ANL/ASD/CP--88244
CONF-951036--14

The Self-Describing Data Sets File Protocol and Toolkit[1]

M. Borland, L. Emery
Argonne National Laboratory; 9700 S. Cass Avenue; Argonne, Il 60439

Abstract

The Self-Describing Data Sets (SDDS) file protocol continues to be used extensively in commissioning the Advanced Photon Source (APS) accelerator complex. SDDS protocol has proved useful primarily due to the existence of the SDDS Toolkit, a growng set of about 60 generic commandline programs that read and/or write SDDS files. The SDDS Toolkit is also used extensively for simulation postprocessing, giving physicists a single environment for experiment and simulation. With the Toolkit, new SDDS data is displayed and subjected to complex processing without developing new programs. Data from EPICS, lab instruments, simulation, and other sources are easily integrated. Because the SDDS tools are commandline-based, data processing scripts are readily written using the user's preferred shell language. Since users work within a UNIX shell rather than an application-specific shell or GUI, they may add SDDS-compliant programs and scripts to their personal toolkits without restriction or complication. The SDDS Toolkit has been run under UNIX on SUN OS4, HP-UX, and LINUX. Application of SDDS to accelerator operation is being pursued using Tcl/Tk to provide a GUI.

1 Self-Describing Data

As used in this paper, the concept of self-describing data starts from the recognition that when a scientist thinks of data, he typically associates a number of attributes with it. Among these are

1. The name by which the data is known.
2. The units of the data, if any.
3. The meaning of the data (i.e., a description).
4. A mathematical symbol to represent the data, if appropriate.
5. The type of data (e.g., floating-point, integer, or character).

A self-describing file protocol (SDFP) should incorporate these attributes automatically. Further, the user of self-describing data obtains the data by name. That is, unlike traditional text and unformatted data, SDFP does not require the user to know, for example, the column of a table in which a certain quantity appears.

This is a crucial feature of the self-describing data concept, since it enables one to construct generic programs that can perform similar operations in several applications. A good example would be a graphics program that plotted data by name, rather than being written specifically for a single application. (Compare this to the current state in accelerator physics, where many similar simulation codes have their own custom-made graphics packages.)

For the purpose of discussion, assume that the data can be meaningfully organized into a single table. This table might, for example, contain various quantities such as position, Twiss parameters, element name, etc. Any program compliant with the SDFP in which the data was written would be able to access the columns of this table without regard for how the data was organized. The program would access the data by name, using routines supplied by the creator of the SDFP. If the program needed only certain columns of data (e.g., position and horizontal beta function), it would access only those columns. The presence of additional columns of data (e.g., dispersion), would be

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

irrelevant. Further, the program wouldn't need to know what the source of the data was—it could be from any source, including direct user input into a file, output from any compliant simulation code, or measurement on a real accelerator.

Consider the number of accelerator codes that perform comparable computations and produce comparable output in dissimilar formats. If all of these codes employed the same SDFP for their output, users and programs could access data without regard for the code that generated it. Post-processing or multistage simulation would be possible without custom integration of programs. That is, programs would not need to be custom designed to provide output to each other in order to work well together. This would happen automatically through the SDFP mechanism. This allows users to combine programs in ways that were not planned by the creators of the programs.

The only constraint would be that the codes used the appropriate names for quantities. In practice, this restriction can be reduced by providing a name translation table. Also, programs can be designed to request data under several different but equivalent names, e.g., "betax" and "betah" for the horizontal beta function.

Another significant advantage of using an SDFP is the possibility of making program "toolkits." We have already noted that using an SDFP allows programs to read data from many sources. If a program is created that both reads and writes data in the same SDFP, then this program may potentially be used to alter data "upstream" of any other program that reads the SDFP. If many programs exist that fulfill this requirement, then these programs can be employed in nearly arbitrary sequences to process data. The usefulness of each program is vastly increased by the possibility of using it as part of such a sequence. Such a set of programs comprises a toolkit, taking advantage of the SDFP to virtually guarantee that any member program will provide readable data to any other member program.

