ANL/ASD/RP--88322

User's Guide for SDDS Toolkit Version 1.4

Michael Borland
Advanced Photon Source

July 6, 1995

The Self Describing Data Sets (SDDS) file protocol is the basis for a powerful and expanding toolkit of over 40 generic programs. These programs are used for simulation postprocessing, graphics, data preparation, program interfacing, and experimental data analysis.

This document describes Version 1.4 of the SDDS commandline toolkit. Those wishing to write programs using SDDS should consult the *Application Programmer's Guide for SDDS Version 1.4*[1]. The first section of the present document is shared with this reference.

This document does not describe SDDS-compliant EPICS applications, of which there are presently 25. They will be covered in a separate manual.

1 Why Use Self-Describing Files?

Before answering the question posed by the title of this section, it is necessary to define what a self-describing file is. As used here, data in self-describing files has the following attributes:

- The data is accessed by name and by class. For example, one might ask for "the column of data called X", or "the array of data called Y". Self-describing data is *not* accessed by position in a file; e.g., one would not ask for "the third column of data".
- Various attributes of the data that may be necessary to using it are available. For example, one can ask "what are the units of column X?", "what is the data-type of array Y?", or "how many dimensions does array Y have?" .

The primary advantage of accessing data and its attributes by name rather than the traditional position method is that one can then construct generic tools to manipulate data. Self-describing data contains the information that tools need to manipulate various types of data correctly. For example, one can plot data with a generic tool that accepts the names of the quantities to plot; such a tool will be able to plot data of different types (e.g., integer or floating-point), and display relevant information (e.g., units) on the plot.

Another advantage of self-describing data is that it makes the interface between programs more robust and flexible. Since programs only look for data by name, insertion of additional data into a file is irrelevant. Multiple programs may interface to a single program even in the face of differences in what data each places in its output files. E.g., program A may create data in single-precision, with columns called X, Y, and Z. Program B may create data in double-precision, with columns called X, Y, and W. If all programs employ self-describing files, then a properly-written program C could access X and Y from the output of either program A or B. It could also determine that the output of program B didn't contain data called Z, and warn the user of this.

The SDDS file protocol incorporates these aspects of self-describing data. It has been found extremely valuable for storing data from simulation, experiment, and accelerator operation at the Advanced Photon Source (APS). SDDS is made more valuable by the existence of a growing "toolkit" of over 40 generic commandline programs that perform many varied operations using SDDS files. Indeed, while there are more general self-describing protocols than SDDS, to the author's knowledge only SDDS has a powerful, generic program toolkit built around it. In the author's opinion, this is possible because SDDS protocol is general but not *too* general. The SDDS Toolkit is used to postprocess simulation output, to analyze experimental and archival data, to prepare data for input to other programs, to provide a bridge between separate simulation codes, to display data graphically, to collate and section accelerator save/restore files, and much more.

While it is very flexible, SDDS is also fairly simple. Because SDDS features interchangeable binary and ASCII formats, it is an easy matter to create an SDDS data set "by hand", when necessary. It is also easy to modify existing programs to print in SDDS protocol, and to create headers to convert existing text data to SDDS. At the same time, data archivers, large-scale simulations, and similar applications can store data in binary for quick access and disk economy. These and other features contribute to the widespread use of SDDS at APS.

2 Definition of SDDS Protocol

2.1 Introduction

An SDDS file is referred to as a "data set". Each data set consists of an ASCII header describing the data that is stored in the file, followed by zero or more "data pages" or "data tables" (the former term is preferred, though the latter is used in many places). The data may be in ASCII or unformatted (i.e., "binary"). Each data page is an instance of the structure defined by the header. That is, while the specific data may vary from page to page, the structure of the data may not.

Three types of entities may be present in each page: parameters, arrays, and columns. Each of these may contain data of a single data type, with the choices being long and short integer, single and double precision floating point, single character, and character string. The names, units, data types, and so forth of these entities are defined in the header.

Parameters are scalar entities. That is, each parameter defined in the header has a single value for each page. Each such value may be a single number or a single character string, for example.

Arrays are multidimensional entities with potentially varying numbers of elements. While there is no restriction on the number of dimensions an array may contain, this quantity is fixed throughout the file for each array. However, the size of the array may vary from page to page. Thus, a given two-dimensional array might be 2x2 in one page, 3x5 in the next, etc.

Columns are vector entities. All columns in a data set are organized into a single table, called the "tabular data section". Thus, all columns must contain the same number of entries, that number being the number of rows in the table. There is no restriction on how many rows the tabular data may contain, nor on the mixing of data types in the tabular data.

It is possible to design more sophisticated data protocols than SDDS, and this has in fact been done. However, the more flexible a protocol is, the more difficult it becomes to write generic programs that operate on data. Experience with SDDS has shown that there is very little data that cannot be *conveniently* stored in one or more SDDS files. In fact, most applications need only the parameter and tabular data facilities. Frequently, complex data is separated into several parallel files; the SDDS toolkit provides support for multifile operations that make this convenient.

The following is an example of a very simple SDDS file. Users who would prefer not to read the detailed description of the protocol in the next section may profit from using this example as

a guide.

```
SDDS1
! This is a comment line.  The previous line is required and identifies
! the file as SDDS.
! Define parameters:
&parameter name=Description, type=string &end
&parameter name=xTune, type=double &end
&parameter name=yTune, type=double &end
! Define columns:
&column name=s, type=double, units=m, description="longitudinal distance" &end
&column name=betax, type=double, units=m, description="horizontal beta function" &end
&column name=betay, type=double, units=m, description="vertical beta function" &end
&column name=ElementName, type=string &end
! Declare ASCII data and end the header:
&data mode=ascii &end
! First come the parameter values for this page, in the order defined:
Twiss parameters for the APS
35.215
14.296
! Second comes the tabular data section for this page, which has
! 50 rows in this example:
50
   0.000000   14.461726    9.476181        _BEG_
   3.030000   15.096567   10.445020          L01
   3.360000   15.242380   10.667547          L02
   3.860000   17.308605    9.854735           Q1
   3.975000   18.254680    9.419835          L11
   4.190000   20.094943    8.640450          L12
   4.520000   23.100813    7.529584          L13
   5.320000   21.435972    7.949178           Q2
   5.410000   20.278542    8.350441          L21
   5.620000   17.705808    9.332877          L22
   5.920000   14.341175   10.848446          L30
   6.420000   10.719036   12.405601           Q3
   7.120000    7.920453   12.969811          L41
⋮

  27.600000   14.461726    9.476181          L01
! The file may end at this point, or a new page may follow.
```

At this point, those who are new to SDDS may wish to skip to the Manual Pages Overview (section 3) in order to get a feel for the capabilities of the Toolkit. The details of SDDS protocol, the subject of the next section, are less important than what can be done with data once it is in SDDS protocol.

2.2 Structure of the SDDS Header

The first line of a data set must be of the form "SDDSn", where n is the integer SDDS version number. This document describes version 1.

The SDDS header consists of a series of namelist-like constructs, called namelist commands. These constructs differ from FORTRAN namelists in that the SDDS routines scan each construct, determine which it is, and use the data appropriately. There are six namelist commands recognized under Version 1. Each is listed below along with the data type and default values.

For each command, an example of usage is given. Several styles of entering the namelist commands are exhibited. I suggest that the user choose a style that makes it easy to pick out the beginning of each command. Note that while each namelist command may occupy one or more lines, no two commands may occupy portions of the same line.

Any field value containing an ampersand must be enclosed in double quotes, as must string values containing whitespace characters.

Another character with special meaning is the exclamation point, which introduces a comment. An exclamation point anywhere in a line indicates that the remainder of the line is a comment and should be ignored. A literal exclamation point is obtained with the sequence `\!`, or by enclosing the exclamation point in double quotes.

The commands are briefly described in the following list, and described in detail in the following subsections:

- **description** — Specifies a data set description, consisting of informal and formal text descriptions of the data set.
- **column** — Defines an additional column for the tabular-data section of the data pages.
- **parameter** – Defines an additional parameter data element for the data pages.
- **array** — Defines an additional array data element for the data pages.
- **include** — Directs that header lines be read from a named file. Rarely used.
- **data** — Defines the data mode (ASCII or binary) along with layout parameters, and is always the last command in the header.

The `column`, `parameter`, and `array` commands have a `name` field that is used to identify the data being defined. Each type of data has a separate "name-space", so that one may, for example, use the same name for a column and a parameter in the same file. This is discouraged, however, because it may produce unexpected results with some programs. Names may contain any alphanumeric character, as well as any of the following: `@ : # + - % . _ $ & /` . The first letter of a name may not be a digit.

2.2.1 Data Set Description

```
&description
    STRING text = NULL
    STRING contents = NULL
&end
```

This optional command describes the data set in terms of two strings. The first, `text`, is an informal description that is intended principly for human consumption. The second, `contents`, is intended to formally specify the type of data stored in a data set. Most frequently, the `contents` field is used to record the name of the program that created or most recently modified the file.

Example:

```
&description
        text = "Twiss parameters for APS lattice",
        contents = "Twiss parameters"
&end
```

Note: In many cases it is best to use a string parameter for descriptive text instead of the `description` command. The reason is that the Toolkit programs will allow manipulation of a string parameter.

2.2.2 Tabular-Data Column Definition

```
&column
    STRING name = NULL
    STRING symbol = NULL
    STRING units = NULL
    STRING description = NULL
    STRING format_string = NULL
    STRING type = NULL
    long field_length = 0
&end
```

This optional command defines a column that will appear in the tabular data section of each data page. The `name` field must be supplied, as must the `type` field. The type must be one of `short`, `long`, `float`, `double`, `character`, or `string`, indicating the corresponding C data types. The `string` type refers to a NULL-terminated character string.

The optional `symbol` field allows specification of a symbol to represent the column; it may contain escape sequences, for example, to produce Greek or mathematical characters. The optional `units` field allows specification of the units of the column. The optional `description` field provides for an informal description of the column, that may be used as a plot label, for example. The optional `format_string` field allows specification of the `printf` format string to be used to print the data (e.g., for ASCII in SDDS or other formats).

For ASCII data, the optional `field_length` field specifies the number of characters occupied by the data for the column. If zero, the data is assumed to be bounded by whitespace characters. If negative, the absolute value is taken as the field length, but leading and trailing whitespace characters will be deleted from `string` data. This feature permits reading fixed-field-length FORTRAN output without modification of the data to include separators.

The order in which successive `column` commands appear is the order in which the columns are assumed to come in each row of the tabular data.

Example (using `sddsplot` conventions for Greek and subscript operations[?]):

```
&column name=element, type=string, description="element name" &end
&column
    name=z, symbol=z, units=m, type=double,
    description="Longitudinal Position" &end
&column
    name=alphax, symbol="$ga$r$bx$n", units=m,
    type=double, description="Horizontal Alpha Function" &end
&column
    name=betax, symbol="$gb$r$bx$n", units=m,
```

```
    type=double, description="Horizontal Beta Function" &end
&column
    name=etax, symbol="$gc$r$bx$n", units=m,
    type=double, description="Horizontal Dispersion" &end
.
.
.
```

2.2.3 Parameter Definition

```
&parameter
    STRING name = NULL
    STRING symbol = NULL
    STRING units = NULL
    STRING description = NULL
    STRING format_string = NULL
    STRING type = NULL
    STRING fixed_value = NULL
&end
```

This optional command defines a parameter that will appear along with the tabular data section of each data page. The name field must be supplied, as must the *type* field. The type must be one of `short`, `long`, `float`, `double`, `character`, or `string`, indicating the corresponding C data types. The `string` type refers to a NULL-terminated character string.

The optional `symbol` field allows specification of a symbol to represent the parameter; it may contain escape sequences, for example, to produce Greek or mathematical characters. The optional `units` field allows specification of the units of the parameter. The optional `description` field provides for an informal description of the parameter. The optional `format` field allows specification of the `printf` format string to be used to print the data (e.g., for ASCII in SDDS or other formats).

The optional `fixed_value` field allows specification of a constant value for a given parameter. This value will not change from data page to data page, and is not specified along with non-fixed parameters or tabular data. This feature is for convenience only; the parameter thus defined is treated like any other.

The order in which successive `parameter` commands appear is the order in which the parameters are assumed to come in the data. For ASCII data, each parameter that does not have a `fixed_value` will occupy a separate line in the input file ahead of the tabular data.

Example:

```
&parameter name=NUx, symbol="$gn$r$bx$n",
  description="Horizontal Betatron Tune", type=double &end
&parameter name=NUy, symbol="$gn$r$by$n",
  description="Vertical Betatron Tune", type=double &end
&parameter name=L, symbol=L, description="Ring Circumference",
  type=double, fixed_value=30.6667 &end
.
.
.
```

2.2.4 Array Data Definition

```
&array
    STRING name = NULL
    STRING symbol = NULL
    STRING units = NULL
    STRING description = NULL
    STRING format_string = NULL
    STRING type = NULL
    STRING group_name = NULL
    long field_length = 0
    long dimensions = 1
&end
```

This optional command defines an array that will appear along with the tabular data section of each data page. The name field must be supplied, as must the type field. The type must be one of short, long, float, double, character, or string, indicating the corresponding C data types. The string type refers to a NULL-terminated character string.

The optional symbol field allows specification of a symbol to represent the array; it may contain escape sequences, for example, to produce Greek or mathematical characters. The optional units field allows specification of the units of the array. The optional description field provides for an informal description of the array. The optional format_string field allows specification of the printf format string to be used to print the data (e.g., for ASCII in SDDS or other formats). The optional group_name field allows specification of a string giving the name of the array group to which the array belongs; such strings may be defined by the user to indicate that different arrays are related (e.g., have the same dimensions, or parallel elements). The optional dimensions field gives the number of dimensions in the array.

The order in which successive array commands appear is the order in which the arrays are assumed to come in the data. For ASCII data, each array will occupy at least one line in the input file ahead of the tabular data; data for different arrays may not occupy portions of the same line. This is discussed in more detail below.

Example:

```
&array name=Rx, units=R-standard-units, type=double, dimensions=2,
        description="Horizontal transport matrix in standard units",
        group_name="2x2 transport matrices" &end
&array name=R-standard-units, type=string, dimensions=2,
        description="Standard units of 2x2 transport matrices",
        group_name="2x2 transport matrices" &end
&array name=P, units=P-standard-units, type=double, dimensions=1,
        description="Particle coordinate vector in standard units" &end
&array name=P-standard-units, type=string, dimensions=1,
        description="Standard units of particle coordinate vectors" &end
.
.
.
```

2.2.5 Header File Include Specification

```
&include
```

```
    STRING filename = NULL
&end
```

This optional command directs that SDDS header lines be read from the file named by the `filename` field. These commands may be nested.

Example of a minimal header:

```
SDDS1
&include filename="SDDS.twiss-parameter-header" &end
! data follows:
.
.
.
```

2.2.6 Data Mode and Arrangement Defintion

```
&data
    STRING mode = "binary"
    long lines_per_row = 1
    long no_row_counts = 0
    long additional_header_lines = 0
&end
```

This command is optional unless `parameter` commands without `fixed_value` fields, `array` commands, or `column` commands have been given.

The `mode` field is required, and may have one of the values "ascii" or "binary". If binary mode is specified, the other entries of the command are irrelevant and are ignored. In ASCII mode, these entries are optional.

In ASCII mode, each row of the tabular data occupies `lines_per_row` rows in the file. If `lines_per_row` is zero, however, the data is assumed to be in "stream" format, which means that line breaks are irrelevant. Each line is processed until it is consumed, at which point the next line is read and processed.

Normally, each data page includes an integer specifying the number of rows in the tabular data section. This allows for preallocation of arrays for data storage, and obviates the need for an end-of-page indicator. However, if `no_row_counts` is set to a non-zero value, the number of rows will be determined by looking for the occurence of an incomplete or empty line. A comment line does *not* qualify as an empty line in this sense.

If `additional_header_lines` is set to a non-zero value, it gives the number of non-SDDS data lines that follow the `data` command. Such lines are treated as comments.

2.3 Structure of SDDS ASCII Data Pages

Since the user may wish to create SDDS data sets without using the SDDS function library, a more detailed description of the structure of ASCII data pages is provided. Comment lines (beginning with an exclamation point) may be placed anywhere within a data page. Since they essentially do not exist as far as the SDDS routines are concern, I omit mention of them in what follows.

The first SDDS data page begins immediately following the `data` command and the optional additional header lines, the number of which is specified by the `additional_header_lines` parameter of the `data` command.

If parameters have been defined, then the next $N_p - N_{fp}$ lines each contains the value of a single `parameter`, where N_p is the total number of parameters and N_{fp} is the number of parameters for which the `fixed_value` field was specified. These will be assigned to the parameters in the order that the `parameter` commands occur in the header. Multi-token string parameters need not be enclosed in quotation marks.

If arrays have been defined, then the data for these arrays comes next. There must be at least one ASCII line for each array. This line must contain a list of whitespace-separated integer values giving the size of the array in each dimension. The number of values must be that given by the `dimensions` field of the `array` definition. If the number of elements in the array (given by the product of these integers) is nonzero, then additional ASCII lines are read until the required number of elements has been scanned. It is an error for a blank line or end-of-file to appear before the required elements have been scanned.

If tabular-data columns have been defined, the data for these elements follows. If the `no_row_counts` parameter of the `data` command is zero, the first line of this section is expected to contain an integer giving the number of rows in the upcoming data page. If `no_row_counts` is non-zero, no such line is expected. The remainder of the tabular data section has various forms depending on the parameters of the data command, as discussed above. The default format is that each line contains the whitespace-separated values for a single row of the tabular data.

For column and array data, string data containing whitespace characters must be enclosed in double-quotes. For column, array, and parameter data, nonprintable character data should be "escaped" using C-style octal sequences.

More than one data page may appear in the data set. Subsequent data pages have the same structure as just described. If `no_row_counts=1` is given in the `data` command, then a blank or incomplete line is taken to end each data set.

References

[1] M. Borland, "Application Programmer's Guide for SDDS Version 1.4", APS LS Note.

[2] M. Borland, "User's Manual for `elegant`", APS Light Source Note.

[3] M. Borland, "A Self-Describing File Protocol for Simulation Integration and Shared Postprocessors", to appear in *Proceedings of the 1995 Particle Accelerator Conference*, May 1995, Dallas.

[4] M. Borland, "A High-Brightness Thermionic Microwave Gun", Stanford Ph.D. Thesis, 1991, Appendix A.

[5] L. Emery, "Commissioning Software Tools at the Advanced Photon Source", to appear in *Proceedings of the 1995 Particle Accelerator Conference*, May 1995, Dallas.

[6] L. Emery, "Beam Simulation and Radiation Dose Calculation at the Advanced Photon Source with `shower`, an EGS4 Interface", to appear in *Proceedings of the 1995 Particle Accelerator Conference*, May 1995, Dallas.

[7] M. Abramowitz and I. A. Stegun, eds., *Handbook of Mathematical Functions*, Dover Publications, New York, 1965.

3 Manual Pages Overview

The intention is of this section is to provide a means by which the reader can select programs that might suit a given need. For each program, a brief (and usually incomplete) description is given, along with example applications. The example applications provided for each tool are drawn from experience at APS; it is hoped that most will make sense to most readers.

This section is followed by manual pages that give detailed descriptions of each program. Many of the programs have a large number of switches, most of which are optional. In order to help the new user, actual commandline examples are provided for simple use of each program. After understanding these, the user is in a good position to explore the additional capabilities provided by the options.

Note that many of the Toolkit programs process tabular data only (i.e., columns). To use these programs with parameter data, one can use `sddscollapse` to convert parameter data into tabular data. Using pipes will make this more convenient.

Support for SDDS array elements is presently rather sparse in the Toolkit. This reflects the fact that almost all data can be conveniently stored using parameter and column elements. Hence, work has concentrated on providing tools that manipulate such data. Future versions of the Toolkit will provide more array support.

Most of the Toolkit programs process data pages sequentially. That is, in many cases the requested processing is performed on each successive page of the input file and delivered to successive pages of the output file.

