The Booster Monitor Program at Fermilab: Data Retrieval and Analysis

Chandra Lane Jacobs

August 22, 2003

Abstract

In this paper, the function of the Booster Monitor program is described; especially the devices it can plot and why it is an important diagnostic tool to monitor the Booster. Also, data retrieval of Booster Monitor data from the data logger is explored, including ways to represent this data graphically in both Root and several web-based java applications available to the public.

1 Introduction

The Booster Monitor program was created in 2001 by Gerald Guglielmo in the Fermilab Computing Division. The current version is v3_0_2 and is based on Application Framework v2.3.45. The Booster Monitor application requests device snapshots from the Booster and statistically analyzes each snapshot by calculating the mean, standard deviation, and standard deviation of the standard deviation. These statistics are compared to a set of baseline statistics, which are calculated from the previous snapshot taken by the application. The first snapshot is thus its own baseline. The Booster Monitor also has the ability to compare these values to a set of reference values that the user can create.

When the incoming data differs from the baseline or reference set by more than the user-defined tolerance, the plot is flagged and a message is sent to the Data Logger. In this case, the baseline is reset to the deviant values. Therefore, the real purpose of the Booster Monitor program is to track changes over time in devices. Ultimately it is hoped that this record of component performance can be correlated to beam quality in the Booster. On the smaller scale, perhaps magnet or RF performance can be correlated to changes in the loss patterns.

2 The Booster Monitor program

2.1 Accessing the Booster Monitor Program

The Booster Monitor program must be run on the Beams Division network so that the Data Acquisition Engine (DAE) can be accessed. Therefore an account on Nova, the Beams Division server, must be obtained. However, to get in to Nova from outside the Beams Division also requires an Outland VAX account. Once these accounts are obtained, one can Telnet to the Beams Division as follows, not forgetting, of course, to add the particular host if using Unix:

- xhost + nova.fnal.gov
- telnet -x -F outland.fnal.gov
- telnet -x -F nova.fnal.gov

Once this is done, the actual Booster Monitor program can be found on the web, as a web-startable Java application from the Beams Division Application index page. That means Java is needed (found at http://java.sun.com/), but fortunately it already exists on Nova. Then do the following:

- Go to http://www-bd.fnal.gov/appix/index.html using Mozilla
- Click “from the Web” under “Launch Applications”
- Make sure the web start helper application is available to Mozilla. To do this, go to the Edit tab and click Preferences. Then go to Navigator → Helper Applications → New Type... and include the following settings:

  MIME Type: application/x-java-jnlp-file

  Suffixes: jnlp

  Application: usr/local/javaws/javaws/javaws %s
- Finally, expand the B * Booster tree and double click on Booster Monitor

2.2 Using the Booster Monitor Program

The Booster Monitor program consists of sixteen separate snapshot windows. A snapshot window can look at one particular device at a time and can be enlarged by double clicking on it. A snapshot can be reassigned to a different snapshot window by clicking on its corresponding “Plot” tab.

To use the program, a connection must first be made to the DAE. This is done by clicking the plug icon located at the bottom right hand corner. This is also where the data taking node can be changed. When the DAE is ready to take data, the plug will change colors and “plug-in.”

There are also three tabs located above the device snapshots representing the Monitor page, the Deviations page and the Editor page. The Monitor page plots the average device value, the Deviations page plots the standard deviation of the device from the average, and the Editor tab allows the user to modify the configuration.

Once a snapshot request is made, data begins filling the plots, automatically rescaling if data is out of bounds. The different colors on the plots represent the following:

- blue: baseline average value
- green: current average value
- red: sigma of the baseline average value multipled by the threshold value, added and subtracted form the baseline average
- yellow: reference set average value
- grey: sigma of the reference set average value multiplied by the threshold value, added and subtracted from the reference set average

If any point in the snapshot deviates more than the set threshold value (which is located on the bottom panel and can be set), the plot is flagged, and the “FLAG” color will change from green to red. The threshold value also controls when data is sent to the Data Logger. The baseline gives an idea of short-term stability.

To load a reference set, click the “Reload RefSet” button located at the bottom of the program to choose a reference set. A reference set is simply some pre-defined baseline statistics, whether it be particular device specifications, or just a set of saved baseline statistics from a previous running of the Booster Monitor program. Thus, a reference set can also be saved by clicking the "Save RefSet" button. Reference sets give an idea of long-term stability.

A set of devices can either be loaded from memory or created from scratch. To load a set of devices, hit the "ReadConfig" button on the bottom of the program. Expand the "test" folder then click on the "test" sub-folder to see the available device configurations.

Snapshot creation is done via the Editor tab and then by clicking the "Add Snapshot" button. The snapshots will begin filling with data, and the "S" located towards the bottom right of the program will increment. This number represents the number of snapshot callbacks received. The "A" adjacent to it represents the snapshot plot incrementor for accelerator device array data, which the program cannot yet support.

The standard configuration for a snapshot consists of the following:

- Name: DeviceName (ex. B:QL1)
- ClockName: BOOSTER_RESET_FOR_MINIBOONE
- ClockDelay: 0 (the delay from clock event trigger)
- Length: 200 (number of points to read from device)
- Rate: 7100 (snapshot sample rate in Hz)
- Offset: 0 (offset into start of device array)
- WinSize: 20 (number of points used in calculating statistics)
- FIFO: True
- DefaultAnalysis: True

The Action Entry window will eventually allow for different statistical packages, so it need not be used in the meantime.