2 The SDDS File Protocol

The principles elucidated in the previous section have been implemented in the Self-Describing Data Set (SDDS) protocol, developed at APS. Any SDFP implementation must make certain assumptions about what type of data will be stored and how it will be arranged. These assumptions comprise the data model for the protocol.

An SDDS file consists of a header section and a data section. The header section declares that the file is an SDDS file, and what SDDS version it is in. It further defines zero or more data elements that together constitute a data structure. The SDDS model organizes the data section into a series of "data pages," each of which is an instance of the data structure defined in the header. That is, each page of any file must contain the same elements but may contain different specific data. For example, each page could contain the Twiss parameters for a different accelerator or for a different tuning of the same accelerator.

Within each page, the following classes of data are recognized:

1. Parameter data, consisting of single values that may either be fixed throughout the file or vary from page to page.
2. Tabular data, consisting of an arbitrary number of rows of mixed-type data. The data in the table is referred to by the name of the column. Any number of columns may be defined, but the same columns are expected for each page of the file.
3. Array data, where each element is of fixed but arbitrary dimension (i.e., an arbitrary number of array indices is allowed). Each array is accessed separately, by name, but may be placed in a group with other arrays. The size of an array may vary from page to page. (Note that a parameter is essentially an array containing a single value, but has simplified access from within a program compared to an array.)

Any element of these data classes may have one of the following C-language data types: `float`, `double`, `short`, `long`, `char`, and `char *`. These are, respectively, single- and double-precision floating point, short and long integers, single characters, and character strings. All of these types may be mixed in the tabular data section, but each column must contain data of fixed type.

SDDS has no restrictions on the numbers of each type of element, on the length of the tabular data, on the dimension of arrays, or on the size of arrays.

To continue Twiss parameters example, one might create an SDDS file containing:

1. Parameters: The name of the lattice, the tunes, the chromaticities, the acceptances, etc.
2. Columns: The element name, the position, the Twiss parameters, apertures, etc.
3. Arrays: The response matrix, matrices for tune and chromaticity adjustment, etc.

While it is possible to design a self-describing file protocol that is more general than SDDS, we have found that SDDS allows us to conveniently store almost any data. At worst, the user may need to store disparate data in different, parallel files; this actually has the advantage of making the data easier and faster to access. When evaluating the desirability of a more flexible data model, it is important to recognize that too much generality will adversely affect ease of development, ease of use, and development time. The complexity of highly general models may explain why they are not in wide-spread use.

Some examples of the types of data stored in SDDS at APS: particle tracking input and output; Twiss parameters and transfer matrices along an accelerator; beam loss data from tracking simulations; simulated orbit/trajectory correction results and statistics; orbit/trajectory data from the APS accelerators; simulated rf cavity fields; backup/restore files for the APS accelerators; archival data for the APS accelerators; video images and other density maps; data transferred from digital oscilloscopes and spectrum analyzers.

Of course, each of these types of data could have been stored in other types of files. What is new is that SDDS allows the user to store all of these types of data in the same type of file and to use the same set of tools with all of them. For example, the same program that plots beta functions vs position can also be used to plot archived oscilloscope signals vs time, or to make scatter plots of simulation tracking data. Similarly, the same program that does FFTs of simulation tracking data from a simulation can do FFTs of turn-by-turn BPM readings from the APS storage ring. These FFTs, of course, would be plotted by the same program as all the other data.

Another advantage of SDDS is that the data may be either ASCII or unformatted (i.e., "binary"). The SDDS header is in ASCII and has a familiar namelist format. It is a simple matter to create an SDDS file using print statements within a program, giving immediate access to a range of SDDS tools (see the next section). The SDDS header also has features to make it easy to convert existing text data into SDDS protocol; in many cases, one simply creates a header and attaches it to the top of the file. We have found that these features were extremely important in helping users to switch from text printouts and other custom formats to SDDS.