3.1 SDDS Toolkit Programs by Category

3.1.1 Data Analysis Tools

- `sddschanges` (4.5) — Analyzes changes in column data from page to page in a file, relative to a reference file or the first page. Example application: finding changes in a waveform that is acquired repeatedly, where successive waveforms are on successive pages.
- `sddscorrelate` (4.12) — Computes correlation coefficients and correlation significance between column data. Example application: finding correlations among time series data collected from process variables, and evaluating their signficance to find possible cause-and-effect relationships.
- `sddsderiv` (4.13) — Does numerical differentiation of multiple data columns versus a single column, with optional error propogation.
- `sddsdigfilter` (4.14) — Performs time-domain digital filtering of column data. Example applications: low pass, high pass, band pass, or notch filtering of data to eliminate unwanted frequencies.
- `sddsenvelope` (4.15) — Analyzes column data across pages to find minima, maxima, averages, standard-deviations, etc., on a row-by-row basis. Example application: finding the envelope and average of a set of waveforms.
- `sddsfft` (4.17) — Does Fast Fourier Transforms of column data. Example application: finding signficant frequency components in time-varying data, or finding the integer tune of an accelerator from a difference orbit.

- sddshist (4.19) — Does histograms of column data. Example application: finding the distribution of a readback that is sampled many times, or of particle coordinates from an accelerator tracking simulation.

- sddshist2d (??) — Does two-dimensional histograms of column data. Example applications: finding the two-dimensional distribution of a pair of readbacks that are sampled many times. or of two particle coordinates (e.g., x and y position) from an accelerator tracking simulation.

- sddsinteg (4.20) — Does numerical integration of multiple data columns versus a single column, with optional error propogation. Example application: finding the field integral an accelerator magnet from a longitudinal field scan.

- sddsinterp (4.21) — Does interpolation of multiple data columns as a function of a single column. Example application: finding the required current to obtain a desired excitation in a magnet, or interpolating a curve at positions given in a second file.

- sddsoutlier (4.22) — Eliminates statistical outliers from data. Example application: eliminating bad or nonrepresentative data points prior to searching for correlations with sddscorrelate, or computing statistics with sddsprocess.

- sddspeakfind (4.23) — Finds values of columns at locations of peaks in a single column. Example application: finding the position and height of peaks in a power spectrum obtained from a FFT.

- sddsprocess (4.27) — Probably the most-used toolkit program, excepting sddsplot. Allows creating new parameters and columns with user-specified equations; filtering and matching operations; printing, editing, scanning, and subprocess operations; statistical and waveform analysis of column data to produce new parameters; and much more.

- sddssmooth (4.33) — Smooths columns of data using multipass nearest-neighbor averaging. Example application: reducing noise in a frequency spectrum prior to finding peaks.

- sddszerofind (4.39) — Finds values of columns at locations of interpolated zeroes in a single column. Example application: finding zeros of a tabulated function that isn't known analytically.

3.1.2 Data Fitting Tools

- sddsexpfit (4.16) — Does an exponential fit to column data. Example application: finding the exponential lifetime of a beam in a storage ring, or the half-life a radioactive sample.

- sddsgfit (4.18) — Does gaussian fits to column data. Example application: finding the width of a resonance, or the rms size of a beam profile.

- sddspfit (4.24) — Does polynomial fits to column data, including error analysis. Will do fits to specified orders, fits of specified symmetry, and adaptive fitting.

3.1.3 Data Manipulation Tools

- sddsbreak (4.4) — Breaks data pages into new, separate pages based on changes in column data and other criterion. Example applications: reorganizing a file to have a limited number of rows in each page, or to have a new page started when a gap is seen in the data.

- sddscollapse (4.7) — Collapses a data set into a single data page by deleting the tabular data and turning the parameters into columns. Example application: abstraction of summary properties of data set following analysis with sddsprocess.

- sddscombine (4.8) — 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. Example application: bringing together comparable but distinct data for analysis with sddsprocess. Using sddsprocess, sddscombine, and sddscollapse in sequence repeatedly is a powerful way to analyze and collate large amounts of data.

- sddsconvert (4.11) — Allows conversion of a data set between binary and ASCII, with optional deletion and renaming of columns, arrays, and parameters . Example application: conversion to binary of an ASCII data set created by a simple program, or by a text editor.

- sddsselect (4.31) — Copies rows from one file based on the presence or absence of matching data in another file. Example application: finding all of the rows from one file that do not appear in a second file.

- sddssort (4.34) — Sorts the tabular data section of a data set by the values in named columns. Optionally eliminates duplicate rows.

- sddssplit (4.35) — Places each page of a file in a separate, new file. Example application: getting selected pages of a file into separate, single-page files for use with a program that only recognizes the first page.

- sddsxref (4.38) — Creates a new data set by adding selected rows from one data set to another data set. Example application: cross-referencing the turn-by-turn coordinates of particles in a tracking simulation with the initial coordinates using a particle ID number.

3.1.4 Graphics Tools

- sddscongen (4.9) — Creates an SDDS data set by evaluating an rpn expression over a defined 2 dimensional grid. Example application: generating values of a function of two variables on a grid for plotting with sddscontour.

- sddscontour (4.10) — Makes contour and color-map plots from an SDDS data set column, or from a rpn expression of the values in the columns of a data set. Supports FFT interpolation and filtering. Example application: displaying data from a two-dimensional magnetic field scan.

- sddsplot (4.25) — A highly flexible, device-independent graphics program, equally capable of "quick-and-dirty" or publication quality graphics. Example application: making an X-windows movie of several columns of data that change from page to page in a file.

3.1.5 File Protocol Conversion Tools

- awe2sdds (5.1) — Converts a file in awe self-describing format[4] to SDDS.

- col2sdds (5.2) — Converts a file in column self-describing format to SDDS.

- citi2sdds (??) — Converts Hewlett-Packard CITI files to SDDS.

- hpif2sdds (??) — Converts Hewlett-Packard HP54542 scope internal format to SDDS.

- hpwf2sdds (??) — Converts Hewlett-Packard HP54542 scope text format to SDDS.
- lba2sdds (??) — Converts Spiricon Laser Beam Analyzer files to SDDS.
- mpl2sdds (5.4) — Adds mpl data files[4] to an SDDS data set.
- sdds2math (4.1) — Converts SDDS data to a format accepted by Mathematica.
- sdds2mpl (5.3) — Extracts data columns or parameters from an SDDS data set and creates mpl data files[4].
- sdds2spreadsheet (4.2) — Converts SDDS data to a format accepted by the Excel and Wingz spreadsheets.

3.1.6 Text-based Data-review Tools

- sdds2stream (4.3) — Takes column or parameter data from a list of SDDS data sets and delivers it to the standard output as a stream of values. Example application: getting data into a shell variable for use in a script.
- sddsprintout (4.26) — Makes customized printouts from column, parameter, and array data in an SDDS data set. Example application: making a nicely-formatted printout of data that needs to be reviewed manually.
- sddsquery (4.29) — Prints a summary of the SDDS header for a data set. Also prints bare lists of names of defined entities, suitable to use with shell scripts that need to detect the existence of entities in the data set.

3.2 Toolkit Program Usage Conventions

In order to make the multitude of Toolkit programs easier to use, the developers have attempted to use consistent commandline argument styles. The Toolkit programs all require at least one commandline argument. Therefore, if a program is executed without commandline arguments, it is assumed that the user is asking for help. In this case, a help message is printed that shows syntax and (usually) describes the meaning of the switches. In general, program usage is of the following form:

programName *fileNames switches.*

Probably the simplest example would be

sddsquery *fileName*,

which would invoke sddsquery to describe the contents of an SDDS file. A slightly more complicated example would be

sddsquery *fileName* -columnList,

which invokes sddsquery to list just names of columns in a file.

Programs assume that any commandline argument beginning with a minus sign ('-') is an option; all others are assumed to be filenames. Note that case is ignored in commandline switches. The specific meaning of a filename is dictated by its order on the commandline. For example, if two filenames are given, the first would commonly be an input file while the second would commonly be an output file.

In some cases, a command with a single filename implies replacement of the existing file. For example,

sddsconvert *fileName* -binary

would replace the named file with a binary version of the same data. This command is completely equivalent to

`sddsconvert -binary` *fileName*

That is, unlike many UNIX commands, the position of filenames relative to options is irrelevant.

One might also wish to make a new file, rather than replacing the existing file. This could be done by

`sddsconvert -binary` *fileName fileName2*

Note that while the option may appear anywhere on the commandline, the order of the filenames is crucial to telling the program what to do.

In following manual pages and in the program-generated help text, program usage is described using the following conventions:

- The first token on the commandline is the name of the program.
- Items in square-brackets ([]) are optional. Items not in square brackets are required.
- Items in curly-brackets ({}) represent a list of choices. The choices are separated by a | character, as in
  { *choice1* | *choice2* | *choice3* }
- Items in italics are descriptions of arguments or data that must be supplied by the user. These items are not typed literally as shown.
- Items in normal print are typed as shown, with optional abbreviation. These are usually switch keywords or qualifiers. Any unique abbreviation is acceptable.

In addition to using files, most toolkit programs also take input from pipes, which obviates the need for temporary files in many cases. For those programs that support pipes, one can employ the `-pipe` option. This option provides a good example of what options look like. For example, one could do the following to test binary-ascii conversion:

`sddsconvert -binary -pipe=out` *fileName* `| sddsconvert -ascii -pipe=in` *fileName1*

The `-pipe=out` option to `sddsconvert` tells it to deliver its output to a pipe; it still expects a filename for input. Similarly, the `-pipe=in` option to `sddsquery` tells it to accept input from a pipe.

The `-pipe` switch may be given in one of five forms: `-pipe`, `-pipe=input,output`, `-pipe=output,input`, `-pipe=input`, `-pipe=output` . The first three forms are equivalent. In a usage message, these forms would be summarized as `-pipe[=input][,output]`. One could also use abbreviations like `-pipe=i`, `-pipe=i,o`, etc. For convenience in the manual, the data stream from or to a pipe will often be referred to by the name of the file for which it substitutes. Note that you may not deliver more than one file on the same pipe.

3.3 Data for Examples

In order to make examples simpler to present, it helps to have hypothetical data files to refer to. I will assume the existence of several data files that I hope will be familiar to many readers. An ASCII version of each file is provided in the SDDS distribution package. This gives new users some data to "play with" in getting familiar with SDDS. These files are also used in several demonstration scripts provided in the package.

For each file, I've listed the names of the columns and parameters, and described each. I've given the data types in detail, even though only the distinction between numerical and nonnumerical data

is relevant, just to emphasize that data types can be freely mixed. I've tried to include as little data as is necessary to make useful demonstrations, without simplifying so much as to be trivial.

3.3.1 Twiss Parameters

The example of Twiss parameters for an accelerator is a familiar one. Throughout these pages, it is assumed that two files, APS0.twi and `APS.twi`, exist containing the following data (a simplification of the Twiss output from the accelerator simulation code `elegant`):

- Parameters:
  - nux, nuy – Double-precision values of the x and y tunes.
  - `alphac` — Double-precision values of the momentum compaction factor.
- Columns:
  - `s` – A double precision column of element positions. For simplicity, it is assumed to increase monotonically through the file.
  - `ElementName` – A string column of element names.
  - `ElementType` – A string column of element type identifiers.
  - `betax`, `betay` — Double-precision columns of the beta functions for the horizontal and vertical planes, respectively.
  - `psix`, `psiy` — Double-precision columns of the betatron phase advance.
  - `etax`, `etay` — Double-precision columns of the dispersion functions.

To make it more interesting, `APS0.twi` is a single-page file containing the APS design lattice, while `APS.twi` is a multi-page file with each page corresponding to a different configuration.

In passing, it is appropriate to mention the style of the names used. It has been found helpful to use capitalization at word boundaries to make long names more readable. (In some cases, like `betax`, a certain case is used because it is significant.) When doing so will not create confusion, we also tend to capitalize the first letter of a name, which helps the name to stand out on the command line. Abiding by these conventions tends to result in readable names being created by Toolkit programs that have automatic name generation. Underscores in names are avoided because they increase the length of a name while adding less readability than capitalization.

3.3.2 Data Logging Over Time

One of the most common applications of SDDS for APS commissioning and operation is logging of measured data values at intervals. A set of generic EPICS monitoring programs `sddsmonitor`, `sddsvmonitor` (vector monitoring), and `sddswmonitor` (waveform monitoring) are used for this. One example is the vacuum pressure in the APS ring, which is logged continuously by `sddsvmonitor`; this data consists of readings from ion gauges around the ring. Another example is logging of beam-position-monitor readouts in the Positron Accumulator Ring (PAR) and its input and output beam transport lines using the program `sddsmonitor`.

For use in examples, I'll assume the existence of two files called `SR.vac` and `par.bpm`. These are simplified from actual files collected with the programs just mentioned.

`SR.vac` is a file containing an arbitrary series of data pages, each consisting of a snapshot of the vacuum gauge readings around the ring. There are 40 such readings, one for each sector of the accelerator. Typically, one set of readings is taken every 15 minutes.

- Parameters:
  - `TimeStamp` — A string parameter containing the time at which the snapshot was taken.
  - `TimeOfDay` — A double-precision parameter containing the time of day in hours since midnight.
- Columns:
  - `Index` — A long-integer column containing the row index.
  - `SectorName` — A string column containing the name of the sector each row corresponds to.
  - `Pressure` — A double-precision column containing the pressure readout from the gauges at the time given by `TimeStamp`.

`par.bpm` is a file containing a single page of data with any arbitrary number of rows. The PAR has 16 beam-position-monitors (BPMs), each providing a horizontal (x) and vertical (y) readout. In addition, the beam transport line downstream of PAR (known as the PTB line), contains five BPMs for x and five for y. The data included in the distribution contains only the x values, since these are more interesting:

- Parameters:
  - `TimeStamp` — A string parameter giving the starting time of the data collection.
- Columns:
  - `Time` — A double-precision column giving the elapsed number of seconds since monitoring begain. The values are approximately equispaced.
  - `TimeOfDay` — A double-precision column giving the time of day in hours since midnight.
  - P*quadrant*P*number*x — 16 single-precision readouts of the horizontal beam orbit just prior to beam extraction. *quadrant* ranges from 1 to 4, as does *number*.
  - `PTB:PH`*number*x — four single-precision readouts of the horizontal beam trajectory as the beam passes through the PTB transfer line. *number* ranges from 2 to 5.

4 Manual Pages

Manual pages are written by the program author unless otherwise noted.

4.1 sdds2math

- **description:** sdds2math converts an SDDS file to a file that can be read into Mathematica. The file contains a single Mathematica variable of the form:

```
sdds={description,coldef,pardef,arraydef,associates,tables}
  description={text,contents}
  coldef={coldef-1, coldef-2, ...}
    coldef-n={name,units,symbol,format,type,fieldlength,description}
  pardef={pardef-1, pardef-2, ...}
    pardef-n={name,fixed_value,units,symbol,type,description}
  arraydef={arraydef-1, arraydef-2, ...}
    arraydef-n={name,units,symbol,format,type,fieldlength,group,description}
  associates={associate-1, associate-2,...}
    associate-n={sdds,filename,path,contents,description}
  tables={table-1, table-2, ...}
    table-n={parameters,data}
      parameters={parameter-1, parameter-2, ...}
      data={row-1, row-2, ...}
        row-n={val-1, val-2, ...}
```

A number of Mathematica programs to extract information from this variable are available in the file SDDS.m. To include these routines in your Mathematica program, put this file in your working directory and use the following line in your Mathematica program:

```
Needs["SDDS`"];
```

The programs are:

- `SDDSRead[filename_String]`—returns an SDDS structure from a file.
- `SDDSWrite[sdds_,filename_String]`—writes an SDDS structure to a file.
- `SDDSGetColumnDefinitions[sdds_]`—returns the list of column definitions.
- `SDDSGetParameterDefinitions[sdds_]`—returns the list of parameter definitions.
- `SDDSGetArrayDefinitions[sdds_]`—returns the list of array definitions.
- `SDDSGetAssociates[sdds_]`—returns the list of associates.
- `SDDSGetTable[sdds_,n_:1]`—returns the nth table parameters,data.
- `SDDSGetParameters[sdds_,n_:1]`—returns the parameters from the nth table.
- `SDDSGetParameter[sdds_,p_String,n_:1]`—returns the value of parameter p from the nth table.
- `SDDSGetData[sdds_,n_:1]`—returns the data matrix from the nth table.
- `SDDSGetColumn[sdds_,c_String,n_:1]`—returns the column named c from the nth table.
- `SDDSGetColumn[sdds_,m_,n_:1]`—returns the mth column from the nth table.
- `SDDSGetRow[sdds_,m_,n_:1]`—returns the mth row from the nth table.
- `SDDSGetNColumns[sdds_]`—returns the number of columns.

- SDDSGetNParameters[sdds_]—returns the number of parameters.
- SDDSGetNArrays[sdds_]—returns the number of arrays.
- SDDSGetNAssociates[sdds_]—returns the number of associates.
- SDDSGetNTables[sdds_]—returns the number of tables.
- SDDSGetNRows[sdds_,n_:1]—returns the number of rows in the nth table.
- SDDSGetColumnNames[sdds_]—returns the list of column names.
- SDDSGetParameterNames[sdds_]—returns the list of parameter names.
- SDDSGetArrayNames[sdds_]—returns the list of array names.
- SDDSGetAssociateNames[sdds_]—returns the list of associate names.

- **examples:** Convert a snapshot to a Mathematica file.

  `sdds2math par.050695.snap par.050695.m`

- **synopsis:**

  `sdds2math` *SDDSfilename outputname* `[-comments] [-verbose] [-format=`*printfString*`]`

- **switches:**
  - `comments` — Put helpful Mathematica comments in the file.
  - `verbose` — Write header information to the terminal like sddsquery.
  - `format` — Format for doubles (Default: %g)

- **author:** K. Evans, Jr., ANL/APS.

4.2 sdds2spreadsheet

- **description:** `sdds2spreadsheet` converts an SDDS file to a file that can be read into most spreadsheet programs. You need to consult your particular spreadsheet program to see how it reads ASCII files. For Wingz, the conversion is automatic. Excel 5.0 will bring up its Text Import Wizard.

  Note: Excel lines must be shorter than 255 characters. The Wingz delimiter can only be `\t`.

- **examples:** Convert a snapshot to a Wingz spreadsheet.

  ```
  sdds2spreadsheet par.050695.snap par.050695.wkz
  ```

  Convert a snapshot to an Excel text file.

  ```
  sdds2spreadsheet par.050695.snap p050695.txt
  ```

- **synopsis:**

  `sdds2spreadsheet` *SDDSfilename outputname* `[-delimiter=`*string*`] [-all] [-verbose]`

- **switches:**
  - `delimiter` — Delimiter string (Default is "").
  - `all` — Write parameter, column, array, associate information, too. (Default is data and parameters only)
  - `verbose` — Write header information to the terminal like sddsquery.

- **author:** K. Evans, Jr., ANL/APS.

4.3 sdds2stream

- **description:**
  sdds2stream provides stream output to the standard output of data values from a group of columns or parameters. Each line of the output contains a different row of the tabular data or a different parameter. Values from different columns are separated by the delimiter string. If -page is not employed, all data pages are output sequentially. If multiple filenames are given, the files are processed sequentially in the order given.

- **examples:** To output values of tunes for each page, one line per page:

  ```
  sdds2stream APS.twi -parameters=nux,nuy -delimiter=" "
  ```

  To output values of columns ElementName and betax for the first data page:

  ```
  sdds2stream APS.twi -column=ElementName,betax -page=1
  ```

- **synopsis:**

  sdds2stream {*inputFileList* | -pipe[=input]} [-page=*pageNumber*]
  [-delimiter=*delimitingString*] { -columns=*columnName*[,*columnName*...] |
  -parameters=*parameterName*[,*parameterName*...] } [-filenames] [-rows]
  [-noquotes]

- **files:** *inputFileList* is a space-separated list of SDDS filenames.

- **switches:**

  - -pipe[=input] — The standard SDDS Toolkit pipe option.
  - -page=*page-number* — Specifies the number of the data page for which output is desired. Recall that pages are numbered sequentially beginning with 1. More complete control of which pages are output may be obtained using sddsconvert or sddsprocess as a filter.
  - -delimiter=*delimitingString* — Specifies the delimiting string to be printed to separate row entries or parameters. The delimiter is printed with printf, so that any of the usual escape sequences may be employed.
  - columns=*columnName*[,*columnName*...] — Specifies the names of the columns for which output is desired. For each row of each data page, the specified columns are printed on a single line, separated by the delimiting string. The default delimiting string is a single space.
  - -parameters=*parameterName*[,*parameterName*...] — Specifies the names of the parameters for which output is desired. For each row of each data page, the specified parameters are printed on a single line, separated by the delimiting string. However, since the default delimiting string is a newline, the parameters end up on separate lines.
  - filenames — Specifies that the filename will be printed out as each file is processed.
  - rows — Specifies that the number of rows per page for the tabular data section will be printed out.

