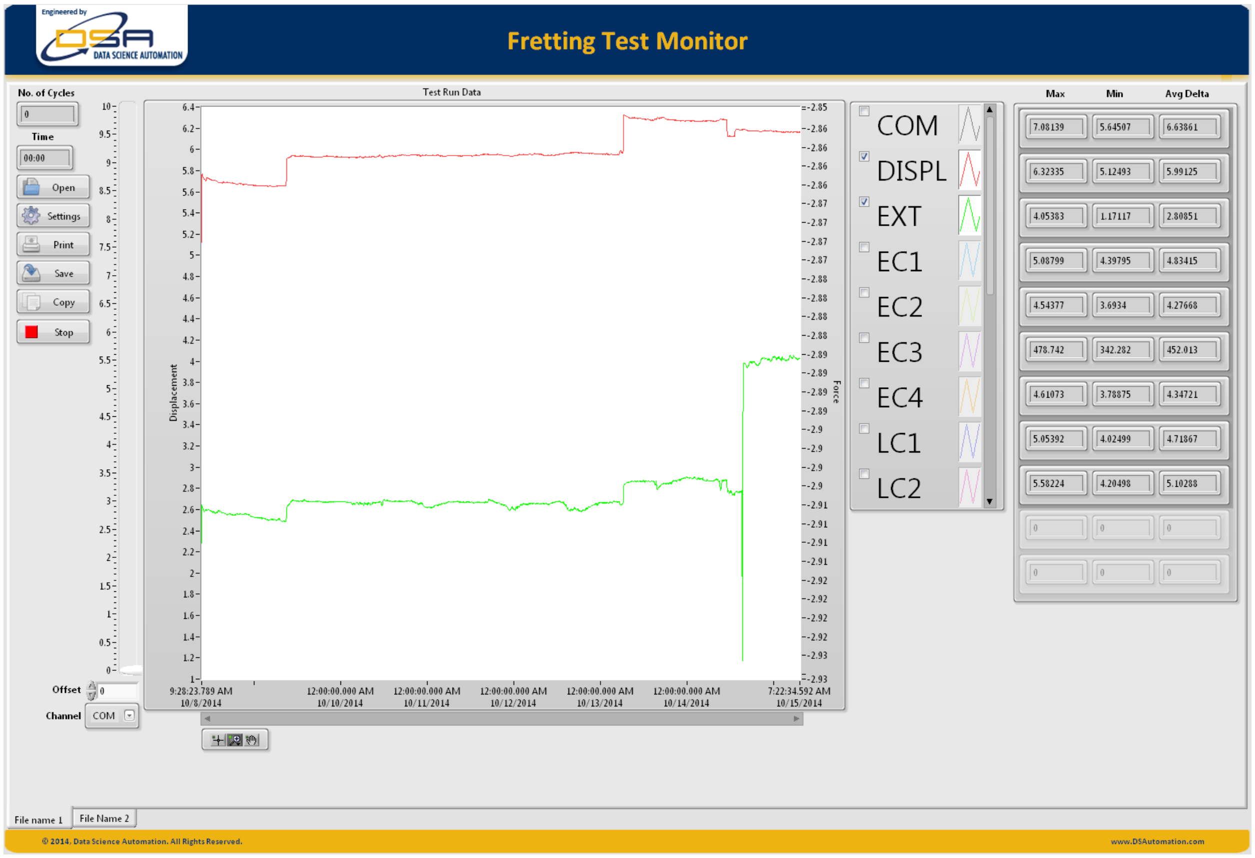
NI Products Used
LabVIEW 2014

Category
Energy
Industrial Machinery & Control

The Challenge
Create a user-friendly interface to view extremely large data files in a single operator screen, even as they are being updated from ongoing tests of advanced energy components.

The Solution
We utilized the power and adaptability of NI LabVIEW, with precision memory management to preserve system resources, while allowing presentation of relevant data.