As noted above, SDDS incorporates a protocol and version identification string as part of the header. This allows immediate detection of whether a given file is in SDDS protocol, and what version of the protocol it is using. The SDDS library routines are configured to allow reading of any SDDS version, so that an SDDS data file never becomes obsolete. This permits upgrades of the protocol itself without disrupting users.

3 The SDDS Toolkit

The SDFP concept is a break with the traditional way that accelerator physicists store data. A further break is the introduction of the toolkit concept for data processing. As described above, a toolkit is group of independent but cooperative commandline programs.

Traditionally, data processing has involved writing single- or few-purpose programs that use data from a single source (e.g., a particular simulation or experiment). This is in spite of the fact that the operations involved in different data processing applications are often essentially identical except for the specific data involved. With the use of an SDFP, it makes more sense to write generic programs that are not specific to experiments or simulations. This is because an SDFP allows one to access data by name, which permits symbolic specification of operations. The simplest example conceptually is graphics, as described briefly in the previous section; there is no good reason that a single graphics program cannot be used for almost all of the codes and data employed in accelerator physics. (In fact, there are several simulation codes in use at APS that use only SDDS files [1].)

While toolkit programs can be shared by users, development is decentralized. Since the only requirement for a toolkit program is that it read or write SDDS files, anyone may make a contribution without any negative impact on the rest of the toolkit. This is an advantage over "all-in-one" packages, which force the user to import his data into a single program and perform all operations within a centrally-controlled environment. In contrast, toolkit programs are combined through use of the local command shell and through command scripts [2]. SDDS-compliant programs automatically work together, with little or no planning on the part of code developers. To date, the authors know of nine individuals at APS who have developed SDDS-compliant programs.

The SDDS Toolkit is a growing group of about 60 programs that uses SDDS files. Most of the programs both accept SDDS input and produce SDDS output. The following sections provide a brief description of a number of the programs, grouped by functional type. Several programs are given more detailed descriptions. The reader may note that most of the programs process column data only, producing either new columns or parameters that reflect the processing. This is an indication of the extent to which this single feature of the SDDS data model serves almost all the cases we have encountered.

The reader is referred to our companion paper [2] for a list of some of the APS commissioning data that has been processed using the SDDS Toolkit, and examples of how the Toolkit programs are coordinated. Extensive hypertext manuals are available with the SDDS distribution package for the Toolkit itself [3] and the SDDS-compliant EPICS programs [4]. A tutorial introduction with an emphasis on EPICS applications is available in [5].

4 SDDS Toolkit Programs

The intention is of this section is to provide an overview of some of the SDDS Toolkit programs that are presently available. For most programs, a brief and usually very incomplete description is given. For the less obvious programs, a brief example is given. A few of the most heavily-used programs are given long descriptions.

4.1 Data Analysis Tools

- `sddschanges`: Analyzes changes in column data from page to page in a file, relative to a reference file or the first page. An example application is computing changes in waveform data over time relative to an initial waveform.
- `sddscorrelate`: Computes correlation coefficients and correlation significance among multiple columns of data. An example application is searching for possible causal relations among many columns of time-series data.
- `sddsderiv`, `sddsinteg`: Numerical differentiation or integration of multiple data columns versus a single column.
- `sddsdigfilter`, `sddsfdfilter`: Perform time- and frequency-domain digital filtering.

- sddsenvelope: Analyzes column data across pages to find minima, maxima, averages, standard-deviations, etc., on a row-by-row basis. An example application would be finding the envelope and average of a repeatedly-sampled waveform.
- sddsexpfit, sddsgfit: Exponential fit or Gaussian fit to column data.
- sddsfft: Computes fast Fourier transforms and power spectral densities of column data. Will provide real and imaginary parts, magnitudes, and phases.
- sddshist, sddshist2d: One- and two-dimensional histograms and simple statistics.
- sddsinterp: Does arbitrary-order polynomial interpolation of multiple data columns as a function of a single column.
- sddsoutlier: Eliminates statistical outliers from data using, for example, a limit on the number of standard deviations by which a point may differ from the mean of the group.
- sddspeakfind: Finds values of many columns at locations of peaks in a single column.
- sddspfit: Does arbitrary order polynomial fits to column data, including error propagation and adaptive fitting.
- sddsprocess: A generic data processing program that, among other features, permits: defining new tabular and parameter data in terms of existing data using user-specified expressions; printing, scanning, editing, and subprocess execution (with output capture) of string data; units conversion; selection of tabular data rows by multiple numerical windows and wildcard matching, with user-specified logic among selection criteria; creation of parameters based on any of 29 types of analyses of tabular data.