- noquotes — Specifies that whitespace-containing string data will be printed without the default double-quotes.

- **see also:**
  - Data for Examples (see 3.3)
  - `sddsprintout` (4.26)
  - `sddsconvert` (4.11)
  - `sddsprocess` (4.27)
- **author:** M. Borland, ANL/APS.

4.4 sddsbreak

- **description:** sddsbreak reads pages from an SDDS file and writes a new SDDS file containing the same data, but with each of the input pages potentially separated into several output pages. The separation involves breaking each input page at one or more locations as determined by one of several user-defined criteria.

- **examples:** Limit the length of pages to 500 rows so that data may be viewed more easily:

  ```
  sddsbreak par.bpm par.bpm1 -rowlimit=500
  ```

  Break the page whenever a gap of more than 15 seconds is seen:

  ```
  sddsbreak par.bpm par.bpm1 -gapin=Time,amount=15
  ```

- **synopsis:**

  sddsbreak [-pipe=[input][,output]] [*inputFile*] [*outputFile*]
  { -gapIn=*columnName*[,{amount=*value* | factor=*value*}] |
  -increaseOf=*columnName* | -decreaseOf=*columnName*
  -changeOf=*columnName*[,amount=*value*[,base=*value*]]
  -rowLimit=*integer* }

- **files:** *inputFile* is an SDDS file containing one or more pages of data to be broken up. *outputFile* is an SDDS file in which the result is placed. Each page of *outputFile* contains the parameter and array values from the page of *inputFile* that is its source.

- **switches:**

  - -pipe=[input][,output] — The standard SDDS Toolkit pipe option.
  - -gapIn=*columnName*[,{amount=*value* | factor=*value*}] — Breaks the page when the value in the named column has a gap. If the amount qualifier is given, then a gap is defined as any occurence of successive values different by more than *value*. If this qualifier is not given, then the *value* is computed as follows: the mean absolute difference (MAD) between successive values for the first page which has more than 1 row is computed; if the factor qualifier is given, then the gap amount is the MAD times the given value; otherwise, it is the MAD times two.
  - -increaseOf=*columnName*, -decreaseOf=*columnName*— These options cause a page break whenever the value in the named column increases or decreases, respectively.
  - -changeOf=*columnName*[,amount=*value*[,base=*value*]] — Breaks the page when the value in the named column changes. If the amount qualifier is not given, then any change is sufficient to break the page. Otherwise, the page is broken whenever the quantity $\lfloor (V - B)/A \rfloor$ changes, where V is the value in the column, A is the value given for amount, and B is the value given for base. If base is not given, then the value in first row for the column is used.
  - -rowLimit=*integer* — Breaks the page after the specified number of rows.

- **see also:**

  - Data for Examples (see 3.3)

- `sddscombine` (4.8)

- **author:** M. Borland, ANL/APS.

4.5 sddschanges

- **description:** `sddschanges` analyzes changes in column data from page to page in a file, relative to reference data in a baseline file or from the first page. It requires that every page in the file have the same number of rows. It produces a multipage output file containing the row-by-row difference between the reference data and the data each page in the input file.

- **examples:** Compute the changes in the dispersion function for several APS lattices:

  ```
  sddschanges APS.twi APS.changes -copy=s -changeIn=betax,betay,etax
  ```

  The output file in this example would have one fewer pages than the input file. Each page would contain the column s from the first page, along with the differences from the first page for betax, betay, and etax. One could also compute the changes relative to the nominal lattice:

  ```
  sddschanges APS.twi -baseline=APS0.twi APS.changes -copy=s
  -changeIn=betax,betay,etax
  ```

  The output file would have one page for every page in the input.

- **synopsis:**

  `sddschanges [-pipe=[input][,output]]` [*inputFile*] [*outputFile*]
  `[-copy=`*columnNames*`]` `[-changesIn=`*columnNames*`]` `[-baseline=`*referenceFileName*`]`

- **files:** *inputFile* is a multipage file containing the data for which changes are desired. *outputFile* is a multipage file containing the changes. The column names in *outputFile* for the changes are created from those in *inputFile* by prepending the string "ChangeIn".

- **switches:**

  - `-pipe=[input][,output]` — The standard SDDS Toolkit pipe option.
  - `-copy=`*columnNames*— Specifies that the named columns should be transferred to the output file without alteration. These data come from the baseline file or from the first page of the input file. A comma-separated list of optionally wildcard-containing strings may be given.
  - -changesIn=*columnNames* — Specifies that the named columns should be transferred to the output file after subtracting the corresponding values from the baseline file or from the first page of the input file. A comma-separated list of optionally wildcard-containing strings may be given.
  - `-baseline=`*referenceFileName* — Specifies the name of an SDDS file from which the reference data for changes should be taken.

- **see also:**

  - Data for Examples (see 3.3)
  - `sddsenvelope` (4.15)

- **author:** M. Borland, ANL/APS.

4.6 sddscheck

- **description:** `sddscheck` is a simple tool to allow checking a file to see if it is a valid SDDS file, or if it is corrupted. The primary use is in shell scripts that need to detect such conditions. `sddscheck` issues one of four messages: `ok`, `nonexistent`, `badHeader`, or `corrupted`. (See `sddsconvert` (4.11) about recovering corrupted files.)

- **examples:** Under UNIX, one could do the following to check a file before plotting it:

  ```
  if (`sddscheck APS.twi` == "ok") plotTwissParameters APS.twi
  ```

  where `plotTwissParameters` is a hypothetical plotting script.

- **synopsis:**

  `sddscheck` *filename*

- **files:** *filename* is the name of a single file to be checked.

- **see also:**

  - progrefsddsconvert

- **author:** M. Borland, ANL/APS.

4.7 sddscollapse

- **description:** `sddscollapse` reads data pages from an SDDS file and writes a new SDDS file containing a single data page. This data page contains only the values of the parameters from the original file, with each parameter forming a column of the tabular data.

- **examples:** To create a new file containing the tunes and other parameters as columns:

  ```
  sddscollapse APS.twi APS.parameters
  ```

  To do a polynomial fit to nux as a function of nuy, and print the results out:

  ```
  sddscollapse APS.twi -pipe=out | sddspfit -pipe=in fit.sdds -column=nux,nuy
  -verbose
  ```

- **synopsis:**

  `sddscollapse [`*inputFile*`] [`*outputFile*`] [-pipe[=input][,output]]`

- **files:** *inputFile* is the name of an SDDS data set to be collapsed. *outputFile* is the result. Note that *outputFile* will not contain any information on the arrays or columns that are in *inputFile.*

- **comment:** In spite of the simplicity of the commandline, this is an extremely useful program. A typical use might involve processing a multipage file using `sddsprocess` to, for example, obtain statistical analyses of columns for each page; the results of such analyses are placed in parameters. Using `sddscollapse` on this file would produce columns of statistical analyses, with one row for each page. One might then further analyze the data using `sddsprocess`. One could also use `sddscombine` to combine several collapsed, processed data sets into a single file, which puts one formally back in the same position as when one started. In this fashion, multi-level data analysis and collation is possible. This is done with some magnetic measurements at APS.

- **see also:**
  - Data for Examples (see 3.3)
  - `sddsprocess` (4.27)
  - `sddscombine` (4.8)

- **author:** M. Borland, ANL/APS.

4.8 sddscombine

- **description:** sddscombine combines data from a series of SDDS files into a single SDDS file with one page for each page in each file. Data is added from files in the order that they are listed on the command line. All of the data files must contain the columns and parameters contained by the first; the program ignores any columns or parameters in a subsequent data file that are not in the first data file.

- **example:** Combine several Twiss parameter files into one file, keeping page boundaries separate.

```
sddscombine APS1.twi APS2.twi APS3.twi APSall.twi
```

- **synopsis:**

```
sddscombine [inputFileList] [outputFile] [-pipe[=input][,output]]
[-merge[=parameterName]] [-overWrite]
```

- **files:** *inputFileList* is a list of space-seperated filenames to be combined. *outputFile* is a filename into which the combined data is placed. If no -pipe options are given, the *outputFile* is taken as the last filename on the commandline. To specify an output file with input from a pipe, one uses sddscombine -pipe=input *outputFile.* Similarly to specify output to a pipe with many input files, use sddscombine -pipe=output *inputFileList.* Since accidentally leaving off the -pipe=output option for the last command might result in replacement of an intended input file, the program refuses to overwrite an existing file unless the -overWrite option is given. A string parameter (Filename) is included in outputFile to show the source of each page.

- **switches:**
  - -pipe[=input][,output] — The standard SDDS Toolkit pipe option.
  - -merge[=*parameterName*] — Specifies that all pages of all files are to be merged into a single page of the output file. If a *parameterName* is given, successive pages are merged only if the value of the named parameter is the same.
  - overWrite — Forces sddscombine to overwrite *outputfile* if it exists.

- **see also:**
  - Data for Examples (see 3.3)

- **author:** M. Borland, ANL/APS.

4.9 sddscongen

- **description:** Creates an SDDS data set by evaluating an rpn expression over a defined 2 dimensional grid. This data set may be plotted using sddscontour.

- **example:** This will generate a two-dimensional color-shaded map of the function $\sin(4\pi(x^2 + y^2))$ on the region x:[-1, 1] and y:[-1, 1]:

```
sddscongen example.sdds -xRange=-1,1,101 -yRange=-1,1,101
-zEquation="x x * y y * + 4 * pi * sin"
sddscontour example.sdds -shade example.sdds -equalAspect
```

- **synopsis:**

sddscongen *outputfile* -xRange=*lower*,*upper*,*nPoints* -yrange=*lower*,*upper*,*nPoints*
-zEquation=*rpnExpression* [-rpnCommand=*rpnExpression*]
[-rpnDefinitions=*rpn-defnsFile*]

- **switches:**
  - xRange=*lower*,*upper*,*nPoints*, yRange=*lower*,*upper*,*nPoints* — Specifies the 2 dimensional grid over which data is generated. x is the horizontal variable and y the vertical.
  - -zEquation=*rpnExpression* — Specifies the rpn expression that is evaluated at each point of the grid.
  - -rpnCommand=*rpnExpression* — Specifies the name of a file containing rpn input. The named file is read before any other operations are performed.
  - -rpnDefinitions=*rpn-defnsFile* — Specifies a string to submit to rpn prior to beginning evaluation of the equation on the grid.

- **see also:**
  - sddscontour (4.10)
  - rpn (4.41)

- **author:** M. Borland, ANL/APS.

4.10 sddscontour

- **description:** sddscontour makes contour and color-map plots from an SDDS data set column, or from a rpn expression in terms of the values in the columns of a data set. It supports FFT interpolation and filtering. If the data set contains more than one data page, data from successive pages is plotted on separate pages.

  The file must contain tabular data with at least one numeric column, which will be organized into a 2d array with R rows and C columns. By default, the values are assumed to come in row-major order (i.e., the file should contain a series of R sequences each containing the C values of a single row). The parameters of the 2d grid over which the plot is to be made are communicated to the program in one of two ways:

  1. The string parameters `Variable1Name` and `Variable2Name` contain the names of the x and y axis variables, which I'll represent as x and y respectively. The program expects to find six more parameters, with names x`Minimum`, x`Interval`, and x`Dimension`, and similarly for y. These parameters must be numeric, and contain the minimum value, the interval between grid points, and the number of points, respectively, for the dimension in question.
  2. The numeric parameters `NumberOfRows` and `NumberOfColumns` contain the values of R and C, respectively.

- **example:** This will generate a two-dimensional color-shaded map of the function $\sin(4\pi(x^2+y^2))$ on the region x:[-1, 1] and y:[-1, 1]:

```
sddscongen example.sdds -xRange=-1,1,101 -yRange=-1,1,101
-zEquation="x x * y y * + 4 * pi * sin"
sddscontour example.sdds -shade example.sdds -equalAspect
```

- **synopsis:**

  sddscontour *SDDSfilename switches*

- **switches:**

  - Choice of what to plot:

    [-quantity=*columnName* | -equation=*rpnExpression*]

    * quantity — Specifies the name of the column to make a contour or color map of.
    * equation — Specifies a rpn expression to make a contour or color map of. The expression may refer to the values in the columns by the appropriate column name, and may also refer to the variable values by name.

  - rpn control:

    [-rpnDefinitionsFiles=*filename*[,*filename*...]]
    [-rpnExpressions=*setupExpression*[,*setupExpression*...]]

    * rpnDefinitionsFiles — Specifies the names of files containing rpn expressions to be executed before any other processing takes place.
    * rpnExpressions — Specifies rpn expressions to be executed before any other processing takes place, immediately after any definitions files.

– Shade and contour control:

{-shade=*number*[,*min*,*max*] | -contours=*number*[,*min*,*max*]}
[-labelContours=*interval*[,*offset*]]

* shade — Specifies that a color (or grey-scale) map should be produced, with the indicated *number* of shades mapped onto the range from *min* to *max*. If *min* and *max* are not given, they are taken to be equal to the minimum and maximum data values.
* contours — Specifies that contour lines should be drawn, with the indicated *number* of lines for the range from *min* to *max*. If *min* and *max* are not given, they are taken to be equal to the minimum and maximum data values.
* labelContours — Specifies that every $interval^{th}$ contour line, starting with the $offset^{th}$ line, should be labeled with the contour value.

– Image processing:

[-interpolate=*nx*,*ny*[,floor | ceiling | antiripple]]
[-filter=*xcutoff*,*ycutoff*]

* interpolate — Specifies that the 2d map should be interpolated to have *nx* times more rows (or x grid points) and *ny* times more columns (or y grid points). Since FFTs are used to do the interpolation, the original number of grid points must be a power of.2, as must the factor. Giving a factor of 1 disables interpolation for the dimension in question. floor, ceiling, and antiripple specify image processing of the interpolated map. floor and ceiling respectively force values below (above) the minimum (maximum) value of the data to be set equal to that value. antiripple causes the map to be altered so that non-zero values in the new map between zero values on the original map are set to zero; this suppresses ripples that sometimes occur in regions where the data was originally all zero.
* filter — Applies low-pass filters to the data with the specified normalized cutoff frequencies. The integer cutoff values give the number of frequencies starting at the Nyquist frequency that are to be eliminated.

– Plot labeling:

[-xLabel=*string*] [-yLabel=*string*] [-title=*string*] [-topline=*string*]
[-topTitle] [-noLabels] [-noScales] [-dateStamp]

* xLabel, yLabel, title, topline — These specify strings to be placed in the various label locations on the plot.
* topTitle — Requests that the title label be placed at the top of the plot, rather than at the bottom.
* noLabels — Requests that no labels be placed on the plot.
* noScales — Requests omission of the numeric scales.
* noBorder — Requests omission of the border around the data. Implies -no_scales.
* dateStamp — Requests that the date and time be placed on the pot.

– Miscellaneous plot control:

[-scales=*xl*,*xh*,*yl*,*yh*] [-device=*name*[,*deviceArguments*]] [-swapxy]
[-equalAspect[=-1,1]] [-noBorder]

* scales — Specifies the extent of the plot region.

* `device` — Specifies the device name and optional device-specific arguments. See the `mpl` user's manual for details[?].
* `swapxy` — Requests that the horizontal and vertical coordinates be interchanged.
* `equalAspect` — Requests plotting with an aspect ratio of 1. If the '1' qualifier is given, then the aspect ratio is achieved by changing the size of the plot region within the window; this is the default. If the '-1' qualifier is given, then the aspect ratio is achieved by changing the size of the plot region in user's coordinates.
* `noBorder` — Specifies that no border will be placed around the graph.

– Miscellaneous:

`[-output=`*filename*`] [-verbosity[=level]]`

* `output` — Requests SDDS output of a new file containing the data with any modifications resulting in the processing requested.
* `verbosity` — Sets the verbosity level of informational printouts. Higher integer values of the `level` parameter result in more output.

- see also:
  - `sddscongen` (4.9)
  - `sddshist2d` (??)
  - `rpn` (4.41)
- **author:** M. Borland, ANL/APS.

4.11 sddsconvert

- **description:** `sddsconvert` converts SDDS files between ASCII and binary, and allows wildcard-based filtering-out of unwanted columns and/or rows, as well as renaming of columns.
- **example:** Convert `APS.twi` to binary:

  ```
  sddsconvert -binary APS.twi
  ```

  Convert `APS.twi` to binary and delete the alphax and alphay columns:

  ```
  sddsconvert -binary APS.twi -delete=column,'alpha?'
  ```

- **synopsis:**

  ```
  sddsconvert [inputFile] [outputFile] [-pipe[=input][,output]]
   [{-binary | -ascii}] [-fromPage=number] [-toPage=number]
   [-delete={columns | parameters | arrays},matchingString[,matchingString...]]
   [-retain={columns | parameters | arrays},matchingString[,matchingString...]
   [-rename={columns | parameters | arrays},oldname=newname
        [,oldname=newname...]]
   [-editNames={columns | parameters | arrays},matchingString,editString
   [-description=text,contents]
   [-recover] [-linesPerRow=number] [-nowarnings]
  ```

- **files:** *inputFile* is an SDDS file containing data to be processed. The *outputFile* argument is optional. If it is not given, and if an output pipe is not selected, then the input file will be replaced.
- **switches:**
  - `{-binary | -ascii}` — Requests that the output be binary or ASCII.
  - `fromPage=`*number* — Specifies the first data page of the file that will appear in the output. By default, the output starts with data page 1.
  - `toPage=`*number* — Specifies the last page of the file that will appear in the output. By default, the output ends with the last data page in the file.
  - `-delete=columns | parameters | arrays,`*matchingString*`[,`*matchingString*`...]`, `-retain=columns | parameters | arrays,`*matchingString*`[,`*matchingString*`...]` — These options specify wildcard strings to be used to select entities (i.e., columns, parameters, or arrays) that will respectively be deleted or retained (i.e., that will not or will appear in the output). The selection is performed by determining which input entities have names matching any of the strings. If `retain` is given but `delete` is not, only those entities matching one of the strings given with `retain` are retained. If both `delete` and `retain` are given, then all entities are retained except those that match a `delete` string without matching any of the `retain` strings.
  - `-rename={columns | parameters | arrays},`*oldname*`=`*newname*`[,`*oldname*`=`*newname*`...]` — Specifies new names for entities in the output data set. The entities must still be referred to by their old names in the other commandline options.

- `-editNames=columns | parameters | arrays,`*matchingString*`,`*editString* — Specifies creation of new names for entities of the specified type with names matching the specified wildcard string. Editing is performed using commands reminiscent of emacs keystrokes. For details on editing commands, see `SDDS editing` (4.40).
- `-description=`*text*`,`*contents* — Sets the description fields for the output.
- `-recover` — Asks for attempted recovery of corrupted binary data.
- `-linesPerRow=`number — Sets the number of lines of text output per row of the tabular data, for ASCII output only.
- `-nowarnings` — Suppresses warning messages, such as file replacement warnings.

- see also:
  - Data for Examples (see 3.3)
  - `sddsprocess` (4.27)
  - `SDDS editing` (4.40)
- **author:** M. Borland, ANL/APS.

4.12 sddscorrelate

- **description:** `sddscorrelate` computes correlation coefficients and correlation significance between column data. The correlation coefficient between columns i and j is defined as

$$C_{ij} = \frac{\langle x_i x_j \rangle}{\sqrt{\langle x_i^2 \rangle \langle x_j^2 \rangle}}$$

If $C_{ij} = 1$, then the variables are perfectly correlated, whereas if $C_{ij} = -1$, they are perfectly anticorrelated. The correlation significance is the probability that the observed correlation coefficient could happen by chance if the variables were in fact uncorrelated. Hence, a very small correlation significance means that the variables are probably correlated.

- **examples:** Find the correlations among beam-position-monitor x values in `par.bpm`:

```
sddscorrelate par.bpm par.cor -column='*x'
```

Find the correlations of these readouts with one specific readout only:

```
sddscorrelate par.bpm par.cor -column='*x' -withOnly=P1P1x
```

- **synopsis:**

`sddscorrelate [-pipe=[input][,output]]` [*inputFile*] [*outputFile*]
`[-columns=`*columnNames*`]` `[-excludeColumns=`*columnNames*`]` `[-withOnly=`*columnName*`]`

- **files:** *inputFile* is an SDDS file containing two or more columns of data. For each page of the file, *outputFile* contains the correlation coefficients and significance for every possible pairing of variables requested. *outputFile* also contains three string columns: `Correlate1Name`, `Correlate2Name`, and `CorrelatePair`. These are respectively the name first column in the analysis, the name of the second column in the analysis, and a string of the form *Name1.Name2*.

