An Introduction to Plant Monitoring Through the EPICS Control System*

J.A. Carwardine
Advanced Photon Source, Argonne National Laboratory
9700 South Cass Avenue, Argonne, Illinois 60439 USA

Abstract

The Experimental Physics and Industrial Control System (EPICS) environment [1] provides the framework for monitoring any equipment connected to it. Various tools offer engineers and scientists the opportunity to easily create high-level monitoring applications without having to rely on expert programmers to develop custom programs. This paper is aimed at the first-time or casual user, providing essential information for using several of the tools. Examples are taken from applications in regular use at the Advanced Photon Source (APS).

I. INTRODUCTION

Hardware interface to the EPICS control system is via local input-output controllers (IOCs), implemented at APS using VME-based 68,000-series microprocessors. Each IOC contains a database which references equipment connected to that IOC. Database records are referred to as *process variables* (PV), each one having a unique name. In general, a process variable record is associated with some hardware interface (I/O) connected to external equipment and contains the value associated with that I/O (e.g. the output value of a digital-analog converter, the input value from an analog-digital converter, the state of a binary input, etc.). Process variables can be analog, digital (two or more states), or entire waveforms, depending on the I/O configuration.

Access to process variable data (both input and output) is performed by "channel access," used by the EPICS top-level applications and is also available via C libraries to applications which need access to process variables. There are also utilities for command-line access (*caget* and *caput*). For example, the command *caget P:BM:DacAI* returns the value of the process variable *P:BM:DacAI*.

Each process variable record has a number of fields. In addition to the value field (the default), other fields provide information about the I/O; for example, the maximum and minimum values, the precision, error conditions, etc.

Operator interface to the control system is provided by top-level EPICS applications such as *medm* and the *alarm handler*. These have been supplemented by the SDDS toolkit [2] which can perform sophisticated data collection and post-processing functions.

II. FINDING PROCESS VARIABLE NAMES

The simplest way to find a process variable name is through a medm screen already containing the quantity of interest. By pointing to the quantity with the computer mouse and pushing the appropriate mouse button, the process variable name is displayed on the screen.

A recent addition to the APS control system is an application which allows wildcard searches for process variable names and also identifies the name of the IOC and its physical location in the facility. This application was written using the Tk/Tcl scripting language [3].

III. ALARM HANDLER PRIMER

The EPICS alarm handler [4] performs background monitoring of specified process variables and alerts the user to any change in state (e.g., if a power supply trips).

In order to use the alarm handler, a configuration file must be created. This lists the process variables to be monitored and how they are to be displayed to the user. The command *alh ex1.alhConfig &* starts the alarm handler with the configuration file *ex1.alhConfig* and produces a window such as the one shown in Figure 1.

Figure 1: *Alarm Handler* Normal Screen

The color of the center part of the window indicates the overall condition of the process variables being monitored. If there are no alarms, the center is grey. If there are alarms present, the center may be white, yellow, or red (depending on the highest *severity* alarm present). If a new alarm occurs, the center of the window starts flashing (the color of the new alarm), and the computer starts beeping. Clicking the center part of the window with the mouse brings up a detailed screen (Figure 2).

---

*Work supported by U.S. Department of Energy, Office of Basic Energy Sciences, under Contract No. W-31-109-ENG-38.