Also in the Editor tab is the configuration name, the threshold sigma, and the maximum number of snapshots requested per job. These are all user defined. Changes made in the Editor tab will not take effect until the "Reconfig" button is hit on the main page.

2.3 Booster Monitor Output to the Data Logger

For each device, the Booster Monitor program sends an array of data to the Data Logger. The items in the array are:

1 == clock event number [29=1D in hexadecimal]

2 == data point index

3 == average

4 == standard deviation

5 == standard deviation of the standard deviation

6 == 1.0 if baseline, 0.0 if reference set discrepancy

7 == unique user id for the session

Additionally, a heartbeat is sent periodically to the Data Logger as a means to determine if a run is in progress. The heartbeat is also sent at the start and end of the run. When the heart beat is sent, [2] is set to -1 while [3], [4], [5] are set to zero.

Assuming no reference set is used, and assuming that the maximum number of points requested from the device is 200, the Data Logger sees the following: for the first snapshot requested, [2] simply counts up from 1 to 200 as it is essentially its own baseline. None of these values are out of spec, as no previous data is available to compare it to. After 200, only data that is out of range of the threshold value is sent to the Data Logger. So [2] therefore stops incrementing predictably, and also has some -1s interspersed to represent the heartbeat.

Also sent to the Data Logger is the time stamp of when the data was taken. The formula relating absolute time and points requested of a device is:

$$t_n = t_0 + \frac{n}{f}, \tag{1}$$

where t_0 is the event delay, n is the point number in the cycle and f is the snapshot sample rate. Given the set values of $t_0 = 0$, n = 200, and f = 100, the equation becomes:

$$t_n = \frac{200}{7100Hz} = 28.2ms. \tag{2}$$

Thus, of the 33ms it takes beam to get around Booster, the Booster Monitor program records 28.2 ms. In this fashion, the max n can be found:

$$33ms * 7100Hz = 234.4ms. \tag{3}$$

Thus, I do not know why more points are not typically requested.

2.4 Devices

An incomplete list of devices that the Booster Monitor program can look at are listed below. These are devices that the program has definitely run over. In general, the Booster Monitor program can run over any device capable of the snapshot method. Certain devices whose data come through IRMs only cannot be looked at by the Booster Monitor program. IRMs, or Internet Rack Monitors, are simply our newest data collectors.

Magnets

B:QL1	→	QL24	long quads
B:QS1	→	QS24	short quads
B:SL1	→	SL24	long skew quads
B:SS1	→	SS24	short skew quads
B:HS1	→	HS24	horizontal short ramped corrections
B:VL1	→	VL24	vertical long ramped corrections
B:SEXTL			long sextupole ramps
B:SEXTS			short sextupole ramps

LLRF Information	**BEX bump Information**	**Beam Loss Monitors**
B:RAG	B:IBEX2	B:BLML01 → B:BLML24
B:ROF	B:IBEX3	B:BLMS01 → B:BLMS24
B:APG	B:IBEX4	
B:RPOS	B:IBEX12	
B:FPERR	B:IBEX13	
B:RFSUM	B:IBEX14	

RF Gap Envelope	**Phase Detector Error (for RF)**
B:RF01GE → B:RF18GE	B:PD01E → B:PD18E

Miscellaneous

L:4VELO	Linac Velocity
E:LM870 → E:LM871	
E:TLI864	
E:TLI870	

2.5 Problems with the Booster Monitor Program

One problem is the Booster Monitor program's propensity for stalling. Sometimes it freezes while snapshots are being requested from it, and no data will be received. A possible explanation for this is that the program will wait for a job completion callback before making another request for that clock event. If the callback is lost, the program will not progress at all. Troubleshooting this consists of the following:

- In the terminal window from which the program was started type <ctrl> /
- ps -aux to find the process ID (pid)
- kill -3 <pid>
- Messages that hopefully shed some light on the problem will be outputted to the terminal window

A quick fix is to change nodes. The Booster Monitor program usually runs on dse07.fnal.gov, but if for whatever reason that node is getting a lot of traffic, switching to a more obscure node, like dse08 or dpe12, may allow the program to run without stalling.

3 Using the ACNET Console in Conjuction with the Booster Monitor Program

The Acnet Console is a program that looks at the entire accelerator process. It is useful to access as a means to get current readings from Booster devices and can therefore supplement the Booster Monitor program. To run it from a Kerberos account do:

- xhost + cns55.fnal.gov

- telnet -x -F outland.fnal.gov
- rsh cns55 x_display computername.fnal.gov:0.0 cnsrun mwm

A useful page to get Booster parameters is the B page, especially B2, B11 for devices and B43 for loss monitors.

4 Data Logger Retrieval

In this section, the retrieval of Booster Monitor data sent to the data logger is explained. Currently, there are two ways to access this data: through raw code run on the Beams Division that connects to the DAE, and through web-based programs made available by Beams Division programmers.