Introduction
Data Science Automation (DSA) is a premier National Instruments (NI) Alliance Partner that specializes in automating and educating the world leading companies. Clients choose DSA because of DSA’s deep knowledge of National Instruments products, disciplined process of developing adaptive project solutions, staff of skilled Certified LabVIEW Architects and Certified Professional Instructors, and unique focus on empowerment through education and co-development.
A Routine Project Presents Some Challenges But Nothing that LabVIEW Can’t Handle!


Engineered by
DSA DATA SCIENCE AUTOMATION
Fretting Test Monitor
No. of Cycles
Time
Open
Settings
Print
Save
Copy
Stop
Test Run Data
Displacement
Force
Offset
Channel
COM
DISPL
EXT
EC1
EC2
EC3
EC4
LC1
LC2
9:28:23.789 AM 10/8/2014
12:00:00.000 AM 10/10/2014
12:00:00.000 AM 10/11/2014
12:00:00.000 AM 10/12/2014
12:00:00.000 AM 10/13/2014
12:00:00.000 AM 10/14/2014
7:22:34.592 AM 10/15/2014
File name 1
File Name 2
© 2014. Data Science Automation. All Rights Reserved.
www.DSAutomation.com

Max	Min	Avg Delta
7.08139	5.64507	6.63861
6.32335	5.12493	5.99125
4.05383	1.17117	2.80851
5.08799	4.39795	4.83415
4.54377	3.6934	4.27668
478.742	342.282	452.013
4.61073	3.78875	4.34721
5.05392	4.02499	4.71867
5.58224	4.20498	5.10288
0	0	0
0	0	0

Figure 1 Fretting Test Monitor Main Screen

A fretting test can take days or weeks (see figure 1) to simulate the effects of years of interactions between moving parts. Combining rapid cycling with long duration runs and we get very large files. When the data files produced by the test machine are stored as tab-delimited text files, file sizes can easily exceed 2 Gigabyte. Data Science Automation was asked to develop a viewer that is capable of reading the large files and display them using a variety of display formats.
While many files containing videos or images can easily exceed 2 Gigabytes, tab-delimited files that large are rare. Add to the challenge of a large tab-delimited file the need for the special processing and display, a requirement to monitor and update the display as the file is updated to allow monitoring the test live we discover a unique challenge.
LabVIEW was chosen to take develop the application because of its quick development environment, readily customized graphing capabilities, and native multithreading.

Challenges

File Too Large For “Read From Spreadsheet File”