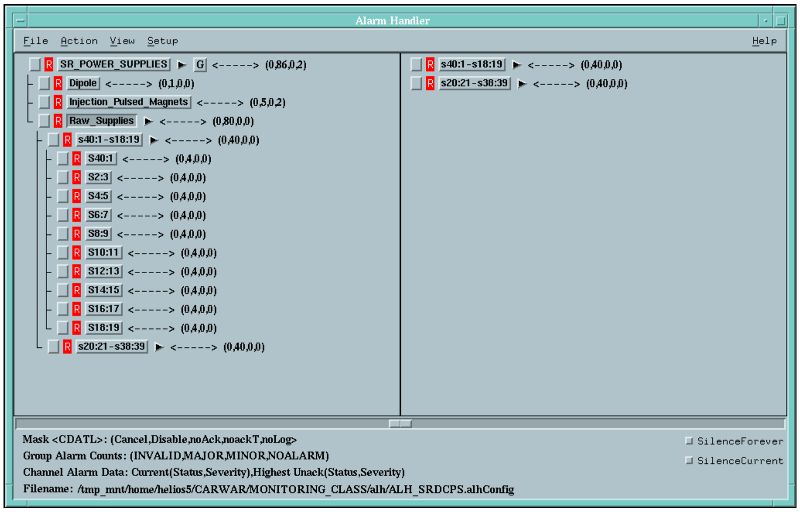

Figure 2: *Alarm Handler* Detailed Screen

This example shows the alarm handler detail screen for power supplies in the APS storage ring. Alarms are indicated by a small box of the appropriate color to the left of the process variable name. Alarms are acknowledged by clicking the box with the mouse (this does not reset the equipment). Unless the alarm has gone away, the alarm box remains, but the alarm handler stops beeping. When the alarm state goes away or is reset, the alarm box disappears from the detail screen.

A. Alarm Handler Features and Limitations

a) The alarm handler works with digital process variables. It therefore naturally lends itself to monitoring the state of a binary input (e.g., the status of a power supply). Analog parameters can be range-checked by setting up high and low limits in associated fields of the process variable, allowing the alarm handler, for example, to monitor that a power supply is not overheating.

b) Any one of three alarm severities (major, minor, invalid) can be associated with a process variable. ‘Invalid’ alarms are normally reserved for communications problems (e.g., with the IOC). Most other problems are configured as major alarms by default.

c) Process variables can be grouped for convenient display on the detail screen. This was shown in Figure 2, where power supplies were grouped by storage ring sector. The groups are shown on the left side of the screen in a tree-like structure. Highlighting a group shows the detail within that group on the right of the screen. Any number of levels can be created.

d) Buttons can be added to the alarm handler detail screen to allow the user to perform a specified action (e.g., pull up a detail control screen). It is also possible to configure the alarm handler to issue a command automatically if a given event occurs.

e) Other buttons can be added which display a specified text message when pushed. This can be used to offer guidance to the operator, for example.

B. Creating Alarm Handler Configuration Files

These can be created with any text editor (e.g., emacs). The following is the complete configuration file for monitoring eight process variables:

GROUP NULL RAW_SUPPLIES
GROUP RAW_SUPPLIES Sectors40&1
CHANNEL Sectors40&1 S40:1:R1:StatusCALC
CHANNEL Sectors40&1 S40:1:R2:StatusCALC
CHANNEL Sectors40&1 S40:1:R3:StatusCALC
CHANNEL Sectors40&1 S40:1:R4:StatusCALC
GROUP RAW_SUPPLIES Sectors2&3
CHANNEL Sectors2&3 S2:3:R1:StatusCALC
CHANNEL Sectors2&3 S2:3:R2:StatusCALC
CHANNEL Sectors2&3 S2:3:R3:StatusCALC
CHANNEL Sectors2&3 S2:3:R4:StatusCALC

The first command *GROUP* defines a new group called *RAW_SUPPLIES*. Contained within this group are two child groups called "*Sectors40&1*" and "*Sectors2&3*", defined in the two subsequent *GROUP* commands. The *CHANNEL* commands identify the process variables for the alarm handler to monitor. The formal usage of the above commands is:

GROUP <parent_name> <child_name>
CHANNEL <group_name> <PV_name>

Three additional commands are used to configure the auto-run commands and buttons which issue a specific UNIX command or display text messages. Their usages are as follows:

$ALARMCOMMAND <alarm> <unix command string>

Where *<alarm>* defines the event to trigger the issuing of the UNIX command string, e.g. UP_MAJOR, DOWN_ANY, would issue the command when a major alarm occurs or when any alarm goes away, respectively.

$COMMAND <unix command string>

This creates a button on the alarm handler detail screen next to the group or channel defined immediately prior to this statement. Pushing the button issues the UNIX command string.

$GUIDANCE <one or more lines of text> $END

This creates a button on the alarm handler detail screen next to the group or channel defined immediately prior to this statement. Pushing the button displays the text in a window.

IV. MEDM SCREENS

The EPICS application *medm* provides a graphical interface to the control system. Medm screens are the cornerstone of the operator interface to the control system and to the accelerator. Custom screens can easily be created graphically by running medm in edit mode (type *medm -edit &* from within an xterm window). The program has a similar look and feel to drawing packages for the PC or MAC and includes predefined graphical objects (such as slider bars, readbacks, etc.) for connecting to process variables. An example of a simple user-created medm screen is shown in Figure 3.