4.1 Web-Based Data Loggers

The web-based Data Loggers are accessed the same way the Booster Monitor program should be accessed–as a web-startable Java application. As outlined in Section 2.1, go to http://www-bd.fnal.gov/appix/index.html using Mozilla and click "from the Web" under "Launch Applications." Expand the "Diagnostics & Utility" tree and then expand the "Synoptic Display" tree.

Several Data Logger programs are found within: two versions of "Array Data Logger Plotter," the "Data Logger Plotter," and two versions of "Thinlets-based Data Logger Plotter." The figure beside the program names correspond to the type of application–a yellow triangle for requiring Java Web Start and a sheet of paper for a server-side application run without any plug-ins in a regular web-browser. The web-startable Array Data Logger Plotter is by far the most useful and least buggy. Entire arrays of data can be retrieved from it and saved as an ASCII file to be analyzed later (see Section 5). To get the Booster Monitor data, one must use the following format for the device name: G_XXX where XXX is the device name without B:, the usual device prefix. For example, the usual device B:QL1 becomes G_QL1. This is very important, as this is the special retrieval "code" for Booster Monitor data from the Data Loggers. Choosing the correct node is also vital–for retrieving data from the the Booster Monitor program, choose node BooMon. The user can also specify the array range–1 through 7 is recommended since that is the only useful part of the array the Booster Monitor stores. The other properties for the Array Data Logger Plotter, such as what time range to look at the device, are straightforward.

Also useful is the Data Logger Plotter. It cannot get full arrays, but rather elements of arrays. So instead of requesting G_QL1, one must request G_QL1[i] where i is some value of the array. Time and node must also be set, as usual. This data can be plotted by activating the "Plot" button, or looked at in ASCII or HTML format. The bad news is that to see the plot, an SVG plug-in must be obtained, which is available at http://www.adobe.com/support/downloads/main.html.

The Thinlets-based Data Logger Plotter is very buggy, at least using Mozilla, so it is not recommended. Also, all of the web-based Data Loggers tend to freeze up occasionally and are not stable, especially when viewed through Linux. They are more stable when viewed from a Windows machine. This instability is inherent in the code and needs to addressed.

4.2 Java-Based Data Loggers

4.2.1 Retrieving Java Source Code Through CVS

Many of the Data Loggers are also available as stand-alone java programs that sit on Nova. The easiest way to access these is through CVS, so first a CVS account should be obtained. CVS stands for Concurrent Versions System, a dominant open-source network-transparent version control system. It's basically a code repository that allows developers to access the latest code via the Internet. The Beams Division CVS directory can be browsed online at: http://www-bd.fnal.gov/cgi-acc/cvsweb.cgi/. To login to CVS in Unix, do:

- set CVSROOT :pserver:domain@nova.fnal.gov:/export/cvs/java
- cvs login

To get code through CVS do:

- cvs checkout filepath

where filepath is the path to the file in the CVS repository. For instance, the stand-alone Booster Monitor program is located at:

- gov/fnal/controls/applications/boosterMonitor/BoosterMonitor.java

Other code sits under the various folders contained in the "controls" folder.

When code is obtained via CVS, the folder structure is maintained. In other words, the gov/fnal/controls/filepath is going to show up in whatever directory it is requested to go to. To keep things simple, it is recommended to only checkout CVS code from one directory–otherwise, the folder structure will get messy.

4.2.2 Using Java-Based Data Loggers

Once the code is obtained, Java is needed to run it. Also the classpath must be set as follows:

- export CLASSPATH=.:/usr/local/dae:/usr/local/dae/jars/govjars.jar:/usr/local/dae/jars/hb14r.jar:/usr/local/dae/jars/jdom-b8.jar:/usr/local/dae/jars/servlet.jar

Note: some programs may require more jars, so the classpath will have to be modified accordingly. However, the classpath above should work with most programs. All additional jars should be found under the /usr/local/dae/jars directory, and must be added one at a time to the classpath. It is suggested to set this manually in the .bashrc file so that it is permanently in the user environment.

To compile and run a program, go to the directory where the code was checked out of (the directory where gov sits) and type "javac filepath.java" to convert and then "java filepath" (without the .java suffix) to run the program. For example, to run the Booster Monitor program from code, type:

- javac gov/fnal/controls/applications/boosterMonitor/BoosterMonitor.java
- java gov/fnal/controls/applications/boosterMonitor/BoosterMonitor

The most useful programs to get used to the Data Logger retrieval methods and structure are located in:

- gov/fnal/controls/examples/

Also, the entire Fermilab Java Control System library of packages, classes and everything else needed to get started with the API is located at:

- http://www-bd.fnal.gov/javadoc/build/api/

The most useful one is FTPExample1.java (see Appendix A) because it is easily modified to allow data retrieval from the Booster Monitor. It outputs to the terminal window the requested device (with a specified array value) from the Booster Monitor over a defined time period. The parameters must be modified through the code, as there is no user interface.

Another useful program is LoggerTest.java (see Appendix B), which I modified from /export/home/patrick, Jim Patrick's directory. It now can output an entire array of Booster Monitor data. Again, the parameters must be modified through the code, as there is no user interface.