- **switches:**
  - `-pipe=[input][,output]` — The standard SDDS Toolkit pipe option.
  - `-columns=`*columnNames* — Specifies the names of columns to be included in the analysis. A comma-separated list of optionally wildcard-containing names may be given.
  - `-excludeColumns=`*columnNames* — Specifies the names of columns to be excluded from the analysis. A comma-separated list of optionally wildcard-containing names may be given.
  - `-withOnly=`*columnName* — Specifies that one of the variables for each correlation will be the named column.

- **author:** M. Borland, ANL/APS.

4.13 sddsderiv

- **description:** sddsderiv differentiates one or more columns of data as a function of a common column. The program will perform error propagation if error bars are provided in the data set.

- **examples:** Find the derivatives of columns J0 and J1 as a function of z:

```
sddsderiv bessel.sdds bessel.deriv -differentiate=J0,J1 -versus=z
```

- **synopsis:**

```
sddsderiv [-pipe=[input][,output]] [input] [output]
-differentiate=columnName[,sigmaName] ...  -versus=columnName[,sigmaName]
[-interval=integer] [-mainTemplates=item=string[,...]]
[-errorTemplates=item=string[,...]]
```

- **files:** *input* is an SDDS file containing columns of data to be differentiated. If it contains multiple data pages, each is treated separately. The independent quantity along with the requested derivatives are placed in columns in *output*. By default, the derivative column name is constructed by appending Deriv to the variable column name. If applicable, the column name for the derivative error is constructed by appending DerivSigma. The data with respect to which the derivative is taken should be monotonically ordered.

- **switches:**

  - -pipe[=input][,output] — The standard SDDS Toolkit pipe option.
  - -differentiate=*columnName*[,*sigmaName* — Specifies the name of a column to differentiate, and optionally the name of the column containing the error in the differentiated quantity. May be given any number of times.
  - -versus=*columnName*[,*sigmaName* — Specifies the name of the independent variable column, and optionally the name of the column containing its error.
  - -interval=*integer* — Specifies the spacing of the data points used to approximate the derivative. The default value of 2 specifies that the derivative for each point will be obtained from values 1 row above and 1 row below the point. In general (ignoring end points, which require special treatment):

$$\frac{dy}{dx}[i] \approx \frac{y[i + Interval/2] - y[i - Interval/2]}{x[i + Interval/2] - x[i - Interval/2]}$$

  - -mainTemplates=*item=string*[,...] — Specifies template strings for names and definition entries for the derivative columns in the output file. *item* may be one of name, description, symbol. The symbols "%x" and "%y" are used to represent the independent variable name and the name of the differentiated quantity, respectively.
  - -errorTemplates=*item=string*[,...] — Specifies template strings for names and definition entries for the derivative error columns in the output file. *item* may be one of name, description, the independent variable name and the name of the differentiated quantity, respectively.

- **see also:**

- sddsinteg (4.20)

- **author:** M. Borland, ANL/APS.

4.14 sddsdigfilter

- **description:**

  sddsdigfilter performs time-domain digital filtering of columns of data. Filters can be combined in series and/or cascade to produce complex filter characteristics. In addition to allowing simple 1-pole lowpass and highpass filters, filter charateristics can be defined using either digital 'Z' or analog 'S' domain transfer functions.

  A digital filter has a Z transform given by

$$\frac{b_0 + b_1 z^{-1} + \ldots + b_n z^{-n}}{a_0 + a_1 z^{-1} + \ldots + a_n z^{-n}},$$

  while an analog filter has a Laplace transform given by

$$\frac{d_0 + d_1 s^1 + \ldots + d_n s^n}{c_0 + c_1 s^1 + \ldots + c_n s^n},$$

- **examples:** These examples assume the existence of a file data.wf containing a waveform stored as a column value that is a function of a column time that has units of seconds.

  Pass data through lowpass filter with a -3dB cutoff of 0.01 Hz:

```
sddsdigfilter data.wf -col=time,value result.wf -low=1,0.01.
```

  Bandstop filter between 10 Hz and 100 Hz:

```
sddsdigfilter data.wf -col=time,value result.wf -low=1,10 -high=1,100
```

  Bandpass filter between 10 Hz and 100 Hz:

```
sddsdigfilter data.wf -col=time,value result.wf -low=1,100 -cascade
-high=1,10
```

  Analog transfer function:

```
sddsdigfilter data.wf -col=time,value result.wf
-analog=D,1.0,0.01,C,0.1,0.3,1.6
```

  Five-sample digital delay:

```
sddsdigfilter data.wf -col=time,value result.wf -digital=B,0,0,0,0,0,1
```

- **synopsis:**

  sddsdigfilter [*inputFile*] [*outputFile*] [-pipe=[input][,output]]
  -columns=*xName*,*yName* [-proportional=*gain*] [-lowpass=*gain*,*cutoffFrequency*]
  [-highpass=*gain*,*cutoffFrequency*] [-digitalfilter=*sddsfile*,*aCoeffName*,*bCoeffName*
  [-digitalfilter=[A,*a0*,*a1*,..,*am*][,B,*b0*,*b1*,..,*bn*]
  [-analogfilter=*sddsfile*,*cCoeffName*,*dCoeffName*
  [-analogfilter=[C,*c0*,*c1*,..,*cm*][,D,*d0*,*d1*,..,*dn*] [-cascade] [-verbose]

- **files:** Two file names are required: the name of the existing input file, and the name of the output file to be produced. The input file must contain at least two columns: one containing to data to be filtered (*yName*) and the other giving time information (*xName*). A linear time scale is assumed for *xName*. The output file is a copy of the input file with an additional column called `DigFiltered`*yName* where *yName* would be the name of the original y-column.
- **switches:**
  - `-pipe[=input][,output]` — The standard SDDS Toolkit pipe switch.
  - `-columns=`*xName*`,`*yName* — The names of the input file data columns.
  - `-proportional=`*gain* — Defines a gain stage, where *gain* is the multiplier applied to the data.
  - `-lowpass=`*gain*`,`*cutoffFrequency* — Defines a lowpass filter stage, where *gain* is the mutiplier applied to the data and *cutoffFrequency* is the -3dB point of the filter in units appropriate to the supplied *xName*.
  - `-highpass=`*gain*`,`*cutoffFrequency* — Defines a highpass filter stage, where *gain* is the multiplier applied to the data and *cutoffFrequency* is the -3dB point of the filter in units appropriate to the supplied *xName*.
  - `-digitalfilter=`*sddsfile*`,`*aCoeffName*`,`*bCoeffName* — Defines a digital filter with coefficients in the supplied SDDS coefficient file. This file must cointain two columns containing the A and B coefficients of a digital 'Z' transfer function. Note that control theory convention assumes that the A0 coefficient is always 1.0. To ensure consistency with the SDDS file, the a0 coefficient is the first row in the A-column and must be implicitly supplied. Although there is little benefit to setting a0 to anything other than 1.0, it is allowed.
  - `-digitalfilter=[A,`*a0*`,`*a1*`,...,`*am*`][,B,`*b0*`,`*b1*`,...,`*bn*`]` — Defines a digital filter with the A and B coefficients of the digital 'Z' transfer function supplied on the command line. Either A or B or both coefficients can be supplied. If no A coefficients are supplied, a0 is set to 1.0. Equally, if no B coefficients are supplied, b0 is set to 1.0. If different numbers of A and B coefficients are suppied, the filter order is determined from the largest order.
  - `-analogfilter=`*sddsfile*`,`*cCoeffName*`,`*dCoeffName* — Defines an analog filter with coefficients in the supplied sdds cefficient file. This file must cointain two columns containing the C and D coefficients of an analog 's' transfer function. Conversion to the digital domain is done using a bilinear transform. Note that the user must ensure adequate data sampled, since the general format does not allow frequency warping based on the filter cutoff frequency.
  - `-analogfilter=[A,`*a0*`,`*a1*`,...,`*am*`][,B,`*b0*`,`*b1*`,...,`*bn*`]` — Defines an analog filter with the C and D coefficients of the analog 'S' transfer function supplied on the command line. Either C or D or both coefficients can be supplied. If no C coefficients are supplied, then c0 is set to 1.0. Equally, if no D coefficients are supplied, then d0 is set to 1.0. Conversion to the digital domain is done using a bilinear transform. Note that the user must ensure adequate data sampled, since the general format does not allow frequency warping based on the filter cutoff frequency.
  - `-cascade` — Defines the start of a new filter stage. Any number of filter stages can be supplied for a single data set. If more than one filter is defined, then the outputs are

summed unless the `-cascade` switch is supplied between the filter definitions in which case the output of the first filter stage is fed into the input of the subsequent filter stage.

  - `-verbose` — Prints the filter coefficients for each filter stage.

- **references** — The digital filtering routines were adapted from Stearns and David, *Signal Processing Algorithms in Fortran and C*, Prentice Hall, 1993

- **author:** John Carwardine, Argonne National Laboratory

4.15 sddsenvelope

- **description:** sddsenvelope analyzes column data across pages to find minima, maxima, averages, standard-deviations, etc., on a row-by-row basis. It produces a single-page output file containing one column for each analysis requested. It will also copy through data from the first page into the output file. It requires that each page of the input file have the same number of rows.

- **examples:** Find the minimum and maximum beta functions for a set of APS lattices:

```
sddsenvelope APS.twi APS.twi.env -copy=s -minimum=beta?  -maximum=beta?
```

- **synopsis:**

sddsenvelope [-pipe=[input][,output]] [*input*] [*output*] [-copy=*columnNames*]
[-maximum=*columnNames*] [-minimum=*columnNames*] [-mean=*columnNames*]
[-sum=*power*,*columnNames*] [-standardDeviation=*columnNames*]
[-rms=*columnNames*] [-slope=*independentVariableName*,*columnNames*]
[-intercept=*independentVariableName*,*columnNames*]

- **files:** *inputFile* is a multipage file containing the data for which row-by-row statistics are desired. *outputFile* is a single-page file containing the statistics. The column names in *outputFile* are created from those in the input file by appending the appropriate suffix from the following list: Max, Min, Mean, StDev, RMS, Sum, Slope, or Intercept.

- **switches:**

  - -pipe=[input][,output] — The standard SDDS Toolkit pipe option.
  - -copy=*columnNames* — Specifies that the named columns should be transferred to the output file without alteration. These data come from the first page of the input file. A comma-separated list of optionally wildcard-containing strings may be given.
  - -maximum=*columnNames*,
    -minimum=*columnNames*, -mean=*columnNames*, -rms=*columnNames* — Specifies that the named columns should be analysed in the indicated fashion. A comma-separated list of optionally wildcard-containing strings may be given.
  - -sum=*power*,*columnNames* —Specifies that the named columns should be analysed in the indicated fashion, i.e., that each output row should be the sum of the values to the indicated power. A comma-separated list of optionally wildcard-containing strings may be given.
  - -slope=*independentVariableName*,*columnNames*,
    -intercept=*independentVariableName*,*columnNames*— Specifies that the named columns should be analysed to get the slope or intercept with respect to the parameter *independentVariableName*. A comma-separated list of optionally wildcard-containing strings may be given for the *columnNames*.

- **see also:**

  - Data for Examples (see 3.3)
  - sddschanges (4.5)

- **author:** M. Borland, ANL/APS.

4.16 sddsexpfit

- **description:** `sddsexpfit` does exponential fits to a single column of an SDDS file as a function of another column (the independent variable). The fitting function is

$$E(x) = C + F * e^{R*x},$$

where x is the independent variable, C is the *constant* term, F is the *factor*, and R is the rate.

- **examples:** Fit an exponential decay to vacuum pressure versus time during a pumpdown:

```
sddsexpfit vacDecay.sdds -columns=Time,Pressure vacDecay.fit
```

Same, but give the program a hint and force it to get a better fit

```
sddsexpfit vacDecay.sdds -columns=Time,Pressure vacDecay.fit -clue=decays
-tolerance=1e-12
```

- **synopsis:**

```
sddsexpfit [-pipe=[input][,output]] [inputFile] [outputFile]
[-columns=xName,yName] [-tolerance=value] [-clue={grows | decays}]
[-guess=constant,factor,rate] [-verbosity=integer] [-fullOutput]
```

- **files:** *inputFile* contains the columns of data to be fit. If *inputFile* contains multiple pages, each page of data is fit separately. *outputFile* has columns containing the independent variable data and the corresponding values of the fit. The name of the latter column is constructed by appending the string `Fit` to the name of the dependent variable. In addition, if `-fullOutput` is given, *outputFile* includes a column with the dependent values and the residual (dependent values minus fit values). The name of the residual column is constructed by appending the string `Residual` to the name of the dependent variable. *outputFile* contains four parameters: `expfitConstant`, `expfitFactor`, `expfitRate`, and `expfitRmsResidual`. The first three parameters are respectively C, F, and R from the above equation. The last is the rms residual of the fit.

- **switches:**
  - `-pipe=[input][,output]` — The standard SDDS Toolkit pipe option.
  - `-columns=`*xName,yName* — Specifies the names of the independent and dependent columns of data.
  - `-tolerance=`*value* — Specifies how close `sddsexpfit` will attempt to come to the optimum fit, in terms of the mean squared residual. The default is 10^{-8}.
  - `-clue={grows | decays}` — Helps `sddsexpfit` decide whether the data is a decaying or growing exponential, i.e., whether R is negative or positive, respectively. If `sddsexpfit` is having trouble, this will often help.
  - `-guess=`*constant,factor,rate* — Gives `sddsexpfit` a stating point for each of the three fit parameters.
  - `-fullOutput` — Specifies that *outputFile* will contain the original dependent variable data and the fit residuals, in addition to the independent variable data and the fit values.

- `-verbosity=`*integer* — Specifies that informational printouts are desired during fitting. A larger integer produces more output.

- see also:
  - Data for Examples (see 3.3)
  - `sddspfit` (4.24)
  - `sddsgfit` (4.18)
  - `sddsoutlier` (4.22)

- author: M. Borland, ANL/APS.

4.17 sddsfft

- **description:** `sddsfft` takes Fast Fourier Transforms of real data in columns. It will transform any number of columns simultaneously as a function of a single independent variable. Strictly speaking, the independent variable values should be equispaced; if they are not, `sddsfft` uses the average spacing. The number of data points need not be a power of two. Output of the magnitude only is the default, but phase and complex values are available.

- **examples:** Take the FFT of time series samples of PAR x beam-position-monitor readouts:

```
sddsfft par.bpm par.fft -column=Time,'P?P?x'
```

- **synopsis:**

```
sddsfft [-pipe=[input][,output]] [inputFile] [outputFile]
-columns=indepVariable[,depenQuantityList] [-padWithZeroes | -truncate]
[-sparse=integer] [-window[={hanning | welch | parzen}]] [-normalize]
[-suppressAverage] [-fullOutput]
```

- **files:** *inputFile* contains the data to be FFT'd. One column from this file must be chosen as the independent variable. By default, all other columns are taken as dependent variables. If *inputFile* contains multiple pages, each is treated separately and is delivered to a separate page of *outputFile*.

  outputFile contains a column `f` for the frequency, along with one or more columns for each independent variable. By default, *outputFile* has one column named `FFT`*indepName* containing the magnitude of the FFT for each independent variable. If `-fullOutput` is specified, *outputFile* contains additional columns for, respectively, the phase (or argument), real part, and imaginary part of the FFT: `Arg`*indepName*, `Real`*indepName*, and `Imag`*indepName*.

- **switches:**

  - `pipe[=input][,output]` — The standard SDDS Toolkit pipe option.
  - `-columns=`*indepVariable*`[,`*depenQuantityList*`]` — Specifies the name of the independent variable column. Optionally, specifies a list of comma-separated, optionally wildcard-containing names of dependent quantities to be FFT'd as a function of the independent variable. By default, all numerical columns except the independent column are FFT'd.
  - `-padWithZeros` — Specifies that the independent data should be padded with zeros to make the number of points equal to the nearest power of two. In some cases, this will result in significantly greater speed.
  - `-truncate` — Specifies that the data should be truncated so that the number of points is the largest power of two not greater than the original number of points. In some cases, this will result in significantly greater speed.
  - `sparse=`*integer* — Specifies that the data should be uniformly sampled at the given integer interval. While this reduces frequency span of the FFT, it may result in greater speed.
  - `window[={hanning | welch | parzen}` — Specifies that data windowing should be performed prior to taking FFT's, and optionally specifies the type of window. The default is hanning. Usually used to improve visibility of small features or accuracy of amplitudes for data that is not periodic in the total sampling time or a submultiple thereof.

- `normalize` — Specifies that FFT's will be normalized to give a maximum magnitude of 1.
- `suppressAverage` — Specifies that the average value of the data will be subtracted from every point prior to taking the FFT. This may improve accuracy and visibility of small components.
- `fullOutput` — Specifies that in addition to the magnitude, the phase, real part, and imaginary part of each FFT will be included in the output.

- **see also:**
  - Data for Examples (see 3.3)
  - `sddsdigfilter` (4.14)
- **author:** M. Borland, ANL/APS.

4.18 sddsgfit

- **description:** `sddsgfit` does gaussian fits to a single column of an SDDS file as a function of another column (the independent variable). The fitting function is

$$G(x) = B + H * e^{\frac{-(x-\mu)^2}{2\sigma^2}},$$

where x is the independent variable, B is the baseline, H is the height, μ is the mean, and σ is the width.

- **examples:** Fit a gaussian to a beam profile to get the rms beam size:

```
sddsgfit beamProfile.sdds beamProfile.gfit -column=x,Intensity
```

- **synopsis:**

```
sddsgfit [-pipe=[input][,output]] [inputFile] [outputFile]
-columns=x-name,y-name[,sy-name] [-fitRange=lower,upper] [-fullOutput]
[-guesses=[baseline=value][,mean=value][,height=value][,sigma=value]]
[-stepSize=factor] [-tolerance=value]
[-limits=[evaluations=number][,passes=number] [-verbosity=integer]
```

- **files:** *inputFile* contains the columns of data to be fit. If *inputFile* contains multiple pages, each page of data is fit separately. *outputFile* has columns containing the independent variable data and the corresponding values of the fit. The name of the latter column is constructed by appending the string `Fit` to the name of the dependent variable. In addition, if `-fullOutput` is given, it includes a column with the dependent values and the residual (dependent values minus fit values). The name of the residual column is constructed by appending the string `Residual` to the name of the dependent variable. *outputFile* contains five parameters: `gfitBaseline`, `gfitHeight`, `gfitMean`, `gfitSigma`, and `gfitRmsResidual`. The first four parameters are respectively B, H, μ, and σ from the equation above. The last is the rms residual of the fit.

- **switches:**
  - `-pipe=[input][,output]` — The standard SDDS Toolkit pipe option.
  - `-columns=`*x-name,y-name* — Specifies the names of the independent and dependent columns of data.
  - `-fitRange=`*lower,upper* — Specifies the range of independent variable values to use in the fit.
  - `-guesses=[baseline=`*value*`][,mean=`*value*`][,height=`*value*`][,sigma=`*value*] — Gives `sddsgfit` a starting point for each of fit parameters.
  - `-stepSize=`*factor* — Specifies the starting stepsize for optimization as a fraction of the starting values. The default is 0.01.
  - `-tolerance=`*value* — Specifies how close `sddsgfit` will attempt to come to the optimum fit, in terms of the mean squared residual. The default is 10^{-8}.

- `-limits=[evaluations=`*number*`][,passes=`*number* — Specifies limits on how many fit function evaluations and how many minimization passes will be done in the fitting. The defaults are 5000 and 100, respectively. If the fit is not converging, try increasing one or both of these. If the number of evaluations is too small, you may get warning messages about optimization failures.
- `-fullOutput` — Specifies that *outputFile* will contain the original dependent variable data and the fit residuals, in addition to the independent variable data and the fit values.
- `-verbosity=`*integer* — Specifies that informational printouts are desired during fitting. A larger integer produces more output.

- **see also:**
  - `sddspfit` (4.24)
  - `sddsexpfit` (4.16)
  - `sddsoutlier` (4.22)
- **author:** M. Borland, ANL/APS.

4.19 sddshist

- **description:** `sddshist` does weighted and unweighted one-dimensional histograms of column data from an SDDS file. It also does limited statistical analysis of data, and basic filtering of data.