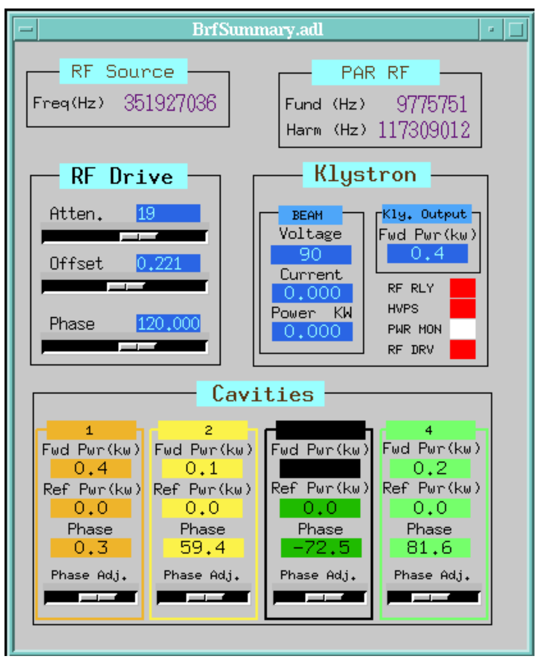

Figure 3: Example *medm* screen

V. SDDS PRIMER

The Self-Describing Data Set (SDDS) data format is widely used at APS [3]. Each data element in the file is given a unique ASCII name by the user which is subsequently used to access the data. An ASCII file header contains information about the data type and its location in the file. The user only need know the name of the data element to access it. Data can be integer values, floating point values or ASCII strings. Whilst the header is always ASCII, the data itself can be either in ASCII or binary format with no change in functionality.

There is a continuously expanding list of both general-purpose and custom programs which take advantage of the flexibility offered by the SDDS data format. For example, it is possible to perform complex mathematics, take statistics, do Fourier transforms, digital filtering, etc., and generate publication-quality plots using data contained in an SDDS file. There are also a number of EPICS-specific applications which collect data, automatically perform experiments, do closed-loop control, etc., all using the SDDS format. Tools also exist to convert data to/from other file formats, e.g., spreadsheets, oscilloscope data, proprietary mathematics packages, etc.

The SDDS tools are all UNIX command-line driven. This allows considerably more power to be incorporated into the programs than would otherwise be possible and allows the creation of custom data collection and analysis tools by combining the tools inside UNIX shell scripts. In some cases, visual interfaces have been created using Tk/Tcl to act as graphical front ends to the command-line programs.

A. The Minimum SDDS File

The following is a minimum SDDS file containing two parameters and two columns of data:

```
SDDS1
&parameter name="pi", type=double, &end
&parameter name="date", type=string, &end
&column name="value", type=double, &end
&column name="square", type=double, &end
&data mode=ascii, &end
3.1415928
3rd September 1995
7
0.0     0.0
1.0     1.0
2.0     4.0
3.0     9.0
4.0    16.0
5.0    25.0
6.0    36.0
```

The order of the data follows the order of the parameter and column definitions in the header. The number of rows in each column is indicated immediately before the start of the column data (in this case 7). The data type can be: *short, long, float, double, string*. A more comprehensive header might contain a text description of the file and many parameters and columns, each with a text description and units (e.g., Amps, mS, etc.).

B. Viewing SDDS Data

Clearly, small ASCII files like the example above could be viewed using either an editor or the UNIX command *more.* But in general, this really isn't a useful way of viewing the data. There are two parts to viewing a file: finding out what is in a file and looking at the data itself.

The command s*ddsquery* allows the user to find out what is in the file. For example, if the above file were called *ex1.sdds*, then the command *sddsquery -column ex1.sdds* would show the names of all the columns in the file. Parameter names can be found by replacing *column* with *parameter.*

If more detailed information is required (say the data type), then the command *sddsquery ex1.sdds* would print out all of the information from the file header.

Data from the file can be accessed using the command *sddsprintout*. This allows printouts of data from a file for specified columns or parameters. For example, the command *sddsprintout ex1.sdds -column=value* will print all rows from the column *value* in the above file as follows:

```
Printout for SDDS file ex1.sdds
     value
---------------
  0.000000e+00
  1.000000e+00
  2.000000e+00
  3.000000e+00
  4.000000e+00
  5.000000e+00
  6.000000e+00
```

Wildcards can be used to select parameters or columns. For example, to view all of the column data, type *sddsprintout ex1.sdds -col='*'*.