Both of these files modified from their original versions can be found in my home directory on Nova:

- /export/home/cjacobs1/JavaStuff/loggertest/LoggerTest.java
- /export/home/cjacobs1/JavaStuff/gov/fnal/controls/examples/FTPExample1.java

One final interesting program is DataLoggerPlotterII.java, which can be found at:

- gov/fnal/controls/applications/DataLoggerPlotter/DataLoggerPlotterII.java

It has a user interface similar to the web-based Data Loggers, but it also has the capability to plot within the program. However, it only works for some devices and seems to only want to plot one thing at a time, even though it has plotted more than one in the past. The useful thing about this program is the x-axis can be changed to a device array value. Thus, it is possible to plot G_QL1[2] versus G_QL1[3].

5 Graphical Analysis of Data Logger Data in Root

From the Array Data Logger Plotter, an ASCII file of all the array values for a device (ex. G_QL1[1:7]) can be obtained. Such a file can be directly imported into Root and plotted, with some slight alterations–the first line of the ASCII file must be removed (the line with the column headers), as well as any invalid characters (like question marks). In other words, the file must be purely numbers. Doing this, one can plot a given member of the array against any other member, and thus can reproduce the snapshots in the

Booster Monitor program. Moreover, this method can be used to get plots of absolute time versus the Booster monitor data, which cannot be obtained by looking at the Booster Monitor program itself. Another interesting thing to do is plot absolute time versus the cycle time to see where data is missing, as well as to see when exactly the heartbeats were taken.

I have written two scripts in Root, which I include in Appendix C and D. The script in Appendix C reads in data from an ASCII file with eight columns and creates a basic graph capable of displaying three plots. It is suggested to use the Array Data Logger Plotter to get the array values 1-7, and save the data to an ASCII file, in the form filename.dat. Similarly, the script in Appendix D reads in an ASCII file as before, but displays the data as two separate graphs, with three plots per graph. It also displays the initial baseline value–that is, every time the [2] array goes from 0-199 incrementally. This way, values can be compared to the initial reading in a device. I have included sample output for both of these programs after the code in their respective appendices.

6 Conclusion

Many methods of data retrieval exist, but none are both easy to use and useful; simply retrieving the data is a big first step. This paper has just broached the topic of data analysis and graphically displaying data retrieved from the Data Logger. The tools outlined here should facilitate further study of the Booster Monitor program, retrieval of data thereof, as well as making sense of the retrieved data. It is hoped that further analysis of the Booster Monitor and its output is pursued, as it should prove to be a useful diagnostic tool of beam in the Booster.

7 Acknowledgments

The following people have helped make my summer productive and enjoyable:

- Timofei Boshekov
- Janet Conrad
- Linda Coney
- Gerald Guglielmo
- Pat Karns, Mike Vincent and the other MCR folk
- Robert Nelson and the other summer students working on MiniBooNE
- Jim Patrick

I would also like to thank Columbia University for offering me this REU, and the NSF for sponsoring it.

A FTPExample1.java

```
package gov.fnal.controls.examples;

import gov.fnal.controls.daq.acquire.*;
import gov.fnal.controls.acnet.*;
import gov.fnal.controls.daq.datasource.*;
import gov.fnal.controls.daq.items.*;
import gov.fnal.controls.daq.events.*;
import gov.fnal.controls.daq.context.*;
import gov.fnal.controls.daq.util.*;
import gov.fnal.controls.daq.snap.*;
import java.util.*;

/**
 * Basic example job to get obtain FTP data
 * Use the trivial GenericPlotDisposition which
 * just calls back to the client with the data.
 * This class just prints it to the terminal
 *
 * @author J. Patrick, modified by Chandra Jacobs
 */

public class FTPExample1 implements PlotCallback {

  private static String device = "G_SL1[2]";

  private DaqJob job;

/**
 * This method is part of the PlotCallback interface.
 * It will be called whenever a new set of readings is available
 */
  public void plotData(Object request, Date timestamp, CollectionContext
              context, int error, int numberPoints, long[]microsecs,
              short[] nanosecs, double[] values) {

/*
```

```
  Print out the readings
  The values have been scaled
*/
    if (error != 0) {
     String message = "ACNET_ERROR";
     try {
     message = AcnetError.errorCodeToText(error);
     } catch (Exception e) {
     System.out.println("Unable to decode acnet error code " + error);
     return;
     }

     System.out.println("Error " + error + ": " + message +
                    " on ftp data");
     return;
    }

System.out.println("Received ftp data at " + timestamp + " with " +
                    numberPoints + " points");
for (int i = 0; i < numberPoints; i++) {
System.out.println(microsecs[i]/1000 + " " + values[i]);
}

  }

/**
 * Constructor for FTPExample1
 *
 * @param devices    Array of device names to read
 * @param updateTime Time between readings in ms
 *
 **/
  public FTPExample1(String device) {
    initJob(device);
  }

/**
 * Set up the DaqJob to get data
```

```
 *
 * @param devices Array of device names to read
 * @param updateTime Time between readings in ms
 *
 */
  private void initJob(String device) {

// This creates a connection to dse07 with name "ReadingTry"

    DaqUser user = new DaqUser ("ReadingTry","dse07") ;

//
// We want to get data from the Accelerator (vs database, etc.)
//
    DataSource from = null;
    //An exception will be thrown on an illegal logger name
    try {
from = new DataLoggerSource("BooMon");
    }   catch (AcnetException e) {
System.out.println("Error accessing data logger " + e.getMessage());
System.exit(-1);
    }
//
// The Dispostion will be a GenericPlotDisposition
// declare this class as the callback class
//
    GenericPlotDisposition to = new GenericPlotDisposition();
    to.establishCallback(this);
//
// The Item will be a FTPRequest
//
    FTPScope fs = new FTPScope(20.0); // 20 Hz

// Uncomment to restrict the duration, otherwise goes on indefinitely
//    fs.setDuration(2.0);

    DataLoggerPlotRequest item = new DataLoggerPlotRequest(device, to);
```

```
    Date now = new Date();
    //go back 10 hours
    long startms = now.getTime() - 10*60*60*1000;
    Date start = new Date(startms);
    DataEvent event = new DeltaTimeEvent(start, now);

    DaqJobControl control = new DaqJobControl();
//
// Create the DaqJob with the above parameters
// Data acquistion does not start until job.start() is called
//
    job = new DaqJob(from, to, item, event, user, control);

}

/**
 * Start the DaqJob that was created in initJob
 *
 */
  public void startJob() {

    try {
      job.start();
      job.waitForSetup();
    } catch (Exception e) {
      System.out.println("whoops, job.start caught: " + e.getMessage());
    }
  }

/**
 *  Main program
 *
 *  @param argv Arguments, ignored in this example
 *
 */

  public static void main(String[] argv) {
```

```
// java FTPExample1

    FTPExample1 test = new FTPExample1(device);
    test.startJob();
  }
}
```