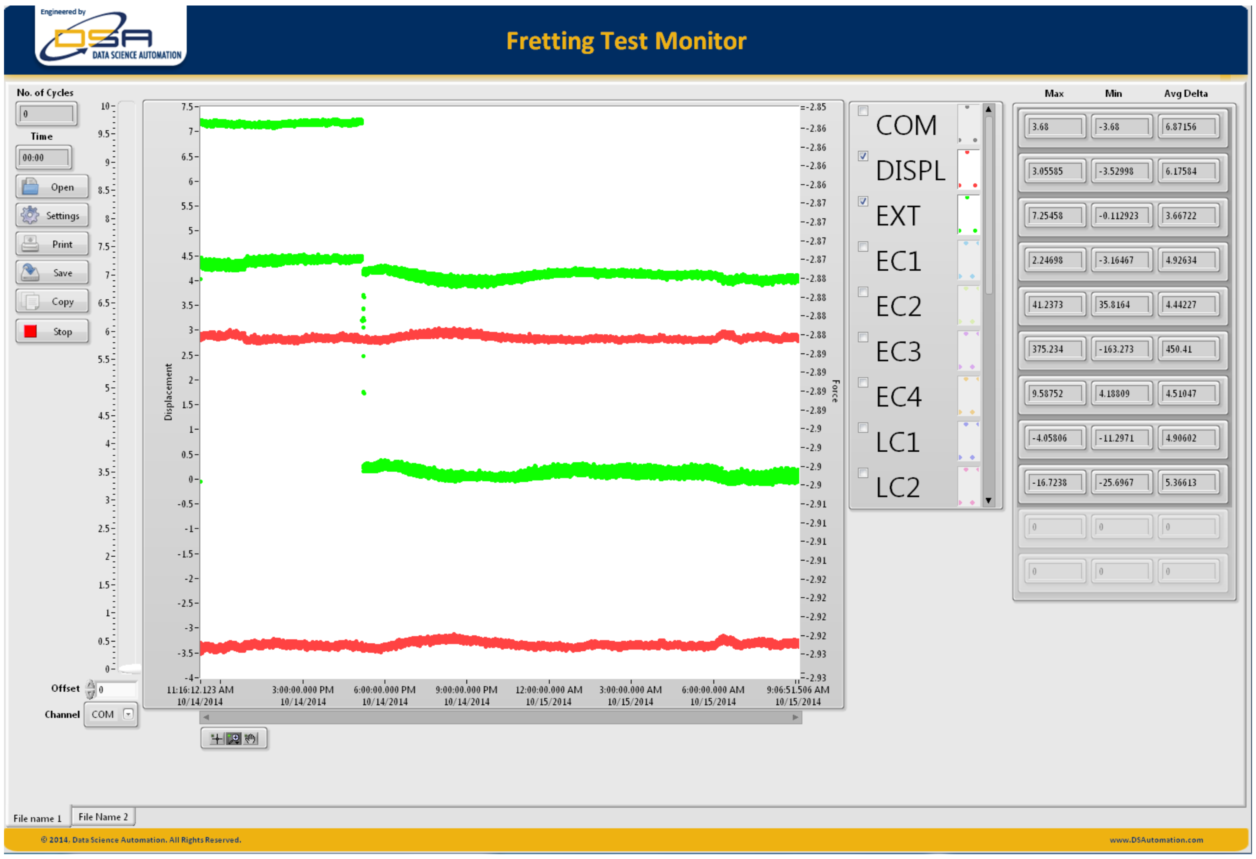
Spreadsheet files are expected to be small compared with the amount of memory installed in a machine. When working with reasonably sized spreadsheet files the entire file can be read at one time. The File Function that ships with LabVIEW named “Read From Spreadsheet File” will check the size of a file and will read the entire file, and convert to the desired format and return the formatted data to the caller. A quick test using that function failed with a strange message indicating an invalid parameter was passed by the caller. Considering that there is only one parameter passed in by the caller, namely, the file specification it appeared the path was incorrect. Further investigation revealed the nature of the failure. “Read From Spreadsheet” was checking the size of the file and using the size to determine how much data should be read from the disk drive. With a files large than 2 Gigabytes we run into a situation where the byte count can’t be represented using an I32. Files larger than 2 gigabytes require more than 31 bits. When a number that is larger than 31 bits is passed to a control configured as I32, the value appears to be a negative value. A negative byte count is invalid and explained the root cause of the error.

Thankfully NI exposes the functionality built in to the “Read From Spreadsheet File” function and allows us to use the shipping functions as a starting point from which we can customize the operation. The shipping functions were modified to use an I64 for all of the file related operations and after modifying all of the related file operations, it was possible to open the file.

Reading the Entire File Quickly Fills Memory
Reading the text contained in a file is only the first step required to locate the data of interest. The text still has to be converted to a numeric form to allow the numeric values to be plotted. We were able to fend off this challenge by reading the text file in small chunks and converting the text to numeric’s before moving on to the next block of text contained in the file. This approached helped us keep the memory demands down to a size that was related to the amount of data in the file rather the number of bytes required to describe the data in a text form.

Not Just Numbers
The log files that were of interest were not simply a few column headers followed by a long set of time stamped data. The software that produces the file allows logging to be started and stopped while the test (that can run for weeks) is running. This fact placed an additional challenge on the file viewing application. The file had to be analyzed to determine where each data section started and stopped as well as the stat time associated with each data section. This was addressed by implementing a scanning routine that would locate the byte offset into the file of each data section. The start time along with the start and stop offset into the file were then used define part of the data file that contained each section of the data. The associated start time was used to define the start time of the data set in each section.