C. Plotting SDDS Data

By far the most frequently used program in the SDDS toolkit is *sddsplot*. This is a powerful publication-quality plotting program for SDDS data files.

To plot the two columns of data in the above example file, type *sddsplot -col=value,square ex1.sdds*. Figure 4 shows the window which is created.

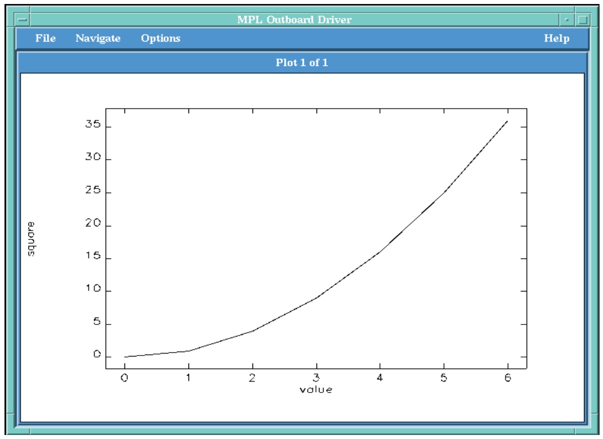

Figure 4: Window generated by *sddsplot*

The plot format can be modified in many different ways. For example, to plot the data points without joining the dots type *sddsplot -col=value,square -graphic=symbol ex2.sdds*.

More than one column can be plotted at the same time. Suppose a column *root* were added to the file *ex1.sdds*. Then both the *square* and *root* columns could be plotted together by using the command *sddsplot -col-value,square -col=value,root ex2a.sdds*. The same result could be obtained by combining the *square* and *root* column names into a single *-column* option with the command *sddsplot -col=value,'(square,root)' ex2.sdds* (note the use of single quotes and braces). Alternatively, the command *sddsplot -col=value,'*' ex2.sdds* would plot all the columns against *value*. There are many other ways in which the plots can be formatted. Examples can be found in [2] and [4]. Plots can also be formatted for inclusion in documents without resorting to an xwindows screen grab.

The following is a selection of the many options available in *sddsplot*:

-graphic allows the user to chose from connected lines, unconnected symbols, or unconnected dots.

-scale allows the user to specify the x- and y-scales.

-separate puts multiple plot requests on separate pages, e.g. *-col=value,'(square,root)' -sep* will produce two separate plots, one being *value* vs. *square*, the other *value* vs. *root*.

-samescale is used with *-separate* option, and forces all plots to have the same x- and y-scales.

-layout defines the layout of plot frames on each page, for example, plots could be put in two rows of two plots per page.

-topline adds a title to the plot.

-xlabel allows labelling of the x-axis with a supplied string, rather than using column names from the file. A similar *-ylabel* option exists for the y-axis.

-filenames adds the names of the data files to each plot.

-date timestamps the plot with the time and date.

D. A Note about SDDS Command-line Formats

All SDDS commands have a generic command-line format. The minimum command line would consist of the command plus a filename, although most commands require at least one option as well. All options consist of the "-" character immediately followed by a keyword (e.g. *-column, -ascii*, etc.).

Many keywords also require parameters, in which case the keyword is immediately followed by the "=" character plus the parameters separated by commas (e.g. *-column=value,square*). No spaces may appear within the expression.

All keywords can be abbreviated to the minimum unique character string (e.g., *-col* is recognized as being *-column*).

Any command-line argument not starting with the "-" character is assumed to be a filename. Many commands which modify an existing file can be given either one or two filenames. The first filename given is always assumed to be the input file and the second (if supplied) to be the name of the output file to be created. If only one is supplied then the input file is replaced.

In most cases, the order of the arguments does not matter. For example, *sddsquery -col ex2.sdds* and *sddsquery ex2.sdds -col* are equivalent. However, in some cases it does matter (e.g., the filename order described above).

E. Getting SDDS Help

All of the SDDS commands require some command-line arguments. Typing the SDDS command without any arguments gets a help message. Note that some can be very long (e.g. *sddsplot*). As an example, the command *sddsprintout* brings up the following message:

usage: sddsprintout [-pipe=[input][,output]]
[<SDDSinput>] [<outputfile>]
[-columns[=<name-list>[,format=<string>][,endsline]]
[-parameters[=<name-list>[,format=<string>][,endsline]]
[-array[=<name-list>[,format=<string>]]
[-fromPage=<number>] [-toPage=<number>]
[-formatDefaults=<SDDStype>=<format-string>[,...]]
[-width=<integer>]

There is also an online SDDS users' manual [5] which provides more detailed descriptions of the command usages.