B LoggerTest.java

```
/*
  Basic example to fetch data from a datalogger.
  This creates a "DaqJob" that fetches the
  data and calls back to the receiveData method below.
  It must be run on the Beams Division network.
*/

import gov.fnal.controls.daq.acquire.*;
import gov.fnal.controls.daq.datasource.*;
import gov.fnal.controls.daq.items.*;
import gov.fnal.controls.daq.callback.*;
import gov.fnal.controls.daq.context.*;
import gov.fnal.controls.daq.events.*;
import gov.fnal.controls.daq.util.*;
import gov.fnal.controls.acnet.*;
import java.util.*;

public class LoggerTest {

  private DaqJob job;
  private long lastReading = 0;
  private int ntotal = 0;

    //Inner Class to receive data

  class Receiver implements PlotCallback {

/*
  This method will be called back as blocks
  of data are received from the data logger.
  This will generally be called multiple times
  as the logger data is broken up into
  manageble sized blocks for large requests
*/

    public void plotData(
```

```
      Object request,
      Date timestamp,
      CollectionContext context,
      int error,
      int numberPoints,
      long[] microSecs,
      short[] nanoSecs,
      double[] values) {

    System.out.println(numberPoints + " readings received");
    ntotal=ntotal+numberPoints;

    if (error != 0) {
      String message = "ACNET_ERROR";
      try {
              message = AcnetError.errorCodeToText(error);
      } catch (Exception e) {
              System.out.println("Unable to decode acnet error code "
                  + error);
              return;
      }
      System.out.println("Error " + error + ": " + message +
                  " on data");
      return;
    }

    if (numberPoints <= 0) return;
    for (int i = 0; i < numberPoints; i++) {
      Date d = new Date(microSecs[i]/1000);
      System.out.println("date: " + d + " " + "microSec: " +
                  microSecs[i] + " data: " + values[i]);
    }
  }
}

public LoggerTest(String logger, String device) {

  DaqUser user = new DaqUser ("ReadingTry","dse08") ;
```

```
    //Declare callback to receive data
    GenericPlotDisposition callback = new GenericPlotDisposition();
    callback.establishCallback(new Receiver());
    DataLoggerPlotRequest item =
    new DataLoggerPlotRequest(device, callback);

     DataSource from = null;
     try {
       from = new DataLoggerSource(logger);
     } catch (AcnetException e) {
       System.out.println("Error creating data logger source "
                         + e.getMessage());
       System.exit(-1);
     }

    GenericPlotDisposition to = new GenericPlotDisposition();
    to.establishCallback(new Receiver());

    //Establish time period for which data is desired
    long now = (new Date()).getTime();

    long end = now;
    //gets data from 12 hours ago
    long start = now - 1*24 * 60 * 60 * 1000;
    Date startDate = new Date(start);
    Date endDate = new Date(end);
    DeltaTimeEvent event = new DeltaTimeEvent(startDate, endDate);
    System.out.println("Start at: " + startDate +
                       " End at: " + endDate);

    DaqJobControl control = new DaqJobControl();

    //Create the job to fetch the data
    job = new DaqJob(from, to, item, event, user, control);

}
```

```
/*
  This method starts the "job" that fetches the data
  It waits for it to complete, then prints the
  number of points seen by the above callback
  and the time spend waiting
*/
  public void startMonitor() {

    try {
      Date d1 = new Date();
      job.start();
      job.waitForSetup();
      job.waitForCompletion();
      Date d2 = new Date();
      long diff = d2.getTime() - d1.getTime();
      System.out.println("Time taken was " + diff);
      System.out.println("Number of readings was " + ntotal);
      //System.exit(0);
    } catch (Exception e) {
      System.out.println("whoops, job.start caught: " + e.getMessage());
    }
  }

    //Main method. Create and start "job" to fetch data
  public static void main(String[] argv) {

      for (int i = 1; i < 8; i++) {

      String device = "G_QL1["+ i + "]";
String logger = "BooMon";

if (argv.length > 0) {
    logger = argv[0];
}
if (argv.length > 1) {
    device = argv[1];
}
```

```
    System.out.println("Reading device: " + device +
                       " from logger " + logger);

    LoggerTest test = new LoggerTest(logger, device);
    test.startMonitor();
    }

    System.exit(0);

  }
}
```