Parsing and Converting Take a Long Time
Reading each byte contained in the data file and determining the significance of the value in the context of the data of the data as a whole kept a single processor busy. The work was shared across multiple cores of the CPU by splitting the work associated with identifying the data section from the code that converted the reading and timestamps. A “Producer-Consumer” architecture was implemented using a fixed size queue between the Producer and the Consumer. As each data section, start time, start and stop byte offsets were identified, a message was passed from the Producer to the Consumer using the fixed size queue. The “Producer-Consumer” architecture allowed one of the CPU cores to work to identify sections while the other core performed the tasks associated with reading the data values in each section. Using a fixed size queue allowed us to realize automatic regulation of the producer when the consumer fell behind reading the data. When the queue was full, the Producer will simply stall waiting for the queue backlog to be emptied by the consumer.


Engineered by DSA DATA SCIENCE AUTOMATION
Fretting Test Monitor
Open
Settings
Print
Save
Copy
Stop
Displacement
Force
COM
DISPL
EXT
EC1
EC2
EC3
EC4
LC1
LC2
11:16:12.123 AM 10/14/2014
3:00:00.000 PM 10/14/2014
6:00:00.000 PM 10/14/2014
9:00:00.000 PM 10/14/2014
12:00:00.000 AM 10/15/2014
3:00:00.000 AM 10/15/2014
6:00:00.000 AM 10/15/2014
9:06:51.506 AM 10/15/2014
File name 1
File Name 2
© 2014. Data Science Automation. All Rights Reserved.
www.DSAutomation.com

No. of Cycles 0

Time 00:00

Offset 0

Channel COM

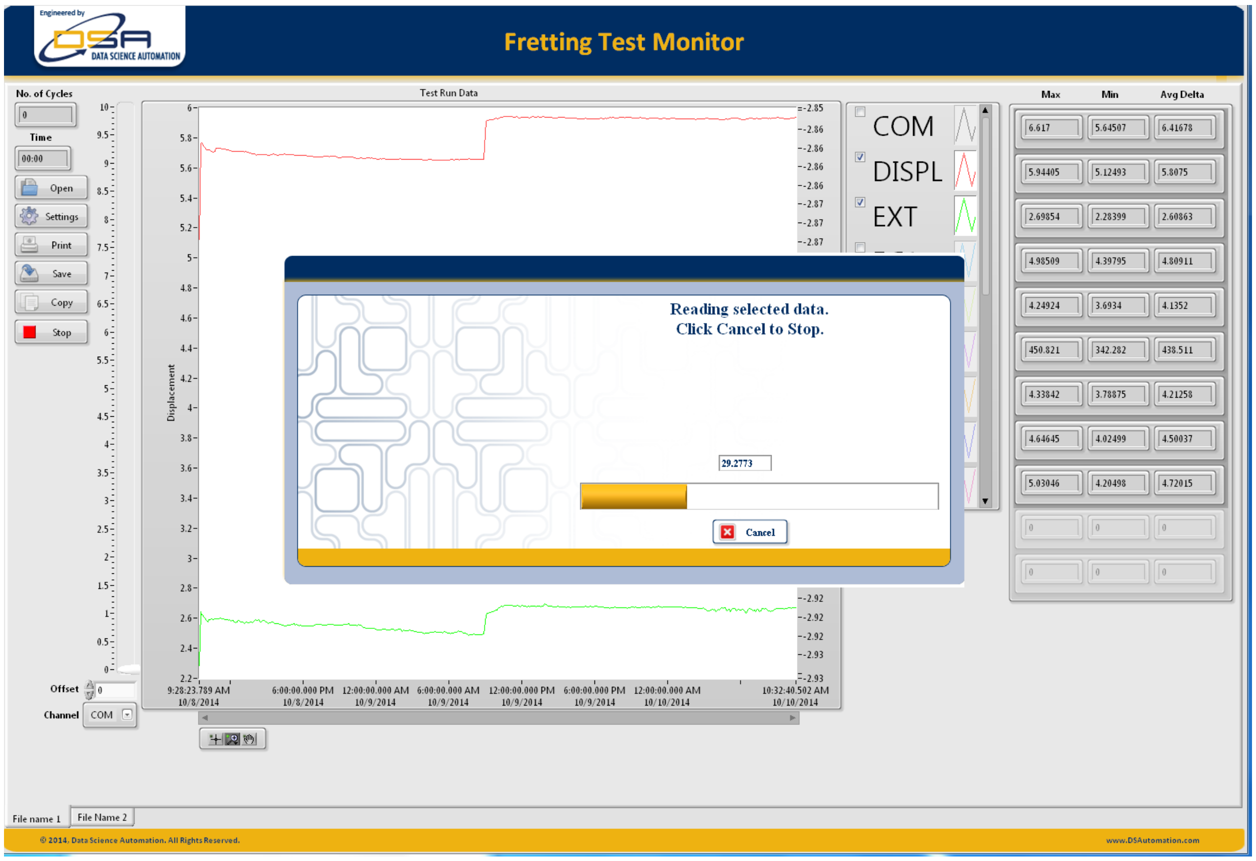
Max	Min	Avg Delta
3.68	-3.68	6.87156
3.05585	-3.52998	6.17584
7.25458	-0.112923	3.66722
2.24698	-3.16467	4.92634
41.2373	35.8164	4.44227
375.234	-163.273	450.41
9.58752	4.18809	4.51047
-4.05806	-11.2971	4.90602
-16.7238	-25.6967	5.36613
0	0	0
0	0	0