  The tabular data analyses include statistics, waveform and pulse parameters, and linear fit parameters. Among the statistics are arithmetic average, rms, standard deviation, median, minimum, and maximum. Waveform analyses include amplitude, baseline, risetime and falltime. Analyses may be invoked for an arbitrary number of data columns using wildcards, with parameter names being constructed from a user-supplied template. Many analyses support weighting and winnowing based on data from secondary columns.
- sddspseudoinverse: Reads the numerical tabular data as though it was a matrix, and outputs the SVD inverse. The tabular data need not form a square matrix. Options allow one to adjust the number of singular values selected. This program is used to calculate gain matrices in a number of generalized feedback applications.
- sddssmooth: Smooths columns of data using multipass nearest-neighbor averaging and optional "despiking." Often used in concert with sddspeakfind to give reliable peak frequencies from noisy spectra.
- sddsslopes: Calculates the slopes and intercepts of selected columns using another column as independent variable. When applied to data logged at intervals using the time data column as the independent variable, this program directly produces time trends. This program has also been used extensively in calculating linear responses of systems to actuators. As such, sddsslopes and sddspseudoinverse are often used to analyze experimental data to produce a gain matrix for feedback.

4.2 Data Manipulation Tools

- sddsbreak: Breaks data pages into new, separate pages based on changes in column data and other criteria. For example, sddsbreak can analyze time-series data to find probable time gaps, and make a separate page for each contiguous section.
- sddscollapse: Collapses a data set into a single data page by deleting the tabular data and turning the parameters into columns. This is an important tool in data collation operations. Typically, one processes multipage tabular data to produce parameters for each page (e.g.,

statistics for several columns), then collapses the result to obtain a single page of tabular data. Several such data sets can be combined using sddscombine, which puts one formally in the same position as when one started. This program is often used with sddsprocess in data reduction cases where thousands or even millions of data points must be reduced to a simple result.

- sddscombine: Combines any number of data sets into a single data set by adding data from each successive data set to a newly-created data set.
- sddsconvert: Allows conversion of a data set between binary and ASCII, with optional deletion and renaming of columns, arrays, and parameters.
- sddssort: Sorts the tabular data section of a data set by the values in any number of named columns, and optionally eliminates duplicate rows.
- sddstranspose: Reads the numerical tabular data as though it was a matrix, and outputs the transpose. Applying sddstranpose again to the output reproduces the original file. This program is used often to convert multi-column one-row SDDS data sets, obtained from sddscollapse, say, into two-column multi-row data sets (an extra string data column comes from the original column names), which are then suitable for plotting. For example, sddsprocess and sddstranspose could be used to take a file with 360 columns of beam-position-monitor samples, and create a plot of rms position variation vs BPM name.
- sddsxref: Creates a new data set by adding selected rows from one data set to another data set. The rows are selected by matching the string or numeric values in a specified column that is present in both of two pre-existing data sets. Will also transfer parameter and array data between files.