C Simple Plot Example in Root

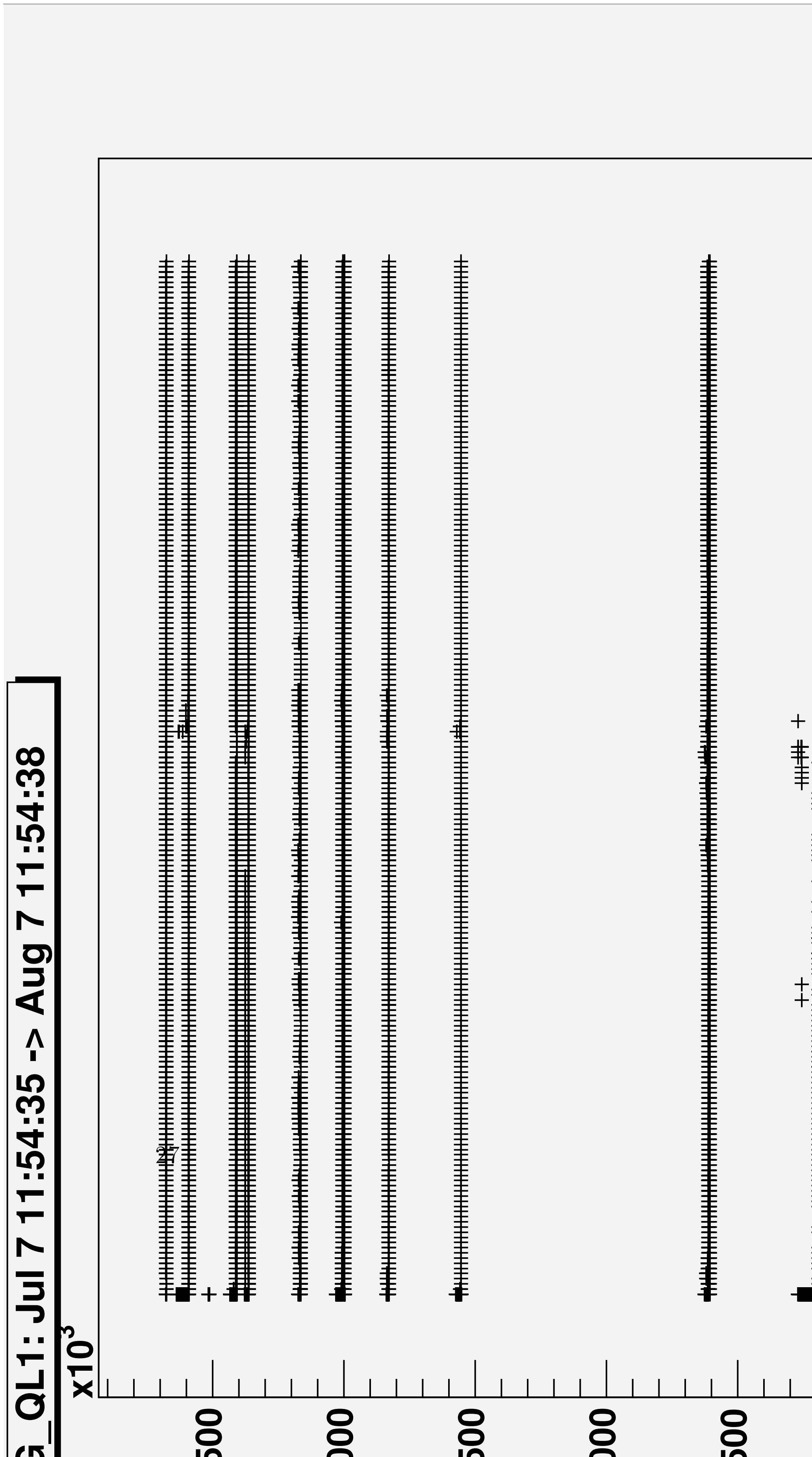
C.1 Simple Plot Code

```
{
//   Convert.c
//
//   This script reads in data from an ASCII file with 8 columns
//   and creates a basic graph.
//   It is suggested to use the Array Data Logger Plotter to get the data
//   (get the array values 1-7), and save it as an ASCII file, xxx.dat.
//   Author: Chandra Jacobs

gROOT->Reset();

#include "Riostream.h"
#include "TGraph.h"

// gets number of lines from the file
   ifstream in;

   Float_t col0, col1, col2, col3, col4, col5, col6, col7;

// the ASCII file to be opened;
// be sure to delete the column headers
   in.open("QL1_Jul7-115435_Aug7-115438.dat");

   Int_t nlines = 0;

   while (1) {
      in >> col0 >> col1 >> col2 >> col3 >> col4 >> col5 >> col6 >> col7;
      if (!in.good()) break;

      nlines++;
   }

   nlines = nlines + 5;
```

```
//gets data
   Float_t Arg0[nlines], Arg1[nlines], Arg2[nlines], Arg3[nlines],
           Arg4[nlines], Arg5[nlines], Arg6[nlines], Arg7[nlines];
   in.close();

   ifstream in;

// QL1 is an ASCII file with 8 columns of data--
// time, plus the array sent from the BooMon.
   in.open("QL1_Jul7-115435_Aug7-115438.dat");

   Int_t lines = 0;

   while (1) {
      in >> col0 >> col1 >> col2 >> col3 >> col4 >> col5 >> col6 >> col7;
      if (!in.good()) break;

      Arg0[lines]=col0;
      Arg1[lines]=col1;
      Arg2[lines]=col2;
      Arg3[lines]=col3;
      Arg4[lines]=col4;
      Arg5[lines]=col5;
      Arg6[lines]=col6;
      Arg7[lines]=col7;

      lines++;
   }

   c1 = new TCanvas ("c1","BooMon QL1 Data from DataLogger"
                     ,200,10,600,400);

// comments here are for more plots being drawn on the same graph--
// only plotting cycle time versus absolute time with comments.
   gr1 = new TGraph(lines, Arg2, Arg0);
/* gr2 = new TGraph(lines, Arg4, Arg0); */
/* gr3 = new TGraph(lines, Arg5, Arg0); */
```

```
   gr1->SetMarkerColor(1);
   gr1->SetMarkerStyle(2);
   gr1->SetMarkerSize(.8);
   gr1->SetTitle("G_QL1: Jul 7 11:54:35 -> Aug 7 11:54:38");
   gr1->Draw("AP");

/*    gr2->SetMarkerColor(2); */
/*    gr2->SetMarkerStyle(3); */
/*    gr2->SetMarkerSize(.8); */
/*    gr2->Draw("P"); */

/*    gr3->SetMarkerColor(4); */
/*    gr3->SetMarkerStyle(5); */
/*    gr3->SetMarkerSize(.8); */
/*    gr3->Draw("P"); */

/*    leg = new TLegend(.7, .15, .85, .3); */
/*    leg->AddEntry(gr1, "QL1[3]", "p"); */
/*    leg->AddEntry(gr2, "QL1[4]", "p"); */
/*    leg->AddEntry(gr3, "QL1[5]", "p"); */
/*    leg->Draw(); */

   c1->Update();
   gr1->GetHistogram()->SetXTitle("Cycle Time: n = 7100Hz * time");
   gr1->GetHistogram()->SetYTitle("Absolute Time (s)");
   gr1->GetYaxis()->SetTitleOffset(1.4);
   c1->Modified();

   in.close();
}
```