- **examples:** Make a 20-bin histogram of a series of PAR x beam-position-monitor readouts:

```
sddshist par.bpm par.bpmhis -data=P1P1x -bins=20
```

- **synopsis:**

```
sddshist [-pipe=[input][,output]] [inputFile] [outputFile]
-dataColumn=columnName [-bins=number | -sizeOfBins=value] [-lowerLimit=value]
[-upperLimit=value] [-filter=columnName,lowerLimit,upperLimit]
[-weightColumn=columnName] [-sides] [-normalize[={sum | area | peak}]]
[-statistics] [-verbose]
```

- **files:** *inputFile* is the name of an SDDS file containing data to be histogrammed, along with optional weight data. If *inputFile* contains multiple data pages, each is treated separately. The histogram or histograms are placed in *outputFile*, which has two columns. One column has the same name as the histogrammed variable, and consists of equispaced values giving the centers of the bins. The other column, named `frequency`, contains the histogram frequencies. Its precise meaning is dependent on normalization modes and weighting. By default, it contains the number of data points in the corresponding bin.

  If requested, *outputFile* will also contain parameters giving statistics for the data being histogrammed. See below for details.

- **switches:**

  - `-pipe[=input][,output]` — The standard SDDS Toolkit pipe option.
  - `-dataColumn=`*columnName* — Specifies the name of the data column to be histogrammed.
  - `-bins=`*number* — Specifies the number of bins to use. The default is 20.
  - `-sizeOfBins=`*value* — Specifies the size of bins to use. The number of bins is computed from the range of the data.
  - `-lowerLimit=`*value* — Specifies the lower limit of the histogram. By default, the lower limit is the minimum value in the data.
  - `-upperLimit=`*value* — Specifies the upper limit of the histogram. By default, the upper limit is the maximum value in the data.
  - `-filter=`*columnName*`,`*lowerLimit*`,`*upperLimit* — Specifies the name of a column by which to filter the input rows. Rows for which the named data is outside the specified interval are discarded. Alternatively, one can use sddsprocess (see 4.27) to winnow data and pipe it into `sddshist`.
  - `-weightColumn=`*columnName* — Specifies the name of a column by which to weight the histogram. This means that data points with a higher corresponding weight value are counted proportionally more times in the histogram.

- `-sides` — Specifies that zero-height bins should be attached to the lower and upper ends of the histogram. Many prefer the way this looks on a graph.
- `-normalize[={sum | area | peak}]` — Specifies that the histogram should be normalized, and how. The default is `sum`. `sum` normalization means that the sum of the heights will be 1. `area` normalization means that the area under the histogram will be 1. `peak` normalization means that the maximum height will be 1.
- `-statistics` — Specifies that statistics should be computed for the data and placed in *outputFile*. These presently include arithmetic mean, rms, and standard deviation. The parameters are named by appending the strings `Mean`, `RMS`, and `StDev` to the name of the data column. If `-weigthColumn` is given, the statistics are weighted.

- **see also:**
  - Data for Examples (see 3.3)
  - `sddshist2d` (??)
  - `sddsprocess` (4.27)
- **author:** M. Borland, ANL/APS.

4.20 sddsinteg

- **description:** sddsinteg integrates one or more columns of data as a function of a common column. The program will perform error propagation if error bars are provided in the data set.

- **examples:** Find the integral $\int \eta_x ds$ for APS lattices

```
sddsinteg APS.twi APS.integ -integrate=etax -versus=s
```

- **synopsis:**

```
sddsinteg [-pipe=[input][,output]] [input] [output]
-integrate=columnName[,sigmaName] ... -versus=columnName[,sigmaName]
[-mainTemplates=item=string[,...]] [-errorTemplates=item=string[,...]]
[-method=methodName] [-printFinal[=bare][,stdout]]
```

- **files:** *input* is an SDDS file containing columns of data to be integrated. If it contains multiple data pages, each is treated separately. The independent quantity along with the requested integrals is placed in columns in *output*. By default, the integral column name is constructed by appending "Integ" to the variable column name. If applicable, the column name for the integral error is constructed by appending "IntegSigma".

- **switches:**

  - `-pipe[=input][,output]` — The standard SDDS Toolkit pipe option.
  - `-integrate=`*columnName*`[,`*sigmaName*`]` — Specifies the name of a column to integrate, and optionally the name of the column containing the error in the integrand. May be given any number of times.
  - `-versus=`*columnName*`[,`*sigmaName*`]` — Specifies the name of the independent variable column, and optinally the name of the column containing its error.
  - `-mainTemplates=`*item*`=`*string*`[,...]` — Specifies template strings for names and definition entries for the integral columns in the output file. *item* may be one of `name`, `description`, `symbol`. The symbols "%x" and "%y" are used to represent the independent variable name and the name of the integrand, respectively.
  - `-errorTemplates=`*item*`=`*string*`[,...]` — Specifies template strings for names and definition entries for the integral error columns in the output file. *item* may be one of `name`, `description`, the independent variable name and the name of the integrand, respectively.
  - `-method=`*methodName* — Specifies the integration method. At present, only trapizoidal rule integration is support, so this option is ignored.
  - `-printFinal[=bare][,stdout]` — Specifies that the final value of each integral should be printed out. By default, the printout goes to stderr and includes the name of the integral. If `bare` is given, the names are omitted. If `stdout` is given, the printout goes to stdout.

- **see also:**

  - `sddsderiv` (4.13)

- **author:** M. Borland, ANL/APS.

4.21 sddsinterp

- **description:** sddsinterp does polynomial interpolation of one or more columns of data as a function of a common independent variable. Interpolation may be done at specified points, at a sequence of points, or at points given in another SDDS file.

- **examples:** Do second-order polynomial interpolation of Twiss parameters at 250 points to get smoother-looking data:

```
sddsinterp APS.twi APS.interp -column=s,betax,betay -order=2 -sequence=250
```

- **synopsis:**

```
sddsinterp [-pipe=[input][,output]] [inputFile] [outputFile]
[-columns=independentQuantity,name[,name...]]
 { -atValues=valuesList |
 -sequence=points[,start,end] |
 -fileValues=valuesFile[,column=columnName][,parallelPages] }
 [-order=number] [-printOut[=bare][,stdout]]
 [-belowRange={value=value | skip | saturate | extrapolate | wrap}[,{abort |
warn}]
 [-aboveRange={value=value | skip | saturate | extrapolate | wrap}[,{abort |
warn}]
```

- **files:** *inputFile* is an SDDS file containing columns of data to be interpolated. One column is selected as the independent variable. Any number of others may be specified as dependent variables. If *inputFile* contains multiple data pages, each is treated separately. *outputFile* contains the independent variable values at which interpolation was performed, in a column with the same name as the independent variable in *inputFile*. Similarly, the interpolated values are placed in *outputFile* under the same names as the independent columns from *inputFile*.

- **switches:**
  - `-pipe[=input][,output]` — The standard SDDS Toolkit pipe option.
  - `-columns=`*independentQuantity*`,`*name*`[,`*name*`...]` — Specifies the names of the independent and dependent variable columns.
  - `-atValues=`*valuesList* — Specifies a comma-separated list of values at which interpolation is done.
  - `-sequence=`*points*`[,`*start*`,`*end*`]` — Specifies a sequence of equispaced points at which interpolation is done. If *start* and *end* are given, they specify the range of these points. If they are not given, the range is the range of the independent data.
  - `-fileValues=`*valuesFile*`[,column=`*columnName*`][,parallelPages]` — Specifies a set of values at which interpolation is to be done. In this case, the values are extracted from a column (*columnName*) of an SDDS file (*valuesFile*). If `parallelPages` is given, then successive pages of *inputFile* are interpolated at points given by successive pages of *valuesFile*. Otherwise, each page of *inputFile* is interpolated at the values in all pages of *valuesFile*; this can take quite some time if both files have many pages with many rows.

- `-order=`*number* — The order of the polynomials to use for interpolation. The default is 1, indicating linear interpolation.
- `-printOut[=bare][,stdout]` — Specifies that interpolated values should be printed to stderr. By default, the printout contains text identifying the quantities; this may be suppressed by specifying `bare`. Output may be directed to the standard output by specifying `stdout`.
- `-belowRange={value=`*value* `| skip | saturate | extrapolate | wrap}[,{ abort | warn}, -aboveRange={value=`*value* `| skip | saturate | extrapolate | wrap}[,{ abort | warn}` — These options specify the behavior in the event that an interpolation point is, respectively, below or above the range of the independent data. If such an out-of-range point occurs, the default behavior is to assign the value at the nearest endpoint of the data; this is identical to specifying `saturate`. One may specify use of a specific value with `value=`*value*. `skip` specifies that offending points should be discarded. `extrapolate` specifies extrapolation beyond the limits of the data. `wrap` specifies that the data should be treated as periodic. `abort` specifies that the program should terminate. `warn` requests warnings for out-of-bounds points.

- **see also:**
  - sddspfit (see 4.24)
  - Data for Examples (see 3.3)
- **author:** M. Borland, ANL/APS.

4.22 sddsoutlier

- **description:** `sddsoutlier` does outlier elimination of rows from SDDS tabular data. An "outlier" is a data point that is statistically unlikely or else invalid.

- **example:** Eliminate "bad" beam-position-monitor readouts from PAR x BPM data, where a bad readout is one that is more than :

```
sddsoutlier par.bpm par.bpm1 -columns=P?P?x -stDevLimit=3
```

Fit a line to readout P1P1x vs P1P2x, then eliminate points too far from the line.

```
sddspfit par.bpm -pipe=out -columns=P1P2x,P1P1x
| sddsoutlier -pipe=in par.2bpms -column=P1P1xResidual -stDevLimit=2
```

Same, but refit and redo outlier elimination based on the improved fit:

```
sddspfit par.bpm -pipe=out -columns=P1P2x,P1P1x
| sddsoutlier -pipe par.2bpms -column=P1P1xResidual -stDevLimit=2
| sddspfit -pipe -columns=P1P2x,P1P1x
| sddsoutlier -pipe=in par.2bpms -column=P1P1xResidual -stDevLimit=2
```

- **synopsis:**

```
sddsoutlier [-pipe=[input][,output]] [inputFile] [outputFile]
[-columns=listOfNames] [-excludeColumns=listOfNames] [-stDevLimit=value]
[-absLimit=value] [-absDeviationLimit=value] [-verbose] [-noWarnings]
```

- **files:** *inputFile* contains column data that is to be winnowed using outlier elimination. If *inputFile* contains multiple pages, the are treated separately. *outputFile* contains all of the array and parameter data, but only those rows of the tabular data that pass the outlier elimination. *Warning*: if *outputFile* is not given and `-pipe=output` is not specified, then *inputFile* will be overwritten.

- **switches:**

  - `-pipe[=input][,output]` — The standard SDDS Toolkit pipe option.
  - `-columns=`*listOfNames* — Specifies a comma-separated list of optionally wildcard containing column names. Outlier analysis and elimination will be applied to the data in each of the specified columns independently. No row that is eliminated by outlier analysis of any of these columns will appear in the output. If this option is not given, all columns are included in the analysis.
  - `-excludeColumns=`*listOfNames* — Specifies a comma-separated list of optionally wildcard containing column names that are to be excluded from outlier analysis.
  - `-stDevLimit=`*value* — Specifies the number of standard deviations by which a data point from a column may deviate from the average for the column before being considered an outlier.
  - `-absLimit=`*value* — Specifies the maximum absolute value that a data point from a column may have before being considered an outlier.

- `-absDeviationLimit=`*value* — Specifies the maximum absolute value by which a data point from a column may deviate from the average for the column before being considered an outlier.
- `-verbose` — Specifies that informational printouts should be provided.
- `-noWarnings` — Specifies that warnings should be suppressed.

- **see also:**
  - Data for Examples (see 3.3)
  - `sddspfit` (4.24)
  - `sddsgfit` (4.18)
  - `sddsexpfit` (4.16)
  - `sddscorrelate` (4.12)
- **author:** M. Borland, ANL/APS.

4.23 sddspeakfind

- **description:**

  sddspeakfind finds the locations and values of peaks in a single column of an SDDS file. It incorporates various features to help reject spurious peaks. The column is considered a function of the row index for the purpose of finding peaks. Hence, the data should be sorted if necessary using sddssort prior to using this program. I.e., if the data contains columns x and y, and one wants x values of peaks in y, then one should ensure that the rows are sorted into increasing or decreasing x order.

  It may also be helpful to smooth the data using sddssmooth in order to eliminate spurious peaks due to noisy data.

- **examples:** Find peaks in a Fourier transform:

```
sddspeakfind data.fft data.peaks -column=FFTamplitude
```

  Sort and smooth the data first:

```
sddssort data.fft -column=f,increasing -pipe=out
| sddssmooth -pipe -columns=FFTamplitude
| sddspeakfind -pipe=in data.peaks -column=FFTamplitude
```

- **synopsis:**

```
sddspeakfind [-pipe=[input][,output]] [inputFile] [outputFile]
-column=columnName [-fivePoints] [-threshold=value]
[-exclusionZone=fractionalInterval] [-changeThreshold=fractionalChange]
```

- **files:** *inputFile* contains the data to be searched for peaks. *outputFile* contains all of the array and parameter data from *inputFile*, plus data from all rows that contain a peak in the named column. No new data elements are created. If *inputFile* contains multiple pages, each is treated separately and is delivered to a separate page of *outputFile*.

- **switches:**

  - -pipe[=input][,output] — The standard SDDS Toolkit pipe option.
  - -column=*columnName* — Specifies the name of the column to search for peaks.
  - -fivePoints — Specifies peak analysis using five adjacent data points, rather than the default three. For three-point mode, a peak is any point which is larger than both of its two nearest neighbors. For five-point mode, the candidate point's nearest neighbors must in turn be higher than their nearest neighbors on the side away from the candidate point.
  - -threshold=*value* — Specifies a minimum value that a peak value must exceed in order to be included in the output. By default, no threshold is applied.
  - -exclusionZone=*fractionalInterval* — Specifies elimination of smaller peaks within a given interval around a larger peak. *fractionalInterval* is the width of the interval in units of the length of the data table.

- `-changeThreshold=`*fractionalChange* — Specifies elimination of peaks for which the fractional change between the peak value and the nearest neighbor points is less than the given amount. If `-fivePoints` is given, the nearest neighbors in question are those 2 rows above and below the peak.

- **see also:**
  - `sddsfft` (4.17)
  - `sddssmooth` (4.33)
  - `sddspeakfind` (4.23)

- **author:** M. Borland, ANL/APS.

4.24 sddspfit

- **description:** `sddspfit` does ordinary and Chebyshev polynomial fits to column data, including error analysis. It will do fits to with specified number of terms, with specific terms only, and with specific symmetry only. It will also eliminate spurious terms.

- **synopsis:**

  `sddspfit [-pipe=[input][,output]] [`*inputFile*`] [`*outputFile*`]`
  `-columns=`*xName*`,`*yName*`[,xSigma=`*name*`][,ySigma=`*name*`] -terms=`*number*
  `[-symmetry={none | odd | even}] | -orders=`*number*`[,`*number*`...]`
  `[-reviseOrders[=threshold=`*value*`][,verbose]] [-chebyshev[=convert]]`
  `[-xOffset=`*value*`] [-xFactor=`*value*`] [-sigmas={absolute=`*value* `| fractional=`*value*`}]`
  `[-modifySigmas] [-generateSigmas[=keepLargest | keepSmallest]]`
  `[-sparse=`*interval*`] [-range=`*lower*`,`*upper*`] [-normalize[=`*termNumber*`]] [-verbose]`
  `[-fitLabelFormat=`*sprintfString*`]`

- **files:** *inputFile* is an SDDS file containing columns of data to be fit. If it contains multiple pages, they are processed separately. *outputFile* is an SDDS file containing one page for each page of *inputFile*. It contains columns of the independent and dependent variable data, plus columns for error bars ("sigmas") as appropriate. The values of the fit and of the residuals are in a columns named *yName*`Fit` and *yName*`Residual`. *outputFile* also contains the following one-dimensional arrays:

  - `Order`: a long integer array of the polynomial orders used in the fit.
  - `Coefficient`: a double-precision array of fit coefficients.
  - `CoefficientSigma`: a double-precision array of fit coefficient errors. Present only if errors are present for data.
  - `CoefficientUnits`: a string array of fit coefficient units.

  outputFile also contains the following parameters:

  - `Basis`: a string identifying the type of polynomials use.
  - `ReducedChiSquared`: the reduced chi-squared of the fit:

    $$\chi^2_\nu = \frac{\chi^2}{\nu} = \frac{1}{N-T}\Sigma_{i=0}^{N-1}\left(\frac{y_i - y(x_i)}{\sigma_i}\right)^2,$$

    where $\nu = N - T$ is the number of degrees of freedom for a fit of N points with T terms.
  - `rmsResidual`
  - *xName*`Offset`, *xName*`Factor`
  - `FitIsValid`: a character having values y and n if the page contains a valid fit or not.
  - `Terms`: the number of terms in the fit.
  - `sddspfitLabel`: a string containing an equation showing the fit, suitable for use with `sddsplot`.
  - `Intercept`, `Slope`, `Curvature`: the three lowest order coefficients for ordinary polynomial fits. These are present only if orders 0, 1, and 2 respectively are requested in fitting. If error analysis is valid, then the errors for these quantities appear as *quantity-Name*`Sigma`.

- **switches:**
  - `-pipe[=input][,output]` — The standard SDDS Toolkit pipe option.
  - `-columns=`*xName*`,`*yName*`[,xSigma=`*name*`][,ySigma=`*name*`]` — Specifies the names of the columns to use for the independent and dependent data, respectively. `xSigma` and `ySigma` can be used to specify the errors for the independent and dependent data, respectively.
  - By default, an ordinary polynomial fit is done using a constant and linear term. Control of what fit terms are used is provided by the following switches:
    * `-terms=`*number* — Specifies the number of terms to be used in fitting. 2 terms is linear fit, 3 is quadratic, etc.
    * `-symmetry={none | odd | even}` — When used with `-terms`, allows specifying the symmetry of the N terms used. `none` is the default. `odd` implies using linear, cubic, etc., while `even` implies using constant, quadratic, etc.
    * `-orders=`*number*`[,`*number*`...]` — Specifies the polynomial orders to be used in fitting. The default is equivalent to *-orders=0,1*.
    * `-reviseOrders[=threshold=`*value*`][,verbose]` — Specifies adaptive fitting to eliminate spurious terms. When invoked, this switch causes `sddspfit` to repeatedly fit the first page of data with each term individually eliminated. If the resultant fit is not significantly worse than the fit containing the term, then the term is judged spurious and is eliminated. This stops only when elimination of any single term makes a significant worsening of the fit. By default, the criterion for a worse fit is one that has a larger reduced chi-squared. The `threshold` qualifier is used to specify how much larger the reduced chi-squared may be and still eliminate a term. The default value is 0.1.
    * `[-chebyshev[=convert]]` — Asks that Chebyshev T polynomials be used in fitting. If `convert` is given, the output contains the coeffients for the equivalent ordinary polynomials.
  - `-xOffset=`*value*, `-xFactor=`*value* — Specify offseting and scaling of the independent data prior to fitting. The transformation is $x \rightarrow (x - \text{Offset})/\text{Factor}$. This feature can be used to make a fit about a point other than x=0, or to scale the data to make high-order fits more accurate.
  - `sddspfit` will compute error bars ("sigmas") for fit coefficients if it has knowledge of the sigmas for the data points. These can be supplied using the `-columns` switch, or generated internally in several ways:
    * `-sigmas=absolute=`*value* `| fractional=`*value* — Specifies that independent-variable errors be generated using a specified value for all points, or a specified fraction for all points.
    * `-modifySigmas` — Specifies that independent-variable sigmas be modified to include the effect of uncertainty in the dependent variable values. If this option is not given, any x sigmas specified with `-columns` are ignored.
    * `-generateSigmas[={keepLargest | keepSmallest}]` — Specifies that independent-variable errors be generated from the variance of an initial equal-weights fit. If errors are already given (via *-column*), one may request that for every point `sddspfit` retain the larger or smaller of the sigma in the data and the one given by the variance.

- `-sparse=`*interval* — Specifies sparsing of the input data prior to fitting. This can greatly speed computations when the number of data points is large.
- `-range=`*lower*, *upper* — Specifies the range of independent variable over which to do fitting.
- `-normalize`[`=`*termNumber*] — Specifies that coefficients be normalized so that the coefficient for the indicated order is unity. By default, the 0-order term (i.e., the constant term) is normalized to unity.
- `-verbose` — Specifies that the results of the fit be printed to the standard error output.
- `-fitLabelFormat=`*sprintfString* — Specifies the format to use for printing numbers in the fit label. The default is "%g".