4.3 Graphics Tools

- sddscontour: Makes contour and color-map plots from a data column interpreted as a matrix, or from many data columns. Optionally does FFT-based interpolation and filtering. Uses the same device drivers as sddsplot.
- sddsplot: A very powerful, generic, device-independent graphics program for plotting tabular and parameter data. The program is equally capable of "quick-and-dirty" and publication-quality plots. Among the supported devices are X-windows, color and black-and-white Postscript, Tektronix, REGIS, and various PC graphics cards. While the program is command-line-driven, it brings up a GUI under X-windows. (Support for graphics using such old methods as the Tektronix and REGIS languages may seem unimportant, however it does enable one to get graphical displays using a wide variety of terminal emulators running on personal computers.)

  sddsplot was designed to handle and organize large amounts of data. Any number of data elements may be plotted from any number of files. The data to be plotted from any file is simply named on the commandline, with optional wildcarding. The program organizes data into a potentially unlimited number of "panels" on a potentially unlimited number of logical output pages; in this context, a panel is a stand-alone graph containing zero or more data points from any number of sources, and occupying all or part of a page. When desired, grouping of data into panels may be specified in numerous ways, such as grouping columns of data by tag parameter values stored in the data files, or grouping columns of data from many different files by the name of the data. For X-windows, a movie mode is available that displays pages in rapid succession. Figure 1 shows an example of a page from a movie made from tracking data for the APS booster; the overlayed histograms are made with sddshist. Figure 2 shows an example of plotting Twiss parameters using sddsplot.

Plotting types include lines, symbols, error bars, dots, impulses or bars, and arrows; things like color and symbol type may be varied automatically by the program to distinguish different data, or these attributes may be specified explicitly. When plotting lines, `sddsplot` can optionally analyze data for gaps and break the line at appropriate points. Data may be filtered and subjected to matching operations based on data in other columns or parameters, as well as sparsed and randomly sampled. Title, label, and legend strings for each set of data may be extracted from the corresponding SDDS file.