Figure 2 See-Saw Display

Special Processing Requirements

The need to reduce the amount of data displayed on the chart presented another CPU intensive task. Two forms of data reduction were required. The first used a user specified number of sample that would be skipped between each point rendered on the chart. The second method used a similar number of values to skip before plotting a pair of data points on two separate plots that were configured using the same color for a point style chart with no connecting lines between points. This latter style was called “See Saw” (see figure 2) and was intended to illustrate the maximum and minimum values of the monitored values.

A similar approach to the Producer-Consumer scheme described above was used with the file read operations being the producer and the analysis and formatting being the consumer.



Reading selected data.
Click Cancel to Stop.

29.2773

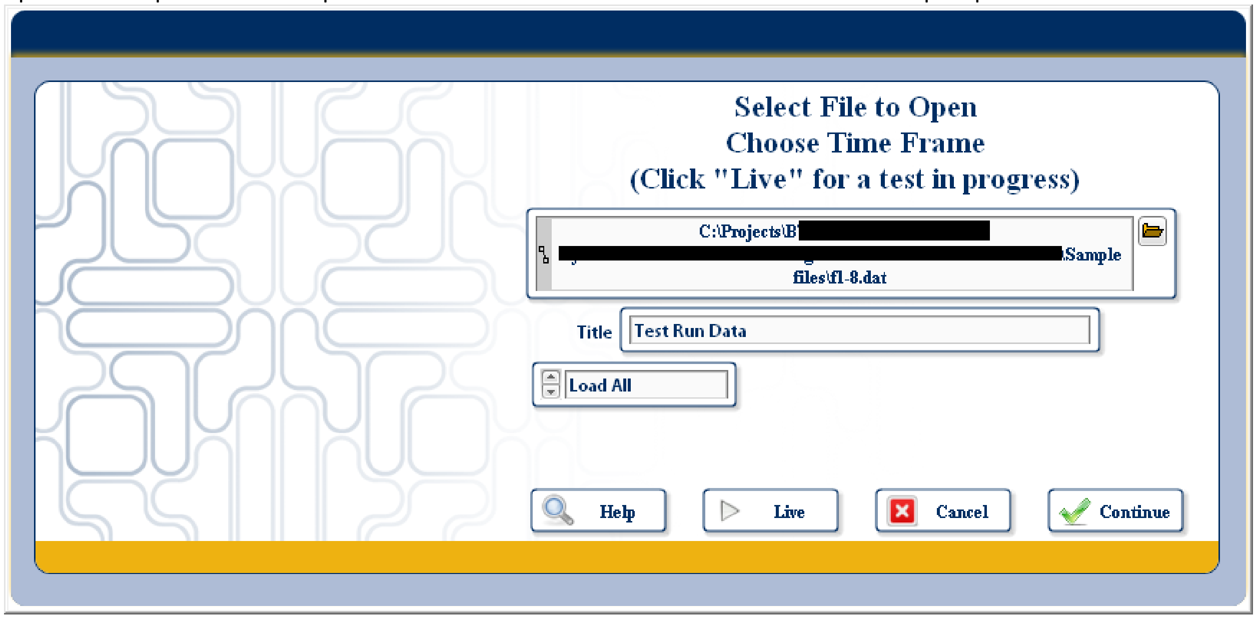
Cancel

Max	Min	Avg Delta
6.617	5.64507	6.41678
5.94405	5.12493	5.8075
2.69854	2.28399	2.60863
4.98509	4.39795	4.80911
4.24924	3.6934	4.1352
450.821	342.282	438.511
4.33842	3.78875	4.21258
4.64645	4.02499	4.50037
5.03046	4.20498	4.72015
0	0	0
0	0	0

Figure 3 Abort-able Progress Bar

But Sometimes We Do Not need to Analyze Everything

Using an abort-able progress bar, the application allowed the user to monitor how far through the file reading progress (see Figure 3) we have gotten and allowed them to stop the file reading before it completed. The Status bar allowed for that option but required both the producer and consumer be able to detect the user's stop request.



Select File to Open
Choose Time Frame
(Click "Live" for a test in progress)

C:\Projects\B[redacted] [redacted]Sample files\f1-8.dat

Title: Test Run Data

Load All

Figure 4 Live Monitoring of Test in Progress

Do It Once and Do Not Repeat the work