- **see also:**
  - Data for Examples (see 3.3)
  - `sddsexpfit` (4.16)
  - `sddsgfit` (4.18)
  - `sddsplot` (4.25)
  - `sddsoutlier` (4.22)
- **author:** M. Borland, ANL/APS.

4.25 sddsplot

- **description:** `sddsplot` is a general purpose device-independent graphics program for displaying parameter and column data from SDDS files. The program is equally capable of quick-and-dirty plots and publication quality graphics. It allows organization of large amounts of data from multiple files into useful plots with minimal effort. It provides line, point, symbol, impulse, error-bar, and arrow plotting, with automatic variation of color, linetype, etc. It can do data winnowing using the data to be graphed or other data in the file. Parameters from a file can be designated for use as plot labels, legends, or for placement on the plot in specified locations. Data pages may be tagged and sorted by multiple criteria.

  `sddsplot` supports various flavors of Postscript, various windows options, and numerous graphics terminals. For X-windows, a GUI interface is generated that supports zoom/pan, cursor readout, movie mode, and much more.

- **examples:** Plot the horizontal beta function for the APS design:

  ```
  sddsplot -columnNames=s,betax APS0.twi
  ```

  Plot the Twiss functions for the APS design:

  ```
  sddsplot -columnNames=s,'(beta?,etax)' APS0.twi
  ```

  Plot the Twiss functions for APS lattices, one plotting page per lattice (i.e., per data page), with different linetypes and a legend:

  ```
  sddsplot -columnNames=s,'(beta?,etax)' APS.twi -graphic=line,vary -legend
  -split=page -separate=page
  ```

  Plot the Twiss functions for APS lattices, one plotting page per function, with each data page shown with a different line type:

  ```
  sddsplot -columnNames=s,'(beta?,etax)' APS.twi -graphic=line,vary
  -split=page -groupby=nameIndex -separate=nameIndex
  ```

- **sddsplot concepts:**

  `sddsplot` has a very large number of options and is very flexible. In most cases, only a very few of these options are employed. In order to make best use of `sddsplot`, it helps to be familiar with certain concepts.

  `sddsplot` supports multiple "plot pages" and multiple "panels" per page. In this context, a "plot page" is a separate sheet of paper for hardcopy devices, and the equivalent for interactive devices. For example, when using the X-windows interface just described, separate plot pages are held in memory so that the user may go back and forth between them, or run them as a movie. A plot page may contain several nonoverlapping panels, each displaying essentially independent graphics. Presently, `sddsplot` divides the plot page into an array of plot panels, each of equal size. The default is one plot panel per plot page.

  Within each plot panel, `sddsplot` may display data from any number of "plot requests". A plot request is a specification to `sddsplot` of what data to plot from what files, and how to

do it. A plot request must contain or indicate a list of names of columns and parameters to display, as well as the names of one or more files from which to extract the data. The data from plot requests are organized into plot panels and plot pages according to certain defaults or explicit instructions. One frequent choice is to move to a new panel for each plot request. However, one may also regroup data to display data from different plot requests together.

For each request, the names of columns and parameters are grouped to form sets of sets of data element names. For example, `-columnNames=s,(betax,betay,etax)` results in formation of three sets of pairs: `(s, betax)`, `(s, betay)`, and `(s, etax)`. In a more complicated example, the sets of dataname sets might include names of error-bar data (e.g., `(x, y, ySigma)`) or vector components (e.g., `(x, y, Ex, Ey)`). To avoid confusion, a set of datanames like those just listed will be referred to as a "name group". Each name group for a request is given a sequential "name index", which can be used as shown in the last example above.

Each panel is divided into two regions, a "plot space" (or "pspace") and a "label space". The pspace is the region where data is displayed. Outside the pspace is the label space, where labels and legends normally appears.

Any point on any plot panel can be referenced by unit coordinates (p, q) that start at zero in the lower left corner of the panel and end at unity in the upper right corner. The extent of the pspace is given in these coordinates. The data or user's coordinates, referred to as (x, y), are mapped onto this space. By default, the region of the pspace that is covered by the user's coordinates is [0.15, 0.90]x[0.15, 0.90]. This leaves room around the plot for numeric and text labels. The pspace may be changed explicitly, or it may be altered implicitly by certain switches (e.g., to make room for legends).

While the user seldom needs to worry about the pspace, it is useful to understand that any point on the plot space may be addressed using the user's or data coordinates (i.e., (x, y)) or the pspace coordinates (i.e., (p, q)).

When `sddsplot` reads data in from files, it collects it into internal data sets. By default, each of these internal data sets contains the all of the data for one name group from one file. That is, an internal data set normally contains all of the data for a name group from an SDDS data set. The phrase "internal data set" is used to maintain the distinction between the SDDS data set and the representation of data from an SDDS data set within `sddsplot`. Associated with each internal data set is the request number, the filename, the file number within the request, the y dataname, the name group index within the request, the starting page number from the file, and an optional user-specified tag value (from the commandline or a parameter in the file). These values may be used to sort and group the data in order to place on individual panels sets of similar data from multiple sources. An instance of this is shown in the last example above.

- **synopsis:**

`sddsplot` [*X11Switches*] [*commonSwitches*] *plotRequestSwitch fileNames localSwitches* [*plotRequestSwitch fileNames localSwitches ...*]

The `sddsplot` command line is organized into three categories. First, one may issue any of the standard X11 switches (e.g., -geometry). Second, one may give a set of switches, indicated by *commonSwitches*, that will apply to all subsequest plot requests.

Third, one gives a series of "plot requests". A plot request starts with one of several switches that give the names of data elements to be plotted. It continues with the names of one or more

files from which this data is to be extracted. In addition, one may include various switches that apply only to current plot request. These may, for example, override any common switches that were set prior to the first plot request. In general, any switch may be given as a common switch (so that it applies to all plot requests unless overridden) or as a local switch.

In the examples above, only a single plot request is exhibited. There are no X11 switches and no common switches set. The plot request is initiated by the `-columnNames` switch. The -graphic and -legend switches are local switches.

- **switches:**
  - Initiating a plot request:
    * `-columnNames=`*xName,yNameList*`[,{`*y1NameList* | *x1Name,y1NameList*`}]` — Specifies the names of columns to be plotted. *xName* may be the name of a numeric or string column, which is normally plotted against the horizontal or x axis. *yNameList* gives the comma-separated, optionally wildcarded names of one or more columns of numeric data. Data for each item in *yNameList* will be paired with the x data for plotting.
      Some types of plotting require additional data, such as error bars or vector components. These are specified with the *x1Name* and *y1NameList*. Each item in *y1NameList* is paired with the corresponding item in *yNameList*; the lists must have the same length. The interpretation of the additional data is specified with the `-graphic=error` or `-arrow` switches. For error bar plotting, one may give error bars for both x and y by giving *x1Name* and *y1NameList*, or for y only by giving *y1NameList*. For arrow plotting, giving *y1NameList* only is allowable for vectors perpendicular to the page. Giving both *x1Name* and *y1NameList* is required for vectors in the plane of the page.
      One may give several `-columnNames` switches in a row in order to specify additional "datanames" for the request. This may be convenient if, for example, one wants several different x variables.
    * `-parameterNames=`*xName,yNameList*`[,{`*y1NameList* | *x1Name,y1NameList*`}]` — Identical to `-columnNames`, except it specifies parameter data to be plotted. As with `-columnNames`, several such options may be given in a row in order to add datanames.
    * `-keep[={names | files}]` — Specifies starting a new plot request, but retaining certain information from the previous request. If given without qualifiers, the datanames (as specified by `-columnNames` or `-parameterNames`) and filenames from the previous request are kept; this allows plotting the same data again in a different way. If the `names` qualifier is given, the datanames from the previous request are retained. If the `files` qualifier is given, the filenames from the previous request are retained.
    * `-mpl[=noTitle][,noTopline]` — Allows plotting of `mpl` data files with `sddsplot`. The x and y columns of the `mpl` file are used. The qualifiers may be employed to inhibit use of the `mpl` plot title and topline.
  - Controlling output type:
    * `-listDevices`—Lists the names of available graphics devices to the standard error output.

* -device=*deviceName*[,*deviceArguments*]—Specifies the name of the graphics device, plus optional device-specific arguments. The default device is "motif", unless the SDDS_DEVICE environment variable if defined, in which case the default device is the one named.
* -output=*filename*—Specifies the name of a file to which graphics output will be sent. Used primarily for hardcopy devices (e.g., Postscript) where the data will be sent to a printer. By default, the data for such devices is printed to the standard output.

– Controlling type of plotting:

* -graphic=*element*[,type=*integer*][,subtype={*integer* | type}][,connect[={*linetype* | type | subtype}][,vary[={type | subtype}]][,scale=*factor*][,{eachFile | eachPage | eachRequest}]—Specifies the type of graphic element to use for data in the present plot request.
element may be one of line, symbol, errorBar, impulse, bar, dot, or ontinue. These are largely self-explanatory. continue specifies continuing whatever was done in the previous request. impulse is a line extending from y=0 to the data value, while bar is a line extending from the bottom of the plot region to the data value.
The type field for the graphic element has different meanings for different elements. For lines, impulses, bars, and dots, the type is the color or line style used, depending on the device. For most devices, values between 0 and 15 inclusive given unique lines. For symbols and error bars, the type specifies the style of symbol or error bar to use; the value is between 0 and 8 inclusive for symbols and between 0 and 1 inclusive for error bars.
The subtype field is meaningful only for symbols and error bars. It specifies the line style or color to be used in making the symbol or error bar. As for the type field for line plotting, the value may be between 0 and 15 inclusive. The connect qualifier is also valid for symbols and error bars only. It specifies that the symbols and error bars should be connected by lines. By default, the line type used is 0.
* -arrowSettings=[,autoScale][scale=*factor*][,{cartesianData | polarData | scalarData}][,linetype=*integer*][,centered][,singleBarb][,barbLength=*value*][,barbA
Specifies parameters for plotting vectors using arrows.
autoScale specifies that the scale factor for the length of arrows should be chosen automatically; if several data pages are being plotted separately, the same scale is used for all of them. scale may be used instead of autoScale to set the factor manually; if both are given, then the factor given with scale multiplies that computed by autoScale.
linetype specifies the line type to use for the arrows, using the same mechanism as for lines in the -graphic switch. The default is 0.
cartesianData, polarData, and scalarData specify the type of data being provided. For the first two, one must have specified both *x1Name* and *y1NameList* in the plot request; for cartesianData, x1 and y1 are the x and y vector components, while for polarData x1 is the length and y1 is the angle in radians from the positive x direction.
centered specifies that arrows should be centered on the corresponding (x, y) point; by default, the arrow starts at the (x, y) point. singleBarb specifies that arrows

should have only a single barb, rather than the default two barbs; this can be significantly faster for large amounts of data. `barbLength` and `barbAngle` specify the length and angle of arrow barbs; the barb length is a specified as a fraction of the arrow length, which the barb angle is specified in degrees.

* `-linetypeDefault=`*integer*— Specifies the default line type for borders, legend text, labels, axes, and so on. If not given, 0 is used.

– Controlling the plotting region:

* `-scales=`*xmin*`,`*xmaxymin*`,`*ymax*—Specifies the region of the plot in user's coordinates. If *xmin* and *xmax* are equal, then autoscaling is used in x, and similarly for y. Note that data outside the specified region is still plotted, so that proper clipping of lines occurs.
* `-unsuppressZero[=x][,y]`—Specifies that x=0 and/or y=0 should be within the region of the plot. If given without qualifiers, both x and y are "unsuppressed".
* `-sameScale[=x][,y]`—Specifies that separate panels of data shall be displayed on the same scales. In other words, any autoscaling is done based on all of the data, rather than simply the data on a particular plot panel. If given without qualifiers, both x and y are affected.
* `-zoom=[xFactor=`*value*`][,yFactor=`*value*`][,{ xCenter=`*value* ` | qCenter=`*value*`}][,{ yCenter=`*value* ` | pCenter=`*value*`}]`—Specifies zoom and pan starting from the scales set by autoscaling or by `-scales`. A factor less than (greater than) unity zooms out (in). For each dimension, one may specify the center of the plot using either the
* `-aspectRatio=`*value*—Specifies the y/x aspect ratio of the plot. The value must be nonzero. If it is positive, then the desired aspect ratio is obtained by altering the pspace. If it is negative, the desired aspect ratio (the absolute value of the value given) is obtained by altering the data coordinate range.
* `-pSpace=`*hMin*`,`*hMax*`,`*vMin*`,`*vMax*—This option is seldom used, but allows control of the region of the panel that is mapped to data coordinates, said region being the "plot space" or "pspace". The first two coordinates give the horizontal extent, while the second two give the vertical extent. The coordinate values are between 0 and 1. The defaults are [0.15, 0.9]x[0.15, 0.9].

– Controlling axes, numeric labels, ticks, and grids:

* `-axes[=x][,y][,linetype=`*integer*`]`—Specifies that axes will be placed on the plot, if they are visible. By default, both x and y axes are created, with the same linetype as the labels, scales, and plot border. One may select a given axis by supply the x or y qualifier. One may specify the line type to use for the axes using the `linetype` qualifier.
* `-tickSettings=[,{`*xy*`}grid][,grid][{`*xy*`}spacing=`*value*`][,{`*xy*`}factor=`*value*`][,{`*xy*`}modul`
  Specifies how to make ticks and numeric labels for the x and y dimensions. All of the qualifiers have an *x* and *y* variant, e.g., `xgrid` and `ygrid`. Some have a variant that includes both x and y (e.g., `grid`). In the case of the grid option. `xgrid` specifies grid lines rather than ticks for the x dimension, `ygrid` is similar for the y dimension, and `grid` specifies grid lines in both dimensions.
  The `factor` qualifiers specify factors to apply to the data values in producing the labels. For example, one might want to muliply small values by a power of ten in

order to get labels that are of order units. The spacing values give the spacing of the ticks and labels with any factor included. I.e., to keep the same number of ticks, factor and spacing values must be increased together. Usually, giving the spacing qualifiers is unnecessary, since sddsplot chooses appropriate values.
The modulus qualifiers allow printing the modulus of the label value rather than the value itself; for example, one might use xmodulus=24 if x was the time in hours over many days. The size qualifiers permit specification of the size of the ticks as a fraction of the range in the opposing dimension; the default is 0.02. The *linetype* qualifiers specify the linetype to be used for ticks and grid lines, using integer values as for the -graph=line switch. The logarithmic qualifiers specify log-style ticks and labels; the implication is that the data being plotted is the base-ten logarithm of something.

* -subTickSettings=[{xy}divisions=*integer*][,[{xy}]grid][,[{xy}]linetype=*integer*][,[{
Specifies whether and how to make subticks or subgrid lines for the x and y dimensions. All of the qualifiers have two or more variants, one that applies to x, one that applies to y, and (in some cases) one that applies to both. For example, xgrid requests grid lines for x, ygrid requests grid lines for y, and grid requests grid lines for both x and y. The divisions qualifiers specify the number of subdivisions of the major tick intervals; the default is none. The linetype qualifiers specify the line type to use for subticks or subgrid lines. The fraction qualifiers specify the size of the subticks as a fraction of the plotting region; the default is 0.01.
* -grid[=x][,y]—This option is superseeded by the -tickSettings option. It permits specification that grids (rather than ticks) will be used for major divisions.
* -noScales—Specifies that no scales (i.e., no ticks, subticks, or numeric labels) will be plotted.
* -noBorder—Specifies that no border will be made around the plot region. Implies -noScales.

– Controlling text labels:

* -xLabel=[{@*parameterName* | *string*}][,offset=*value*][,scale=*value*][,edit=*string*]—Controls size, placement, and content of the x dimension label, which appears directly under the scale labels. The default text is of the form *symbol* (*units*), where the symbol and units are taken from the column or parameter definition fields in the SDDS header for the x data. If the symbol is blank, then the element name is used. Alternatively, the text may be taken from a named string parameter, or from a string that is given explicitly. In addition, the text may be edited using Toolkit editing commands (SDDS editing (4.40)). The offset and scale qualifiers allow changing the position and size of the label. The offset is specified as a fraction of the vertical dimension of the plot region. The scale is simply a multiplicative factor.
Note that if the value of the parameter *parameterName* changes from page to page in a file, and if separate pages are plotted in different panels, then the label for each panel will be different. If the pages are plotted together, the value of the parameter from the first page will be used.
* -yLabel—This switch has identical usage to -xLabel. -yLabel controls the y dimension label. The default text contains the y data names of all the columns and parameters being displayed. If the data all have the same units, the units are dis-

played as well. This information is taken from the appropriate entries in the SDDS header. The `offset` qualifier gives the label offset as a fraction of the horizontal dimension of the plot region.

* `-verticalPrint={up | down}`—Specifies the direction of print for the y dimension label. The default is up.
* `-title`—This switch has identical usage to `-xLabel`. The default text is from the `contents` field of the `description` command in the first file from which data is displayed.
* `-topTitle`—Normally, the title goes below the x dimension label. This switch directs that it be placed at the top of the plot, above the "topline label".
* `-topline`—This switch has identical usage to `-xLabel`. It is blank by default.
* `-filenamesOnTopline`—Directs that the topline text contain the names of the files from which data is displayed.
* `-labelSize=`*fraction*—Specifies a common size for all labels, including numeric labels. The *fraction* is the horizontal size of the characters as a fraction of the horizontal size of the plot region.
* `-noLabels`—Specifies that no labels (i.e., x and y dimension labels, title, and topline label) will be made.
* `-string={@`*parameterName* | *string*`},{xCoordinate=`*value* | `pCoordinate=`*value*`},{yCoordinate=`*value* | `qCoordinate=`*value*`}[,scale=`*factor*`][,angle=`*degrees*`][,justify=`*mode*`][,linetype=`*integer*`][,edit=`*string*`]`—Specifies display of string data on the plot. The string may either be extracted from a named string parameter or given explicitly. If the value of the parameter *parameterName* changes from page to page in a file, and if separate pages are plotted in different panels, then the label for each panel will be different. If the pages are plotted together, the value of the parameter from the first page will be used.

  The coordinates of the string may be specified either in users coordinates (i.e., x and y), or unit coordinates (i.e., p and q); the unit coordinates are (0,0) at the lower left of the plot region and (1,1) at the upper right. `scale` permits changing the size of the letters by a specified factor. `angle` permits changing the angle of the string; a value of 90 gives upward vertical print.

  Normally, text is "left bottom" justified, which means that the coordinates given are those of the left bottom corner of the first letter of the string. Justification may be changed with the `justify` qualifier, which accepts a mode string of the form `{ l | r | c}{t | b | c}`. The letters stand for Left, Right, Center, Top, and Bottom, respectively. The default justification would thus be specified as `justify=lb`.

  The text is normally creating using line type 0. This may be changed with the `linetype` option. As with the other labels, the text may be edited using Toolkit editing commands (`SDDS editing` (4.40)).
* `-dateStamp`—Directs that a time and date stamp be placed on the plot. It appears in the upper left corner of the plot.

– Altering or rearranging data prior to plotting:

* `-factor=[xMultiplier=`*value*`][,yMultiplier=`*value*`]`—Specifies that the x and/or y data for the present request will be multiplied by the given values. Note that it is the users responsibility to ensure that the units that are displayed are corrected, if required.

* -swap—Specifies that the x data will be plotted as y and vice-versa.
* -transpose—Specifies that the data matrix be transposed prior to plotting. This means, for example, that if the plot request specified N columns of y data and if the table contained M rows, one would get a plot of M quantities as a function of the index of the column. The implicit assumption is that the N columns contain comparable quantities. This would allow one to display, for example, how the quantities changed from row to row in the data. Each row of data thus organized is marked as a separate "subpage" (see the -groupBy and -separate switches), so that one can for example split rows onto separate panels.
* -mode={x | y}={linear | logarithmic | normalize | offset | specialScales}[,...]—Invokes one or more standard transformations of data, independently for x and y values. The linear mode is normally the default. logarithmic mode implies that the base-ten logarithmic of the appropriate values is taken prior to plotting. Normally, this does not produce log-type scales; use of the specialScales keyword together with the logarithmic keyword will obtain this. normalize mode directs that data be displayed after independent normalization to the interval [-1, 1]; to do this, the data is divided by the maximum absolute value in the data. offset mode directs that data be shifted so that the smallest value is identically zero.
* -stagger=[xIncrement=*value*] [,yIncrement=*value*] [,files] [,datanames]—Directs that data displayed on the same panel will be incrementally offset for display. This is useful in order to make mountain range plots, or to offset similar data for clarity. xIncrement and yIncrement are used to specify the increments for each dimension; zero is the default. Normally, only data from the same column or parameter is staggered, with the stagger amount increasing with each page in the file. The files qualifier directs incrementing the offset when plotting proceeds to a new file on the same panel. The datanames qualifier directs incrementing the offset when plotting proceeds to a new dataname (i.e., column or parameter name) within the same file on the same panel.
* -enumeratedScales=[interval=*integer*] [,scale=*factor*] [,rotate] [,editCommand=*string*]—Allows control of the display of enumerated value strings when the x data is of string type. interval=*N* specifies displaying and making a tick for every N^{th} enumerated value; the default is 1. scale specifies a factor by which to increase the size of the text. rotate specifies rotation of the printed text from the normal orientation to the optional orientation; if enumerated data is displayed along the x dimension, the normal (optional) orientation is vertical (horizontal) printing. These are reversed if the enumerated data is displayed along the y dimension.