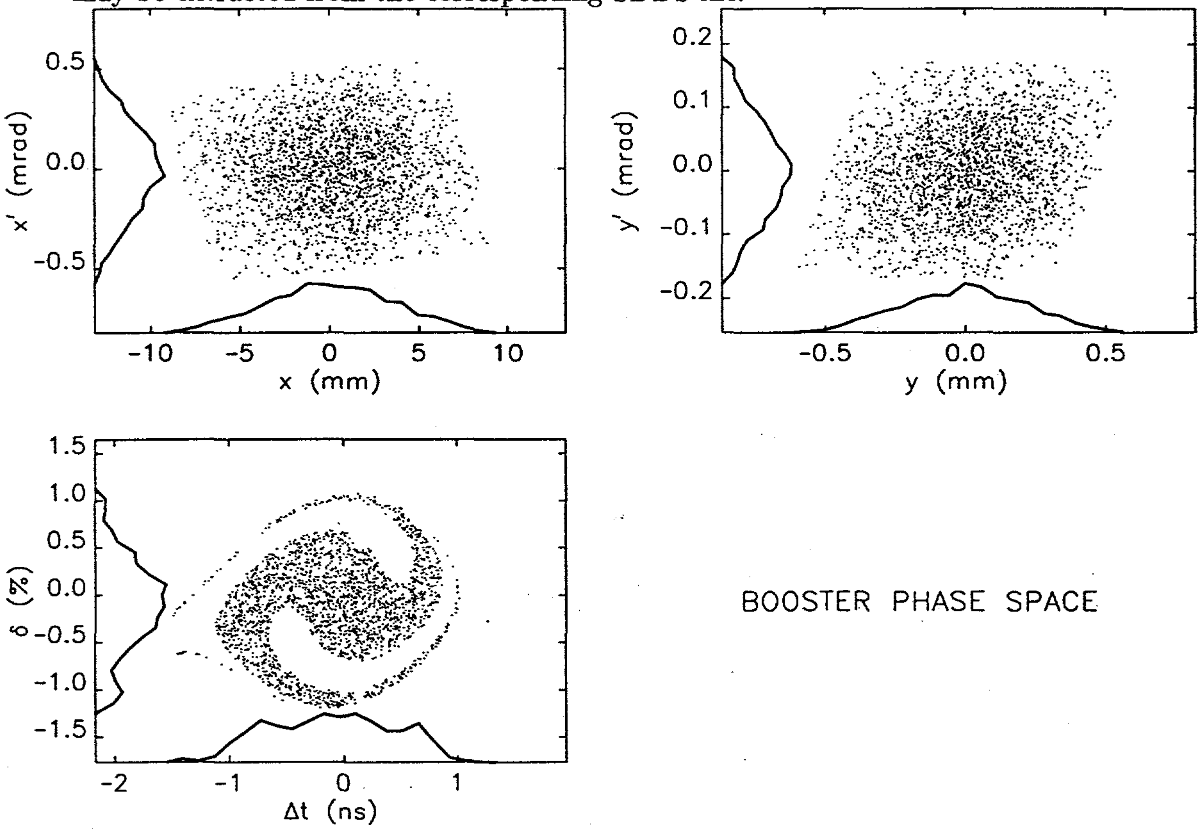

Figure 1: A frame from a movie of APS booster 6-D phase-space tracking simulation, with a deliberate mismatch.

4.4 Miscellaneous Tools

- File Protocol Convertors: Programs are available to convert from the following into SDDS: Hewlett-Packard CITI; Hewlett-Packard HP54542 scope format; Spiricon Laser Beam Analyzer; and others. Programs are available to convert SDDS to the following: Mathematica[2]; Excel and Wingz spreadsheets; others.
- `sddsprintout`: Makes customized printouts from SDDS data. This means that while an SDDS file may contain dozens or even thousands of parameters and columns, a printout can be easily made that contains just the data of interest.
- `sdds2stream`: Takes column or parameter data from a list of SDDS data sets and delivers it to the standard output as a stream of values. Often used to get data into a shell variable in `csh` or `tclsh`.
- `sddsquery`: Prints a summary of the SDDS header for a data set, or delivers this data as a new SDDS file.

[2]Mathematica is a registered trademark of Wolfram Research, Inc.