The prime motivator for developing the application was to be able to monitor a test that was in progress (see figure 4) to determine if it should continue or if adjustments to the test conditions are required. This meant that the same data file may be checked daily over a week or more. In order to avoid the need to repeat all of the work involve in parsing the data files each time the file was opened, the data files were compressed and indexed as they were read. The compressed data files contained only the numeric data and timestamps and greatly reduced the amount of time required to re-open a previously compressed file. The Indexing allowed the user to specify either a “Start and Stop” time or a “Number of Samples” range that would be displayed. The compression and indexing greatly increased the performance when reading a file that had previously been viewed.

Automatically Updated Indexing and Compressed Files

While files that contained information for test that had been completed could be quickly opened using the previously compacted and data and index files, the files for tests that were still active needed to be updated to reflect the new data. The application allowed for the compressed data and indexes to be updated to reflect the previously un-compressed data new data. This was accomplished by yet another producer consumer relationship. After presenting the compressed data, the application began formatting the new data section found in the tab-delimited log file. As each section was discovered, the index file and compressed data files were updated and the new data passed to the data reduction and display formatting functions mentioned earlier. The transition was seamless as the application transitioned from historical viewing to live data viewing.

Conclusion

Using LabVIEW to develop the Fretting Monitor application let us quickly and easily address each challenge that developed as the development moved along. NI shipping code provided a solid basis on which we built the file I/O routines that were required. A standard Produced-Consumer design pattern that harnessed the native multithreading built into LabVIEW allowed us to optimize performance. The flexible graphing functions of LabVIEW allowed us to display the data in a manner that was consistent with how the data is understood by the operators that ran the testing facility.

Contact Information

Benjamin A. Rayner info@DSAutomation.com
Data Science Automation Inc., 375 Valley Brook Road, Suite 106, McMurray, PA 15317-3370
www.DSAutomation.com