– Creating legends:

* -legend=[{xy}symbol | {xy}description | filename | specified=*string* | parameter=*name*] [,editCommand=*string*] [,firstFileOnly] [,scale=*factor*]—Specifies creation of a legend for the datanames in the current request. By default, the legend text is the symbol field for the y data; if the symbol is blank, the dataname is used. xsymbol and ysymbol specify use of the x or y data symbols, or the datanames if the requested symbol is blank. xdescription and ydescription specify use of the indicated description fields. filename specifies use of the name of the file from which the data comes. specified=*string* specifies use of the given

string. parameter=*name* specifies use of the contents of the named string parameter. Any legend text may be editing using SDDS editing commandsSDDS Editing (??) via the editCommand qualifier. If firstFileOnly is given, only the first file in the request will have legends generated. If scale=*factor* is given, the legend text size is scaled by the given factor.

* -lSpace=*qmin,qmax,pmin,pmax*—Specifies the region in which legends will be placed. The coordinates are pspace coordinates. Since the legends are typically outside the pspace, the coordinates may be greater than unity. For example, the default values are [1.02, 1.18]x[0.0, 1.0]. This option is usually used to place the legend inside the pspace, or to extend the size of the lspace to accomodate long legend text.

– Creating overlays:

* -overlay=[{xy}mode=*mode*] [,{xy}factor=*value*] [,{xy}offset=*value*] [,{xy}center]—Normally, sddsplot displays all data on a single panel on the same scale. In some cases, one wants to overlay data that is on a different scale from other data on the panel. One way to do this is with the -overlay switch, which gives convenient control of how overlayed data is displayed. Any data in a plot request for which this switch is given will be overlayed as specified.
  The xmode and ymode options allow two types of scaling for x and y independently. A mode of normal means that the indicated data is treated normally. The default mode is unit, which means that the data is scaled so that its full range is equal to the full coordinate range of the plot in the appropriate (x or y) dimension.
  The data is further adjusted according to any additional qualifiers given. The center qualifiers offset the data so that the data is centered in the plot space; normally, zero in the data is mapped to zero in the user's coordinates. The factor qualifiers scale the data by the given factor about the center value. The offset qualifiers offset the data by specified amounts; if mode=normal, the offset is in user's coordinates, otherwise it is in pspace coordinates.
  Users needing only the factor facility should consider the -factor switch, since it is easier to use.

– Controlling plot panels:

* -newPanel—Specifies that the current plot request will start a new plot panel.
* -endPanel—Specifies that the current plot request will end the current plot panel.
* -layout=*hNumber,vNumber*[,limitPerPage=*integer*]—Specifies the layout of panels on each plot page. The maximum number of panels on any page is the product of *hNumber* and *vNumber*, which are the number of panels horizontally and vertically, respectively. The default is *hNumber*=1 and *vNumber=1*. If limitPerPage is given, then only the specified number of panels will appear on any page; for example, -layout=2,2,limit=3 would imply three panel spaces per page, with one left blank.

– Grouping, sorting, and separating data:

* -sever[=xgap=*value*] [,ygap=*value*]—For line plotting, sddsplot will normally connect points sequentially without regard for gaps in the data. The -sever switch specifies various means of locating gaps in data and directs lifting the "pen" whenever a gap occurs. If -sever is given without qualifiers, the pen is lifted whenever

the x value decreases; this is useful for plotting data where the x value is expected to increase monotonically for each group of points.

The xgap and ygap data are more sophisticated, and more generally applicable. For each dimension for which severing is requested, the pen is lifted whenever the absolute difference of two successive values exceeds a defined limit. This limit is specified either in absolute or fractional terms using the *value* entry. If *value* is positive, the gap threshold is equal to *value*. If *value* is negative, the gap threshold is *-value* times the mean spacing between successive points; a value of -1.5 has been found to work well for data that is roughly equispaced with occasional missing points.

* `-tagRequest={`*number* | *@parameterName*}—Specifies that data from the current requested will be tagged with either the given (generally floating-point) *number*, or with the values from the numeric parameter *parameterName*. Using the `-groupBy` and `-separate` options permits grouping and sorting of data by tag values. If a data set has multiple pages in the file, and if pages are split (see `-split` below), then parameter-tagged data will have the parameter value from the first page in each group of pages.

*

`-groupBy[=request][,tag][,fileIndex][,nameIndex][,page][subpage][,fileString][.`
—Specifies how internal data sets will be ordered. `-sortBy` might have been a more appropriate name for this switch. The qualifiers that appear in the list are shown in the order that corresponds to the default sorting. The file index is the sequential number within the request of the file from which the internal data set is taken; the file string is the name of the file. The name index is the sequential index within the request of the dataname group for the internal data set, while the name string is the name of the y data. The page is the sequential number in the file of the first SDDS data page from which data appears in the internal data set. The subpage is a sequential number within each internal data set, which allows subdivision of the internal data set. The request is the sequential number of the plot request that resulted in generation of the internal data set. The tag is a single user-supplied value or a value read from a parameter that is associated with each internal data set; by default, all data sets are tagged with the value 0. If a file is split into several internal data sets, each may have a different tag value if the tag is read from a parameter; in this case, the data sets are eached tagged with the value for the first included data page.

The order in which the qualifiers to `-groupBy` are given determines the priority of sorting by the various criteria. In the default ordering, data sets are sorted by request number, subsorted by tag (usually a null operation unless data is tagged by the user), subsubsorted by file index, subsubsubsorted by dataname index, etc. Each successive qualifier results in moving the indicated sort criterion to the next highest priority. Any qualifiers not given are retained in the default order.

If one wanted to bring together, for example, internal data sets with the same data name, one would give `-groupby=nameString`. In this case, the new sorting priority would be `nameString`, `request`, `tag`, etc.

* `-separate[={`*numberToGroup* | `groupsOf=`*number* | `fileIndex` | `fileString` | `nameIndex` | `nameString` | `page` | `subpage` | `request` | `tag}]`—Specifies how to separate internal data sets onto panels. If given with

no qualifiers, each internal data set is placed on a separate panel. If given with a single integer argument, or with the `groupsOf` qualifier, then the specified number of data sets appear on each panel; the data sets are assign to panels in the order determined by `-groupBy` or the default thereof.
If one of the other qualifiers is given, then panel separation occurs when the indicated criterion changes as the data sets are accessed in sorted order. Most commonly, one uses `-groupby=`*criterion* `-separate=`*criterion*. For example, one might want to group by filename and separate by filename.

* `-split={pages[,interval=`*integer*`]` | `parameterChange=`*name*`[,width=`*value*`][,offset=`*value*`]` | `columnBin=`*name*`,width=`*name*`[,start=`*value*`][,completely]}`—As discussed in the introductory sections, when `sddsplot` reads data for one dataname group from a file, it normally concatenates data from successive pages to form a single internal data set. This would mean, for example, that all of the data from the file would be displayed with the same linetype or symbol. The `-split` switch overrides this behavior, splitting the data into multiple internal data sets.
The simplest and most commonly-used way of doing this is to split the data page boundaries; this is done using the `-split=pages` mode. The optional `interval` specifies spliting after a specified number of page boundaries. Splitting data does not imply that the data will appear on separate plot panels, but allows this and other possibilities. (To separate page-split data onto panels, one uses `-separate=pages`, as discussed above.)
One can also page-split based on the value of a parameter, using `-split=parameterChange`. This directs that a new internal data set will be started wheneven the named parameter changes. For numeric parameters, the `width` and `start` qualifiers may be used. If `width` is specified, the change must exceed the given value before a split occurs. If `start` is specified, the reference value for changes is set to the given value; otherwise, the first parameter value is used. (For example, one might wish to split when a parameter changed by 5 units referenced from 2.5 units, giving boundaries of 7.5, 12.5, etc.; this would be obtained with `width=5,start=2.5`.)
The `columnBin` mode is different from the other two modes. Rather than splitting data into internal data sets at page boundaries, it groups or bins data into subpages according to the value in a specified numeric column. (It is appropriate only for plotting column data.) `columnBin` mode may be used with `pages` mode to split and subsplit data into pages and subpages. For example, one might have a data file with many pages of time-series data. One might want to plot each page separately, but within each page one might want to color-code the points according to some value in the table (e.g., a valid-data indicator). This would be accomplished using `-split=pages,columnBin=`*name*`,width=`*value* `-separate=pages -graph=dot,vary,eachPage`.
* `-omniPresent`—Specifies that the data sets from the current request will appear on all plot panels.

– Winnowing data:

* `-limit=[xMinimum=`*value*`][,xMaximum=`*value*`][,yMinimum=`*value*`][,yMaximum=`*value*`]`—Specifies limits to be placed on x and y values prior to plotting. Points beyond the

indicated limits are eliminated from the data prior to plotting. This complements the facility available from `-filter` and `-match` in that one need not specify the name of the data one is winnowing with. This permits easier filtering of data from many columns or parameters.

* `-sparse=`*interval*`[,`*offset*`]`—Specifies that only every $interval^{th}$ point will be used. If *offset* is not given, the first point in the internal data set is the first taken; otherwise, the $offset^{th}$ point is the first taken.
* `-sample=`*fraction*—Specifies random sampling of data to retain only the indicated fraction of the points. *fraction* gives the probability that any point will be used. Hence, the data actually used may vary from run to run since the random number generator is seeded with the system clock.
* `-clip=`*head*`,`*tail*`[,invert]`—Specifies removal of *head* points from the beginning and *tail* points from the end of each internal data set. If `invert` is given, the points that would have been removed are instead the only ones used.
* `-filter={column |`
  `parameter}`*,rangeSpec*`[,`*rangeSpec*`[,`*logicOperation*`...]]` — Specifies winnowing each internal data set based on numerical data in parameters or columns. A *range-spec* is of the form *name=lower-value,upper-value*`[,!]` , where `!` signifies logical negation. A point passes a `column`-based filter if the value in the named column is inside (or outside, if negation is given) the specified range, where the endpoints are considered inside. `parameter`-based filters are similar, except that the point passes only if the value of the named parameter for the page from which it comes is acceptable. One or more range specifications may be combined to give a accept/reject status by employing the *logic-operations*, & (logical and) and | (logical or).
* `match` — Specifies winnowing based on data in string parameters or columns. A *match-test* is of the form *name=matchingString*`[,!]`, where the matching string may include the wildcards * (matches zero of more characters) and ? (matches any one character). In other respects, `match` is just like `filter`.
  If the first character of *matchingString* is '@', then the remainder of the string is taken to be the name of a parameter or column. In this case, the match is performed to the data in the named entity.

- **special characters:** `sddsplot` supports Greek and mathematical characters in labels and strings through special sequences embedded in text strings. A similar mechanism is used to allow character-by-character control over size and positioning. The special sequences are of the form `$`*character*, where *character* may be one of the following:

  – `a, b, n`: provide subscript and superscript control. `a` puts the character Above the normal position (superscript), `b` puts the character Below the normal position (subscript), and `n` returns to Normal.

  – `g, r`: provide for switching between Greek and Roman character sets. `$g` switches into Greek mode, while `$r` switches back to Roman mode. The correspondance between Greek characters and the alphabet is shown in Figure ??. For example, to make a lower-case alpha, one would use `$ga$r`.

  – `s, e`: provide for switching between Special and normal characters. `$s` switches to special character mode, which provides mathematical and other symbols. Figure ?? shows the

correspondance between special characters and keyboard characters. For example, to make a ± symbol, one would employ `$sa$e`, while a right-pointing arrow would be obtained with `$s5$e`.

  - `i`, `d`: provide for Increasing and Decreasing the character size. The two sequences `$i` and `$d` are inverses of each other. `$i` increases the size of subsequent characters by 50%, while `$d` decreases the size of subsequent characters by $33\frac{1}{3}$%. These are seldom used, since `sddsplot` provides other means of controlling the size of characters in labels and strings.
  - `u`, `v`: provide for motion of the baseline Up and down by one half character height.
  - `t`, `f`: provide for making Taller and Fatter characters. `$t` makes characters twice as tall while maintaining width, while `$f` makes characters half as tall while maintaining width.
  - `h`: specifies moving back one half space.

- **environment variables:**
- **see also:**
  - Data for Examples (see 3.3)
  - `SDDS editing` (4.40)
- **author:** M. Borland, ANL/APS.
- **acknowledgements:** `sddsplot` uses device driver code from the program `GNUPLOT`, with modifications and enhancements made at Argonne. The GNUPLOT code is covered by a separate copyright, and is used by permission of the authors. See the `GNUPLOT_README` file included with the distribution for restrictions associated with this code.

  The GUI-interface X-windows program (`mpl_motif`) was written by K. Evans of ANL/APS.

4.26 sddsprintout

- **description:**
  sddsprintout provides formatted text output of data from columns and parameters. It is similar to sdds2stream, but provides better control of the appearance of the text.

- **examples:** Make a printout of APS design beta functions along with the tunes:

```
sddsprintout APS0.twi -column=ElementName -column='beta?'  -parameters='nu?'
```

- **synopsis:**

sddsprintout [-pipe=[input]] [*SDDSinput*] [*outputFile*] [-width=*integer*]
[-columns[=*nameList*[,format=*string*][,endsline]]
[-parameters[=*nameList*[,format=*string*][,endsline]] [-fromPage=*number*]
[-toPage=*number*] [-formatDefaults=*SDDStype=formatString*[,...]]

- **files:** *SDDSinput* is the SDDS file from which data is printed. *outputFile* is a file to which the printout will go; by default, the printout goes to the standard output.

- **switches:**
  - -pipe[=input] — The standard SDDS Toolkit pipe option.
  - -width=*integer* — Specifies the width of the output line in number of characters. The default is 130.
  - -columns=*nameList*[,format=*string*][,endsline] — Specifies the names of columns to appear in the printout. *nameList* may contain one or more comma-separated strings, each of which may contain wildcards. If more than one string is given, the list must be enclosed in parentheses, e.g., -columns='(betax,betay)'. The format qualifier may be used to specify a printf-style format string for the named columns; in this case, all of the columns must have the same data type. The format string should contain a width field, to ensure proper alignment of text; e.g., %30s rather than %s. If the endsLine qualifier is given, a line break is issued after the last column of the list is printed.
  - -parameters=*nameList*[,format=*string*][,endsline] — Specifies the names of parameters to appear in the printout. Identical to -columns in other respects.
  - fromPage=*number* — Specifies the first data page of the file that will appear in the printout. By default, the printout starts with data page 1.
  - toPage=*number* — Specifies the last page of the file that will appear in the printout. By default, the printout ends with the last data page in the file.
  - formatDefaults=*SDDStype=formatString*[,...] — Specifies default printf format strings for named SDDS data types. The *SDDStype* qualifier may be one of float, double, long, short, string, or *character*.

- **see also:**
  - Data for Examples (see 3.3)
  - sddsstream (??)

- **author:** M. Borland, ANL/APS.

4.27 sddsprocess

- **description:**
  sddsprocess operates on the data columns and parameters of an existing SDDS data set and creates a new data set. The program supports filtering and matching operations on both tabular data and parameter data, definition of new parameters and columns in terms of existing ones, units conversions, scanning of string data to produce numeric data, composition of string data from other data types, statistical and waveform analyses, and other operations.

- **examples:** Compute the square-roots of the beta-functions, which are the beam-size envelopes:

  ```
  sddsprocess APS.twi -define=column,sqrtBetax,"betax sqrt"
  -define=column,sqrtBetay,"betay sqrt"
  ```

  Compute the horizontal beam-size, given by the equation

  $$\sigma_x = \sqrt{\epsilon_x \beta_x + (\eta_x \sigma_\delta)^2}$$

  ```
  sddsprocess APS.twi -define=parameter,epsx,8.2e-9,units=nm
  -define=parameter,sigmaDelta,1e-3 -define=column,sigmax,"epsx betax *
  sigmaDelta etax * sqr + sqrt",units=m
  ```

- **synopsis:**

  sddsprocess [-pipe[=input][,output]] [*inputFile*] [*outputFile*] *options*

- **files:** *inputFile* is an SDDS file containing data to be processed. If no options are given, it is copied to *outputFile* without change. *Warning:* if no output filename is given, and if an output pipe is not selected, then the input file will be replaced.

- **switches:**

  - Data winnowing: Any number of the following may be used. They are applied in the order given. Note that -match and -test are the most time intensive; thus, if several types of winnowing are to be applied, these should be used last if possible.

    * -filter={column | parameter},*rangeSpec*[,*rangeSpec*[,*logicOperation*...]] — Specifies winnowing *inputFile* based on numerical data in parameters or columns. A *range-spec* is of the form *name=lower-value,upper-value*[,!] , where ! signifies logical negation. A page passes a given filter by having the named parameter inside (or outside, if negation is given) the specified range, where the endpoints are considered inside. A tabular data row passes a given filter in the analogous fashion, except that the value from the named column is used. One or more range specifications may be combined to give a accept/reject status by employing the *logic-operations*, & (logical and) and | (logical or).

* match — Specifies winnowing *inputFile* based on data in string parameters or columns. A *match-test* is of the form *name=matchingString*[,!], where the matching string may include the wildcards * (matches zero of more characters) and ? (matches any one character). In other respects, match is just like filter.
  If the first character of *matchingString* is '@', then the remainder of the string is taken to be the name of a parameter or column. In this case, the match is performed to the data in the named entity. For column-based matching, this is done row-by-row. For parameter-based matching, it is done page-by-page.
* -test={column | parameter},*test*[,autostop] — Specifies winnowing of *inputFile* based on a test embodied in an rpn expression. The expression, *test*, may use the names of any parameters or columns. If autostop is specified, the processing of the data set (or data page) terminates when the parameter-based (or column-based) expression is false.
* -clip=*head*,*tail*[,invert] — Specifies the number of data points to clip from the head and tail of each page. If invert is given, the clipping retains rather than deletes the indicated points.
* -sparse=*interval*[,*offset*] — Specifies sparing of each page with the indicated interval. That is, only every $interval^{th}$ row starting with row *offset* is copied to the output. The default value of *offset* is 0.
* -sample=*fraction*— Specifies random sampling of rows such that approximately the indicated fraction is kept. Since a random number generator is used that is seeded with the system clock, this will usually never be the same twice.

– rpn calculator initialization:

* -rpnDefinitionsFiles=*filename*... — Specifies a list of comma-separated filenames to be read in as rpn definitions files. By default, the file named in the RPN_DEFNS environment variable is read.
* -rpnExpression=*expression*[,repeat] — Specifies an rpn expression to be executed. If repeat is not specified, then the expression is executed before processing begins. If repeat is specified, the expression is executed just after each page is read; it may use values of any of the numerical parameters for that page. This option may be given any number of times.

– Scanning from, editing, printing to, and executing string columns and parameters:

* -scan={column | parameter},*newName*,*sourceName*,*sscanfString*[,*definitionEntries*] — Specifies creation of a new numeric column (parameter) by scanning an existing string column (parameter) using a sscanf format string. The default type of the new data is double; this may be changed by including a *definitionEntry* of the form type=*typeName*. With the exception of the name field, any valid namelist command field and value may be given as part of the *definitionEntries*.
* -edit={column | parameter},*newName*,*sourceName*,*edit-command* — Specifies creation of a new string column (parameter) called *newName* by editing an existing string column (parameter) *sourceName* using an emacs-like editing string. For details on editing commands, see SDDS editing (see 4.40).

* -print={column | parameter},*newName*,*sprintfString*,*sourceName*[,*sourceName*...][,*definitionEntries*] — Specifies creation of a new string column (parameter) by editing an existing string column (parameter) using a emacs-like editing command. For details on editing commands, see SDDS editing (see 4.40).
* -system={column | parameter},*newName*,*commandName*,*definitionEntries* — Specifies creation of a new string column (parameter) by executing an existing string column (parameter) using a subprocess. The first line of output from the subprocess is acquired and placed in the new column (parameter).