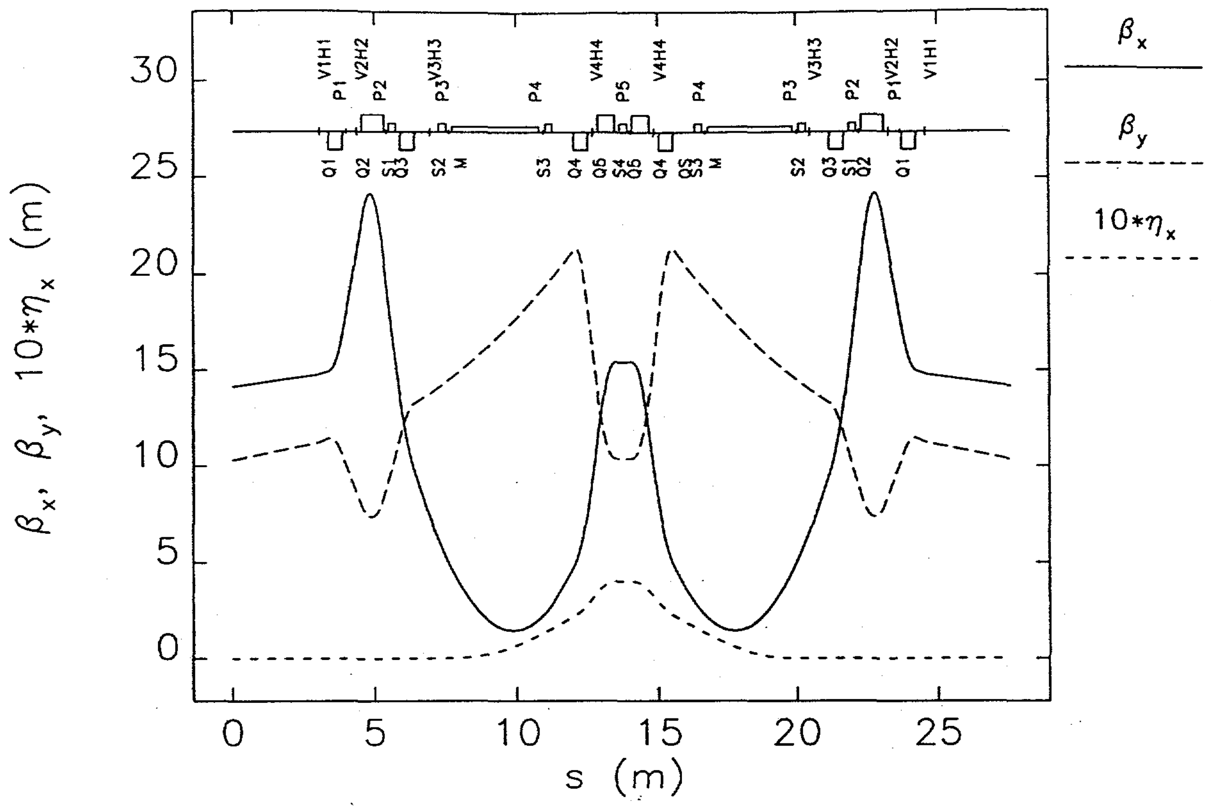

Figure 2: Twiss parameters for one sector of the APS ring.

5 SDDS-Compliant EPICS Tools

The control system at APS is the Experimental Physics and Industrial Control System (EPICS) [6]. A toolkit approach is adopted in designing workstation applications that communicate with EPICS. In order to take advantage of the existing SDDS toolkit, those applications that exchange data between EPICS and data files use the SDDS file protocol.

The tools listed below are very general. They don't presume that an accelerator is being controlled. These tools can easily be adapted to non-EPICS control systems by replacing the EPICS channel access libray calls in the source codes with appropriate substitutes. A set of detailed case studies of the use of some of these tools is available in [2].

5.1 Configuration Save and Restore Tools

- `burtrb`, `burtwb` (Back-Up and Restore Tool): These programs perform EPICS save and restore operations, reading data from and writing data to the control system. SDDS protocol is used both for the "request file" (configuring a read operation) and the "snapshot file" (containing data from a read operation or for a write operation). Because SDDS is used, a snapshot file can function as a request file, as it contains all of the essential data from the request file. Information such as time stamp, user ID, and operator comments is stored in the file as SDDS parameters; additional parameters may be added at will without side effects. Operations such as selection of subsets and comparison of snapshots are performed using various SDDS Toolkit programs, forming the basis for a save/compare/restore facility [7].

- `toggle`: Alternates between two sets of process variable values stored in snapshot files at a regular interval. Example application: Alternate between two configurations of an accelerator for debugging accelerator performance.

5.2 Data Collecting Tools

- `sddsmonitor`, `sddsstatmon`, `sddsvmonitor`, `sddswmonitor`: These programs provide various types of data logging into SDDS files. Each is configured by one or more SDDS files containing names of process variables (PVs) to read. The various versions are, respectively, for logging values for simple lists of PVs; statistics for the same; values based on list multiplication and organized into vectors; and waveform PVs. The vector and waveform versions will accept input data for scalar logging as well, thus partially duplicating the function of `sddsmonitor`. Normally, the programs log data at a fixed, user-specified time interval. However, options are provided for event-based data logging. For all of the programs, data taking may be controlled by a parent process using messages in the standard input stream. All except `sddsstatmon` accept additional SDDS input specifying test limits for conditional data logging. For `sddsmonitor`, data may be stored in a circular buffer and written to disk only when user-defined trigger or glitch events are detected.

- `sddssnapshot`: Reads values of process variables and writes them to a snapshot file. This program differs from `burtrb` in that it may operate in a server mode in which a new file is written to a named output file whenever the signal SIGUSR1 is received. An application would be taking successive sets of data triggered on certain events determined by another process, without remaking any channel access connections.

