I’ve been building computer controlled things for more than 35 years. In any large project you always get to the point where strange things are happening and you don’t understand why its doing what its doing. In general my solution to this is to record everything, all sensor inputs, all drive outputs, every thing. While in concept this sound easy, it isn’t. You may have a sensor providing data at 1000Hz and the result of calculation being done every 5 seconds. You may also have hundreds of inputs and outputs with only a small number of them of interest for any one problem. In the past I’ve solved this problem by creating structures to hold logged data, writing them to a memory pool or file record and then writing special tools to decode them.  
  
This leads to several problems:

* Both the encoder and decoder code need to be modified when you change what you are logging.
* If a structure changed (say added a field) its hard to compare results from old/new records together.
* Its real easy to make a clerical error in either the encoding or decoding function.
* When you go back to old files its hard to know what format they are in.

I’ve recreated this code in various forms more times than I could count.  
If C++ supported proper class introspection one could probably solve this problem in a robust way where the logging would record the structure name and data layout as part of saving it. Unfortunately C++ does not presently support introspection. (Tools like Boost do, but that hammer is a bit big fo4r a small embedded device)  
  
The newest C++ standards do however support inline constructor arguments for C++ class/structure member variables, and these can be mutated into a pretty good approximation of c++ class introspection.  
  
For this years SparkFun AVC the very first piece of code I wrote was a robust logging infrastructure and the tools necessary to encode/decode, display it.,

This set of tools allows:

* Logging of arbitrary fixed size data structures .
* Looging different records at different rates
* Automatically stores the date/time revision for every source file into the data record at startup.
* Automatically stores the logged structures names and layout information into the data record.
* Stores the layout stream once and only once the first time any structure is used
* This information is communicated in the data record to the decoding tools.
* Mutiple decoding tools convert the Binary efficient record into:
  + JSON file.
  + CSV file with any easily specified subset of the data fields specified.
  + In Javascrip array of object inside a browser.

All of the code and code examples within this article can be found under my personal github page.  
<https://github.com/pbreed/DataLogginExample.git>  
  
While this article is an official netBurner publication the code I’m describing here is my personal tool. You are free to use and or modify it an any way, but it is not officially supported and not up to the level f production code quality.   
  
In a real application one would likely have lots of parameters, the car had 65 data fields.  
(An example of a Car data set in the carfiles subdiretory for this projects git hub) but without building you a robot to monitor I can’t really do that for an example. So what I’m going to do is instrument the netburner multiple connection TCp example. I’m going to generate several “Log” events, on connection, on error, on data reception, and on timetick.  
  
So what do I mean by log event, I mean we store a named structure in the data log.

So we will use the macros in the “datalogger.h” to build a structure..  
  
START\_INTRO\_OBJ(TCPConnectObj,"TCPConnect")

uint16\_element hisport {"hisport"};

uint16\_element thefdnum {"fdn"};

uint16\_element theaindx {"aindex"};

ipaddr\_element hisaddr {"hisaddr"};

END\_INTRO\_OBJ;

So we will then define one of the structures…  
TCPConnectObj TheTcpConnectObject;

And use it:

TheTcpConnectObject.hisport=client\_port;

TheTcpConnectObject.hisaddr=client\_ip;

TheTcpConnectObject.thefdnum=fda;

TheTcpConnectObject.theaindx=i;

TheTcpConnectObject.Log();

If you look in the example you will see that we made 4 different log objects and put them in various places in the code. They are used to record details of various events….

When you run this code and exercise it these events are stored in the big log.

You can then FTP to the log and get a binary file that holds all of these records….  
(Make sure you download in binary mode).

Once you have the bin file what can you do with it?

You can three things with it :

# Use the command line read.exe tool to convert to lists or csv separated files the the data in it.

Basic usages are:

Read -? //Get help

Read –L filename //Generate a list of all the fields stored in the file

Read –A –D –O Options //Read data, generate a CSV file and only include fields listed in the “options” file.  
  
  
A usage example:  
  
Run the example program and generate some data….(IE make and break some TCP connections send some text etc..)

ftp to the netburner and download the log file into mylog.bin

Now run Read –L mylog.bin >fieldlist.txt

Now open fieldlist.txt in an editor and remove any fields you don’t want reported….

Now run Read –A –D –O fieldlist.txt >MyData.csv  
  
The –A option tells it to output a new record every time it gets a new record. But sometimes you don’t want to do that. The tool allows fine control over when it outputs a new line of data. If you don’t use the –A but instead add emit on the line after the field you want to trigger a new record then only those record will be output….  
  
So in our example every time a tcp connection is closed we put a TCPClose object in the record….  
So suppose out options file has:  
  
TCPConnect.hisaddr  
TCPClose.why emit

The the system will only output two fields, hisadd and why it closed, but ONLY when it closes….  
  
  
These CSV files can be loaded into your favorite spread sheet and you can go over things in detail.

You can plot data, measure things, try to understand what your program is doing when the complexity gets beyond what normal mortals can handle.

## Use the Command line tool JsonConvert to make a JSON record file of the data…

Jsonconvert mylog.bin >mylog.json