VI. EPICS DATA COLLECTION

A. Using sddsmonitor

This program allows the value of any process variable to be logged on a periodic basis. For example, it may be required to monitor a power supply output current every 0.2 seconds for 10 minutes, or to log the storage ring vacuum every 10 seconds for 8 hours.

In order to use sddsmonitor, a configuration file must be created containing the names of the process variables to be monitored. Here is an example:

SDDS1
&description text="example list of names", &end
&column name="ControlName", type=string, &end
&column name="ReadbackName", type=string, &end
&data mode=ascii, &end
3
P:BM:CurrentAI P:BM:Current
P:Q1:CurrentAI PQ1:Current
P:Q2:CurrentAI PQ2:Current

The first column, *ControlName,* is the process variable name; the second column, *ReadbackName*, is the name to be given to the data column in the output file. The *description* text string is optional.

Assuming the above file is called *ex1.monitor* and that a total of 20 samples are required, one every 2 seconds, the command would be *sddsmonitor ex1.monitor par.sdds -steps=20 -interval=2*. The new file *par.sdds* will contain the following columns:

Step
Time
TimeOfDay
DayOfMonth
P:BM:Current
PQ1:Current
PQ2:Current

The columns *TimeOfDay* and *DayOfMonth* are in units of hours and days, respectively.

Having collected the data, *sddsplot* could then be used to plot the data. For example, to plot the dipole current against time of day, type *sddsplot -col=TimeOfDay,P:BM:Current -topline="PAR Dipole Current vs. Time" par.sdds*. An example result is shown in Figure 5. Instead of specifying the number of steps, the total time for collecting data could be specified; e.g., in the above example, replacing the *-steps=20* by *-time=40* would produce the same result. The time can also be specified in hours or days, for example, *-time=2,d* collects data for 2 days, *-time=1,h* for 1 hour.

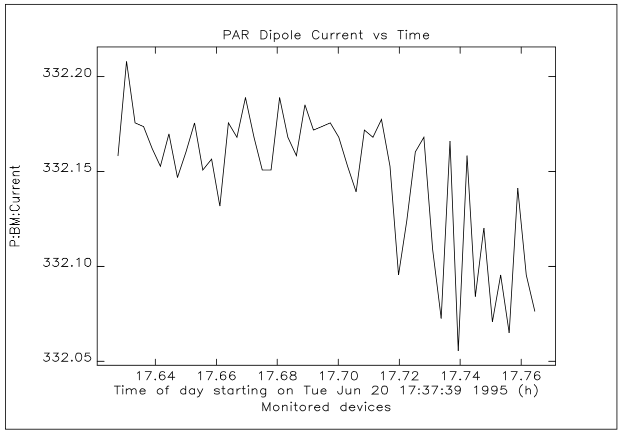

Figure 5: Example Data from sddsmonitor

Another feature of *sddsmonitor* is the *-trigger* option. This acts much like a digital oscilloscope trigger. Data is sampled at the specified rate but is not logged until the trigger event occurs. This is particularly useful for collecting data around a rare event. For example, it may be required to monitor the process variables in the above example, but only if the current exceeds 200A. This could be achieved by replacing *-steps=20* by *-trigger=B:BM:CurrentAI,level=200,slope=+,before=2,after=200.* A total of two samples would be logged before, and 200 samples logged after the current exceeded 200A.

B. Data Rate Limitations

Note that since the data is collected over the network, there are limitations to the rate at which data can be collected. Typically, a process variable cannot be read faster than about 10Hz.

C. Collecting Waveform Data

In some cases data is collected in the form of waveforms rather than individual samples and is processed by EPICS as a waveform process variable. A variant of *sddsmonitor*, called *sddswmonitor*, allows collection of data from these waveform process variables. The process variable names can be specified in a configuration file with columns supplied on the command line.

VII. SDDS DATA CONVERSION

Several programs are available to convert data between SDDS and other formats, either to analyze data from specialized equipment, or to make SDDS data available to other programs. Data can also be exchanged with several mathematical programs. Some of the available conversion programs are described below.

A. ASCII to Binary

In SDDS, ASCII and binary data files are completely interchangeable. The benefits of ASCII files are obvious, but as the files get larger, there is a size and access-time penalty, so binary files become preferable.