- `sddsexperiment`, `sddsvexperiment`: These programs provide for experiment execution and data collection. The most basic usage involves variation of a setpoint in equal steps, with data gathering for each setpoint. More sophisticated applications may involve variation of an arbitrary number of setpoints on a multidimensional grid. Also supported are computation of setpoints from user-supplied equations, allowing for nonlinear variation. Subprocess execution is supported to permit scripts to be run in coordination with setpoint variation and data gathering; these scripts may acquire specialized types of data (e.g., video images) or perform specialized tasks (e.g., adjusting a motorized attenuator to obtain a specified power level). Data gathering includes optional computation of averages and error bars at each measurement point. (The two variants of the program differ in the mechanism by which the measured process variable names are supplied and the way the data is organized in the output file.)

5.3 Control Tools

- `sddcontrollaw`: This program performs simple feedback on process variables. Basically, a set of control process variables is used to regulate to zero a set of readback process variables. Optionally, the program can hold constant the initial values of the readback process variables. The values of the control process variables ($\mathbf{u_n}$) applied at some iteration of the feedback are related to the values of the readback process variables ($\mathbf{x_n}$) at the previous iteration through the matrix equation $\mathbf{u_n} = \mathbf{u_{n-1}} - K\mathbf{x_n}$, a variant of the control law equation in control theory. An SDDS input file specifies the gain matrix K and all the process variable names.

  To make the control robust, a series of validity tests on process variable values are implemented by means of an additional SDDS file containing the names of process variables and their corresponding limit values. Feedback iterations proceed only when all of the test process variables are within the specified range. When one of the tests fails, control is suspended (i.e., the control variable values are untouched by the program) until all of the tests succeed.

Some of the applications have been: removing slow energy drifts in the linac, correction of linac and most beamline trajectories, correction of positron accumulator ring and storage ring orbit, "slow" feedback on storage ring gap voltage amplitude to compensate for beam loading, and performing complex experiments in association with `sddsexperiment`.

The gain matrix K is usually determined empirically with the use of `sddsexperiment` or `sddsvexperiment` (to determine parts of the response matrix) and other SDDS tools (to invert the response matrix to obtain the gain matrix). The process of determining the gain matrix and applying `sddscontrollaw` for the different systems of the APS accelerators is essentially the same, only the names of process variables are changed.

- `squishPVs`: This program provides for automated "knob tweaking." It was originally written for first-turn trajectory correction in the APS ring, but is of more general application. The name comes from the way the program flattens out the trajectory. The program takes an SDDS input file giving a series of actuator names and any number of readback names for each actuator. It then adjusts each actuator in turn to minimize either the rms or mean-absolute-value of the corresponding readbacks. While quite slow compared to use of a response matrix and `sddscontrollaw`, `squishPVs` has the advantage of being completely insensitive to, for example, magnet and beam-position-monitor polarity errors.

6 Acknowledgements

The authors wish to acknowledge the following individuals for contributions to the SDDS toolkit and EPICS-specific tools, either in the form of suggestions, bug discoveries, or contributed programs: J. Carwardine (APS), Y. Chung (APS), K. Evans (APS), N. Karonis (FNAL), E. Lessner (APS), S. Milton (APS), G. Rinehart (IPNS), C. Saunders (APS), M. White (APS).

References

[1] M. Borland, "A Self-Describing File Protocol for Simulation Integration and Shared Postprocessors," Proceedings of the 1995 Particle Accelerator Conference, May 1-5, 1995, Dallas, Texas (to be published).

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

[3] M. Borland, "User's Guide for SDDS Toolkit Version 1.4," http://www.aps.anl.gov/asd/oag/manuals/SDDStoolkit/SDDStoolkit.html .

[4] L. Emery, private communication.

[5] J. A. Carwardine, "An Introduction to Plant Monitoring Through the EPICS Control System," these proceedings.

[6] L. R. Dalesio, M. R. Kramer, A. J. Kozubal, "EPICS Architecture," in *ICALEPCS* 1991, pp. 278–281.

[7] D. J. Ciarlette, R. Gerig, "Operational Experience from a Large EPICS-Based Accelerator Facility," these proceedings.