C.2 Simple Plot Output



D Complicated Plot Example in Root

D.1 Complicated Plot Code

```
{
//   This script reads in data from an ASCII file with 8 columns
//   and draws 2 different graphs, arranged on a canvas,
//   with 3 plots on one of the graphs, and 3 on the other.
//   It is suggested to use Array Data Logger Plotter to get the data
//   (get the array values 1-7), and save it as an ASCII file, xxx.dat.
//   Author: Chandra Jacobs


gROOT->Reset();

#include "Riostream.h"
#include "TGraph.h"

// gets the number of lines in the file
ifstream in;

   Float_t col0, col1, col2, col3, col4, col5, col6, col7;

// the ASCII file to be opened;
// be sure to delete the column headers
   in.open("QL1_Jul7-115435_Aug7-115438.dat");
   Int_t nlines = 0;

   while (1) {
      in >> col0 >> col1 >> col2 >> col3 >> col4 >> col5 >> col6 >> col7;
      if (!in.good()) break;

      nlines++;
   }
   nlines=nlines + 1;
   printf("nlines =  %d\n", nlines);
// gets data
   Float_t Arg0[nlines], Arg1[nlines], Arg2[nlines], Arg3[nlines],
```

```
          Arg4[nlines], Arg5[nlines], Arg6[nlines], Arg7[nlines];
   in.close();

   ifstream in;

// QL1 is an ASCII file with 8 columns of data--
// time, plus the array sent from the BooMon.
   in.open("QL1_Jul7-115435_Aug7-115438.dat");

   Int_t lines = 0;

   while (1) {
      in >> col0 >> col1 >> col2 >> col3 >> col4 >> col5 >> col6 >> col7;
      if (!in.good()) break;

      Arg0[lines]=col0;
      Arg1[lines]=col1;
      Arg2[lines]=col2;
      Arg3[lines]=col3;
      Arg4[lines]=col4;
      Arg5[lines]=col5;
      Arg6[lines]=col6;
      Arg7[lines]=col7;

      lines++;
   }

// Make an array of just baseline values
// (ie, the count up from 0-199 in [2])
// comment this out if the program is taking too long to run!
   Int_t index, arraycounter =  0;
   Int_t counter = 0;
   Float_t baseline2[nlines], baseline3[nlines];

   while (index < nlines) {

      if (Arg2[index] == Arg2[index+1] - 1  && index < nlines
          && Arg2[index] != -1 ) {
```

```
 //printf("index =  %d   counter = %d   [2] = %d\n",
                 index, counter, Arg2[index]);
 index++;
 counter++;

 if (counter == 199 && Arg2[index] == 199) {
   // printf("NEWindex =  %d   counter = %d   [2] = %d\n",
   index, counter, Arg2[index]);
    counter = 0;

    while (counter < 200) {
       baseline2[arraycounter] = Arg2[index - 199];
       baseline3[arraycounter] = Arg3[index - 199];

       printf("[2] = %d   baseline:%d   arraycounter:%d\n",
                      index, baseline2[arraycounter], arraycounter );
               arraycounter++;
       index++;
       counter++;
    }
    printf("[2]! = %d   baseline:%d\n", index, baseline2[arraycounter] );
    counter = 0;
 }
      }
      else  if (Arg2[index] == -1) {
         printf("FLAG:-1 index =  %d   counter = %d   [2] = %d\n",
index, counter, Arg2[index]);
 index++;
         counter = 0;
      }
      else {