Conversion between the two data modes can be achieved using *sddsconvert*; e.g., to convert the file *ex1.sdds* to binary type *sddsconvert ex1.sdds -binary.* To convert back again use *-ascii* instead of *-binary.*

sddsconvert can also be used to rename or delete data.

B. SDDS to Spreadsheet

The program *sdds2spreadsheet* allows creation of an ASCII file for importing into spreadsheet programs. The resulting file can be delimited by any user-supplied character.

C. Oscilloscope to SDDS

Conversion routines have been written for a number of oscilloscopes and spectrum analyzers used at APS. For example, the program *tek2sdds* converts data from the Tektronics DSA602 oscilloscope to SDDS format. There are also programs to save and restore setups for various instruments using the SDDS data format.

VIII. UNIX SHELL SCRIPTS

The power of many of the above-described tools is significantly enhanced by the use of UNIX shell scripts to combine multiple tools, creating a new custom application. A shell script is a sequence of UNIX commands in a text file. The power of the shell script comes from the ability to quickly develop applications using existing programs and tools. They are extensively used by the APS operations staff for both machine control and analysis [6].

The simplest shell script might contain a single *sddsplot* command and be used to save typing on the command line, whereas a more complicated shell script may perform data collection and detailed analysis of power supply waveforms from the synchrotron accelerator. Any valid sequence of UNIX commands can be included in a shell script, and there are many programming-type features available such as program looping and the use of variables. The following example issues a series of caput commands to change the status of several process variables (in this case to open some vacuum valves).

```
#! /bin/csh
echo “This script opens all the gate valves”
echo “in the synchrotron”
echo “enter <yes> to continue”
if ( “$<“ != “yes” )  then
        echo “exiting”
        exit
endif
foreach valve (01 02 2A 03 04 05 06 07 07)
caput VM:BM:00”${valve}”:openBO 1
end
```

Before the *caput* commands are issued, the script informs the user of its purpose and waits for input (indicated by the *“$<“* in the *if* statement). If the user does not respond with *yes*, then the program exits. Otherwise it issues nine *caput* commands, each time replacing *${valve}* with one of the strings 01, 02 etc.

The following example starts *sddsmonitor* with a predefined configuration file but with a user-supplied data file name:

```
#! /bin/csh
if ( ${#argv} != 1 ) then
   echo “usage: startmon <outputfile>”
   echo “...Starts sddsmonitor for the par power supplies”
   echo “at 10 second intervals for 8 hours”
   exit
endif
set filename = $1
sddsmonitor PAR.mon ${filename} \
   -interval=10,s -time=8,h &
```

In this case, the script checks that the user has supplied a filename on the command line (the *if* statement). If not, a usage message is printed and the script exits. Otherwise the sddsmonitor program is started.

IX. SUMMARY

Using a combination of the EPICS control system, the SDDS toolkit, and UNIX shell scripts, users can easily develop custom monitoring applications with a minimum time investment and without having to develop a custom C program. Such applications are regularly developed by staff involved in day-to-day operations at APS. Often a few minutes spent writing a simple shell script can allow the control system to perform data collection and analysis, rather than having to perform the task manually. The examples given in this paper show that these are available to anyone with access to the control system and are not limited to expert software developers.

X. REFERENCES.

[1] Dalesio, L., Hill, J., Kraimer, M., Murray, D., Hunt, S., Claussen, M., Watson, C., Dalesio, J., "The Experimental Physics and Industrial Control System Architecture," Proceedings of ICALEPCS, Berlin, Germany, Oct. 18-22, 1993.

[2] Borland, M., Emery, L., "The Self-Describing Data Sets File Protocol and Toolkit,” these proceedings.

[3] Ousterhout, J., Tcl and the Tk Toolkit, (Addison Wesley, 1994).

[4] Kramer, M., Cha, B-C K., Anderson, J., “The Alarm Handler Users’ Guide,” URL: http://www.aps.anl.gov/asd/controls/epics/manuals/alh/alh.html.

[5] Borland, M., “Users’ Guide for SDDS Toolkit Version 1.4,” URL: http://www.aps.anl.gov/asd/oag/manuals/SDDStoolkit/SDDStoolkit.html.

[6] Borland, M., Emery, L., Sereno, N., "Doing Accelerator Physics Using SDDS, UNIX, and EPICS,” these proceedings.