         printf("FLAG: 1 index =  %d   counter = %d   [2] = %d\n",
index, counter, Arg2[index]);
         index++;
         counter = 0;
     }
}
```

```
// creates canvas
   c1 = new TCanvas ("Device Analysis",
"QL1 Data from DataLogger: Aug 6 11:02:23 -> Aug 7 00:35:31"
         ,10,40,800,600);
   c1->Range(0,0,25,18);

   TPaveLabel *pl = new TPaveLabel(1,16.3,24,17.5,
                    "QL1 Data from Datalogger","br");
   pl->SetFillColor(18);
   pl->SetTextFont(32);
   pl->SetTextColor(49);
   pl->Draw();

   TText *t = new TText();
   t->SetTextFont(32);
   t->SetTextColor(1);
   t->SetTextSize(0.03);
   t->SetTextAlign(12);
   t->DrawText(3.1,15.5,"Cycle time versus statistics");
   t->DrawText(14.,15.5,"Absolute time versus statistics");

// creates pads
   pad1 = new TPad("pad1","This is pad1",0.02,0.02,0.48,0.83);
   pad2 = new TPad("pad2","This is pad2",0.52,0.02,0.98,0.83);

   pad1->Draw();
   pad2->Draw();

   pad1->cd();
   pad1->Range(-0.255174,-19.25,2.29657,-6.75);

// adds graphs
   gr1 = new TGraph(lines, Arg2, Arg3);
   gr2 = new TGraph(lines, Arg2, Arg4);
   gr3 = new TGraph(lines, Arg2, Arg5);

   gr1->SetMarkerColor(1);
   gr1->SetMarkerStyle(2);
```

```
   gr1->SetMarkerSize(.8);
   gr1->SetTitle("Jul 7 11:54:35 -> Aug 7 11:54:38");
   gr1->Draw("AP");

   gr2->SetMarkerColor(2);
   gr2->SetMarkerStyle(3);
   gr2->SetMarkerSize(.8);
   gr2->Draw("P");

   gr3->SetMarkerColor(4);
   gr3->SetMarkerStyle(5);
   gr3->SetMarkerSize(.8);
   gr3->Draw("P");

//reference value
   gr7 = new TGraph(lines, baseline2, baseline3);
   gr7->SetMarkerColor(3);
   gr7->SetMarkerStyle(6);
   gr7->SetMarkerSize(.6);
   gr7->Draw("P");

   leg = new TLegend(.15, .15, .5, .3);
   leg->AddEntry(gr1, "QL1[3]:Average reading", "p");
   leg->AddEntry(gr2, "QL1[4]:Sigma", "p");
   leg->AddEntry(gr3, "QL1[5]:Sigma of the Sigma", "p");
   leg->AddEntry(gr7, "QL1[3]:Reference Value", "p");
   leg->Draw();

   gr1->GetHistogram()->SetXTitle("Cycle Time: n = 7100Hz * time");

   pad2->cd();
   pad2->Range(-0.43642,-23.75,3.92778,-6.25);

   gr4 = new TGraph(lines, Arg0, Arg3);
   gr5 = new TGraph(lines, Arg0, Arg4);
   gr6 = new TGraph(lines, Arg0, Arg5);

   gr4->SetMarkerColor(1);
```

```
    gr4->SetMarkerStyle(2);
    gr4->SetMarkerSize(.8);
    gr4->SetTitle("Jul 7 11:54:35 -> Aug 7 11:54:38");
    gr4->Draw("AP");

    gr5->SetMarkerColor(2);
    gr5->SetMarkerStyle(3);
    gr5->SetMarkerSize(.8);
    gr5->Draw("P");

    gr6->SetMarkerColor(4);
    gr6->SetMarkerStyle(5);
    gr6->SetMarkerSize(.8);
    gr6->Draw("P");

    leg = new TLegend(.50, .15, .85, .30 );
    leg->AddEntry(gr4, "QL1[3]:Average reading", "p");
    leg->AddEntry(gr5, "QL1[4]:Sigma", "p");
    leg->AddEntry(gr6, "QL1[5]:Sigma of the Sigma", "p");
    leg->Draw();

    gr4->GetHistogram()->SetXTitle("Time (s)");
    gr4->GetXaxis()->SetNdivisions(606);

    c1->Modified();
    c1->Update();

    in.close();
}
```

D.2 Complicated Plot Output