– Creation and modification of numeric columns and parameters:

* -convertUnits={column | parameter},*name*,*oldUnits*,*newUnits*,*factor* — Specifies units conversion for the column or parameter *name*. The *factor* entry the factor by which the values must be multiplied to convert them to the desired units. It is an error if *oldUnits* does not match the original units of the column or parameter. Eventually, the *factor* entry will be made optional by inclusion of conversion information in the program. This option may be given any number of times.
* -define={column | parameter},*name*,*equation*[,*definitionEntries*] — Specifies creation of a new column or parameter using an rpn expression to obtain the values. For parameters, any parameter value may be obtained by giving the parameter name in the expression. For columns, one may additionally get the value of any column by giving its name in the expression; the expression given for -define=column is essentially specifying a vector operation on columns with parameters as scalars. By default, the type of the new data is double. This and other properties of the new column or parameter may be altered by giving *definitionEntries*, which have the form *fieldName=value*; *fieldName* is the name of any namelist command field (except the name field) for a column or parameter, as appropriate. This option may be given any number of times.
  sddsprocess permits read access to individual elements of a column of data using the rpn array feature. For each column, an array of name &*ColumnName* is created; the ampersand is to remind the user that the variable &*ColumnName* is the address of the start of the array. To get the first element of a column named Data, one would use 0 &Data [. This will function only within or following a -define=column or -redefine=column operation. It is an error to attempt to access data beyond the bounds of an array.
* -redefine — This option is identical to -define except that the column or parameter already exists in the input. The equation may use the previous values of the entity being redefined by including the column name in the expression.
* -process=*mainColumnName*,*analysisName*,*resultName*[,description=*string*] [,symbol=*string*] [,weightBy=*columnName*] [,functionOf=*columnName*[,lowerLimit=*value* [,upperLimit=*value*]] [,head=*number*] [,tail=*number*] [fhead=*fraction*] [ftail=*fraction*] [position] [,offset=*value*] [,factor=*value*] — This option may be given any number of times. It specifies creation of a new parameter *resultName* by processing column *mainColumnName*. The column must contain numeric data. *mainColumnName* may contain wildcards, in which case the processing is applied to all matching columns containing numeric data. *resultName* may have a single occurence of the string "embedded in it; if so, *mainColumnName* is substi-

tuted. If wildcards are given in *mainColumnName*, then "if the description field is supplied, it may contain an embedded "substituted.
Recognized values for *analysisName* are:

- average, rms, sum, standardDeviation, mad — The arithmetic average, the rms average, the arithmetic sum, the standard deviation, and the mean absolute deviation. All may be possibly weighted.
- median — The median value.
- minimum, maximum, spread, smallest, largest — The minimum value, maximum value, spread in values, smallest value (minimum absolute value), and largest value (maximum absolute value). For all except spread, the position and functionOf qualifiers may be given to obtain the value in another column when *mainColumnName* has the extremal value.
- first, last — The values in the first and last rows of the page.
- count — The number of values in the page.
- baselevel, toplevel, amplitude — Waveform analysis parameters from histogramming the signal amplitude. baselevel is the baseline, toplevel is the height, and amplitude is height above baseline.
- risetime, falltime, center — The rise and fall times from the 10%-90% and 90%-10% transitions. center is the midpoint between the first 50% rising edge and the first following 50% falling edge after rising above 90% amplitude. Requires specifying a independent variable column with functionOf.
- fwhm, fwtm, fwha, fwta — Full-widths of the named column as a function of the independent variable column specified with functionOf. The letters 'h' and 't' specify Half and Tenth amplitude widths, while 'm' and 'a' specify Maximum value or Amplitude over baseline.
- zerocrossing — Zero-crossing point of the column named with functionOf of the column *mainColumnName*.

Qualifiers for this switch are:

- description=*string*, symbol=*string* — Specify the description and symbol fields for the new column.
- weightBy=*columnName* — Specifies the name of a column to weight values from column *mainColumnName* by before computing statistics.
- functionOf=*columnName* — Specifies the name of a column that *mainColumnName* is to be considered a function of for computing widths, zero-crossings, etc.
- lowerLimit=*value*, upperLimit=*value* — If functionOf is given, specifies winnowing of rows so that only rows for which the independent column data is above the lowerLimit and/or below the upperLimit are included in computations. No data is deleted from *mainColumnName* as it appears in the output.
- head=*number*, fhead=*fraction* — Specifies clipping of the head of the data prior to processing. head gives the number of points to clip, while fhead gives the fraction of the points to clip.
- tail=*number*, ftail=*fraction* — Specifies clipping of the tail of the data prior to processing. tail gives the number of points to clip, while ftail gives the fraction of the points to clip. If head and tail clipping are used, head clipping is performed first.

- `position` — For `minimum`, `maximum`, `smallest`, and `largest` analysis modes, specifies that the results should be the position at which the indicated value occurs. This position is the corresponding value of in column named with `functionOf`.
- `offset=`*value*, `factor=`*value* — Specify an offset and factor for modifying data prior to processing. By default, the offset is zero and the factor is 1. The equation is $x \rightarrow f * (x + o)$.

– Miscellaneous:

– `-ifis={column | parameter | array},`*name*`[,`*name*`...]`, `-ifnot={column | parameter | array},`*name*`[,`*name*`...]` — These options allow conditional execution. If any column that is named under a `ifis` option is not present, execution aborts. If any column that is named under a `ifnot` option is present, execution aborts.

– `summarize` — Specifies that a summary of the processing be printed to the screen.

– `verbose` — Specifies that informational printouts be provided during processing.

– `noWarnings` — Specifies suppression of warning messages.

- **author:** M. Borland, ANL/APS.

4.28 sddspseudoinverse

- **description:** `sddspseudoinverse` views the numerical tabular data of the input file as though it formed a matrix, and produces an output file with data corresponding to the pseudo-inverse of the input file matrix. At present the pseudo-inversion is done using a singular value decomposition. Other methods may be made available in the future.

  Command line options specifies the number of singular values to be used in the inversion process.

  The column names of the input file forms a string column in the output file. The command line option -root allows one to generate column names for the data in the output file. If this option is not present, then the data of the first string column of the input file are made into the column names of the output file. If no string column is present, then names are generated from an internal default.

  This command only operates on the first data set of a file.

- **examples:** The data in file LTP.R12 (matrix of R_{12}'s in a beamline called LTP, say) is inverted to give file LTP.InvR12 (useful for trajectory correction):

  ```
  sddspseudoinverse LTP.R12 LTP.InvR12
  ```

- **synopsis:**

  usage: `sddspseudoinverse` *inputfile outputfile*
  [`-minimumSingularValueRatio=`*realValue* | `-largestSingularValues=`*number*]
  `-root=`*string* `-symbol=`*string* `-verbose`

- **files:** The input file contains the data for the matrix to be inverted. The output file contains the data for the inverted matrix. If only one file is specified, then the input file is overwritten by the output.

- **switches:**

  - `-pipe[=input][,output]` — The standard SDDS Toolkit pipe option.
  - `-minimumSingularValueRatio=`*value* — Used to remove small singular values from the calculation. The smallest singular value kept is determined by multiplying this value of ratio with the largest singular value of the input matrix.
  - `-largestSingularValues=`*number* — Used to remove small singular values from the calculation. The largest *number* singular values are kept.
  - `-root=`*string* — The string specified is used to generate columns names in the output file. The first data column is called *string*0, the second *string*1, etc.
  - `-symbol=`*string* — The string specified is assigned to the symbol field of data column definitions.
  - `-ascii` — Produces an output in ascii mode. Default is binary.
  - `-verbose` — Prints out incidental information to stderr.

- **author:** L. Emery ANL

4.29 sddsquery

- **description:** sddsquery prints a summary of the SDDS header for a data set. Also prints bare lists of names of defined entities, suitable to use with shell scripts that need to detect the existence of entities in the data set.

- **examples:** Get information on the contents of a file:

  ```
  sddsquery APS.twi
  ```

  Get a list of the column names only:

  ```
  sddsquery APS.twi -columnList
  ```

  Get a list of the column names into a shell variable

  ```
  set names = `sddsquery APS.twi -columnList -delimiter=" "`
  ```

- **synopsis:**

  sddsquery *SDDSfilename* [*SDDSfilename...*] [{-arrayList | -columnList | -parameterList | -version}] [-delimiter=*delimitingString*] [-appendUnits]

- **switches:** Normal operation of sddsquery results in a printout summarizing the header of each file. If one of the options is given, however, this printout will not appear. Instead, the selected list of names appears for each file.

  - arrayList — Requests that a list of array names be printed to the standard output, one name per line.
  - columnList — Requests that a list of column names be printed to the standard output, one name per line.
  - parameterList — Requests that a list of parameter names be printed to the standard output, one name per line.
  - -version — Requests that the SDDS version number of the file be printed to the standard output.
  - -delimiter=*delimitingString* — Requests that listed items be separated by the given string. By default, the delimiter is a newline.
  - -appendunits — Requests that the units of each item be printed directly following the item name.

- **see also:**

  - Data for Examples (see 3.3)

- **author:** M. Borland, ANL/APS.

4.30 sddsregroup

- **description:** sddsregroup swaps the row indexing and page indexing of data in an SDDS file. That is, the i^{th} row of all data pages in the input file are collected and made into the i^{th} data page of the output file.

- **examples:** The file bpm.sdds contain the beam position monitor (bpm) readback as a function of time for a series of consecutive bpms in a beamline. The defined columns are Time and x. The parameters are bpmIndex. The file bpm.sdds is regrouped to produce data sets of x vs bpmIndex for each time value. The output is suitable to plot as a movie with sddsplot.

  ```
  sddsregroup bpm.sdds bpm.movie -newparameters=Time -newcolumns=bpmIndex
  ```

- **synopsis:**

  sddsregroup [-pipe=[input][,output]] *inputfile outputfile*
  [-newparameters=*oldcolumnname*,...] [-newcolumns=*oldparametername*,...]
  [-warning] [-verbose]

- **files:** The input file contains the data sets to be regrouped. The output file contains the regrouped data. If only one file is specified, then the input file is overwritten by the output.

- **switches:**

  - -pipe[=input][,output] — The standard SDDS Toolkit pipe option.
  - newparameters — specifies which columns of the input file will become parameters in the output file. By default no new parameters are created, and all columns of the input file are transfered to the output file.
  - newcolumns — specifies which parameters of the input file will become columns in the output file. The columns will necessarily be duplicated in all pages. By default all parameters values are lost.

- **author:** L. Emery ANL

4.31 sddsselect

- **description:** `sddsselect` excludes or includes rows from one file based on the presence of matching data in another file. It is similar to `sddsxref`, but unlike that program does not import data from the second file.

- **examples:** Use a list of quadrupole names to get just the Twiss parameters are the quadrupoles:

  ```
  sddsselect APS.twi quadNames.sdds APSquad.twi -match=ElementName -reuse
  ```

  where `ElementName` is a column in both `APS.twi` and `quadNames.sdds` giving the name of a magnet. Use the same file to get the Twiss parameters everywhere but at the quadrupoles:

  ```
  sddsselect APS.twi quadNames.sdds APSquad.twi -match=ElementName -reuse
  -invert
  ```

- **synopsis:**

  `sddsselect [-pipe[=input][,output]] [`*input1*`] `*input2*` [`*output*`]`
  `{-match=`*columnName1*`[=`*columnName2*`] | -equate=`*columnName1*`[=`*columnName2*`] }`
  `[-invert] [-reuse[=page][,rows]]`

- **files:** *input1* is an SDDS file from which rows of data will be selected for inclusion in *output*. If *input1* contains multiple pages, they are processed separately. *input2* is an SDDS file containing rows of data to use in selecting data from *input1*. *Warning:* if *output* is not given and `-pipe=output` is not specified, then *input1* will be replaced.

- **switches:**

  - `-pipe[=input][,output]` — The standard SDDS Toolkit pipe option.
  - `-match=`*columnName1*`[=`*columnName2*`]` — Specifies the names of string columns from *input1* and *input2* to compare. If *columnName2* is not given, it taken to be the same as *columnName1*. Data in *columnName* is taken from *input1* and *columnName2* from *input2*. For each row in a page of *input1*, a match for the string in *columnName1* is sought in any row of *columnName2*. If a match is found, the row is accepted.
  - `-equate=`*columnName1*`[=`*columnName2*`]` — Identical to `-match`, except the columns contain numerical data.
  - `-invert` — Specifies that only rows that have no match or equal should be selected for output.
  - `-reuse[=rows][,page]` — By default, if *input1* contains multiple pages, each is selected against the corresponding page of *input2*. In addition, each row of *input2* is matched or equated to only one row of *input1*. If `-reuse=page` is given, then each page of *input1* is selected against the first page of *input2*. If `-reuse=rows` is given, each row of *input2* can select any number of rows of *input1*.

- **sddsmselect** — `sddsmselect` is a variant of `sddsselect` that permits multiple `-match` and `-equate` options for more sophisticated cross-referencing. In other respects, the program is used just like `sddsmselect`. `sddsselect` is much faster, however, for single-criterion matching or equating.

- **see also:**
  - Data for Examples (see 3.3)
  - `sddsxref` (4.38)
- **author:** M. Borland, ANL/APS.

4.32 sddsslopes

- **description:** sddsslopes makes straight line fits of column data of the input file with respect to a selected column used as independent variable. The output file contains a one-row data set with columns of slopes and intercept defined for each input data column specified for fitting. Errors on the slope and intercept may be calculated as an option.

- **examples:** The file corrector.sdds contains beam position monitors (bpms) readbacks as a function of corrector setting. The defined columns are CorrectorSetpoint and the series bpm1, bpm2, etc. The bpm response to the corrector setpoints are calculated with

  ```
  sddsslopes corrector.sdds corrector.slopes
  -independentVariable=CorrectorSetpoint  -columns='bpm*'
  ```

  where all columns that match with the wildcard expression bpm* is selected for fitting.

- **synopsis:**

  usage: slopes [-pipe=[input][,output]] *inputfile outputfile*
  -independentVariable=*parametername* [-columns=*list-of-names*]
  [-excludeColumns=*list-of-names*] -sigma[=generate] -verbose

- **files:** The input file contains the tabular data for fitting. Only the first data set is read. For optional error processing, additional columns of sigma values associated with the data to be fitted must be present. These sigma column must be named *name*Sigma or Sigma*name*, the former one being searched first.

  The output file contains one data set with one row. The columns defined have names such as *name*Slope, and *name*Intercept where *name* is the name of the fitted data. If only one file is specified, then the input file is overwritten by the output. A string column called IndenpendentVariable is defined containing the name of the indepedent variable.

- **switches:**

  - -pipe[=input][,output] — The standard SDDS Toolkit pipe option.
  - -independentVariable=*parametername* — name of independent variable (default is the first valid column)
  - -columns=*list-of-names* — columns to be individually paired with independentVariable for straight line fitting
  - -excludeColumns=*list-of-names* — columns to exclude from fitting
  - -sigma[=generate] — calculates errors by interpreting column names *name*Sigma or Sigma*name* as sigma of column *name*. If these columns don't exist then the program generates a common sigma from the residual of a first fit, and refits with these sigmas. If option -sigma=generate is given, then sigmas are generated from the residual of a first fit for all columns, irrespective of the presence of columns *name*Sigma or Sigma*name*.
  - -ascii — make output file in ascii mode (binary is the default)
  - -verbose — prints some output to stderr

- **author:** L. Emery ANL

4.33 sddssmooth

- **description:**

  sddssmooth smooths columns of data using multipass nearest-neighbor averaging. Any number of columns may be sorted using an arbitrary number of passes and nearest-neighbors. The smoothed data may be put in place of the original data, or included as a new column.

- **examples:** Smooth data in a Fourier transform:

  ```
  sddssmooth data.fft data.peaks -column=FFTamplitude
  ```

- **synopsis:**

  sddssmooth [-pipe=[input][,output]] [*inputfile*] [*outputfile*]
  -columns=*name*[,*name...*] [-points=*oddInteger*] [-passes=*integer*] [-newColumns]

- **files:**

  inputFile contains the data to be smoothed. *outputFile* contains all of the array and parameter data from *inputFile*, plus at least one column for every column in *inputFile*. Columns that are not smoothed will appear unchanged in *outputFile*. If *inputFile* contains multiple pages, each is treated separately and is delivered to a separate page of *outputFile*.

- **switches:**

  - -pipe[=input][,output] — The standard SDDS Toolkit pipe option.
  - -columns=*columnName*[,*columnName...* — Specifies the names of the column to smooth. The names may include wildcards.
  - -points=*oddInteger* — Specifies the number of points to average to create a smoothed value for each point. The default is three, which implies replacing each point by the average of itself and its two nearest neighbors.
  - -passes=*integer* — Specifies the number of smoothing passes to make over each column of data. The default is 1. In the limit of an infinite number of passes, every point will tend toward the average value of the original data.
  - -newColumns — Specifies that the smoothed data will be placed in new columns, rather than replacing the data in each column with the smoothed result. The new columns are given names of the form *columnName*Smoothed, where *columnName* is the original name of a column.

- **see also:**

  - sddsdigfilter (4.14)

- **author:** M. Borland, ANL/APS.

4.34 sddssort

- **description:**
  sddssort sorts the tabular data section of a data set by the values in named columns. Any number of columns may be involved in the sort, and sorting order may be individually specified.

- **examples:**
  Sort the APS Twiss file into alphabetical order by element name:

  ```
  sddssort APS.twi APS.twi.sorted -column=ElementName
  ```

  Same, but keep only one instance of each row with the same element name:

  ```
  sddssort APS.twi APS.twi.sorted -column=ElementName -unique
  ```

- **synopsis:**

  sddssort [-pipe=[input][,output]] [*SDDSinput*] [*SDDSoutput*]
  -column=*name*[,{increasing | decreasing}] [-column...] [-unique]
  [-noWarnings]

- files:
  SDDSinput is an SDDS file to be sorted. If it contains multiple data pages, they are treated separately. *Warning:* if *SDDSoutput* is not given and -pipe=output is not specified, then *SDDSinput* will be replaced.

- **switches:**

  - -pipe=[input][,output] — The standard SDDS pipe option.
  - -column=*name*[,{increasing | decreasing}] — Requests that the column *name* be used to order the rows of each tabular data section. Each subsequent column request specifies a subsort of the ordering produced by the previous requests. The increasing and decreasing keywords may be given to specify the desired ordering of the (sub)sort, with increasing order being the default.
  - -unique — Specifies that for any rows that are identical in the sort column values, only the first should be included in the output file.
  - -noWarnings — Suppresses warning messages.

- **author:** M. Borland, ANL/APS.

4.35 sddssplit

- **description:**
  `sddssplit` breaks up an SDDS file into one or more separate files, each containing only a single page of data. This may be useful in those instances where a tool or program only processes the first page of a file.

- **examples:**
  Split a Twiss parameter file into separate files:

  ```
  sddssplit APS.twi
  ```

- **synopsis:**

  `sddssplit {-pipe[=input] | `*inputFile*`} [{-binary | -ascii}] [-digits=`*number*`] [-rootname=`*string*`] [-extension=`*string*`] [-firsttable=`*number*`] [-lasttable=`*number*`] [-interval=`*number*`]`

- **files:** *inputFile* is an SDDS file to be split. By default, the output files are created by appending the page number to a "rootname" and adding an extension. That is, the output files have names *rootnamePage.extension*. The default rootname is the name of *inputFile*, while the default extension is "sdds". By default, *Page* is printed using "less the extension.

- **switches:**
  - `-pipe[=input][,output]` — The standard SDDS Toolkit pipe option.
  - `-binary`, `-ascii` — Specifies binary or ASCII output, with binary being the default.
  - `-digits=`*number* — Specifies the number of digits to be used in creating filenames. Leading zeros are included.
  - `-rootname=`*string* — Specifies the rootname to be used in creating filenames.
  - `-extension=`*string* — Specifies the extension to be used in creating filenames.
  - `-firsttable=`*number* — Specifies the first table or page of data to use.
  - `-lasttable=`*number* — Specifies the last table or page of data to use.
  - `-interval=`*number* — Specifies the interval between pages that are used.

- **see also:**
  - `sddsbreak` (4.4)
  - `sddscombine` (4.8)

- **author:** M. Borland, ANL/APS.

4.36 sddstranspose

- **description:** sddstranspose views the numerical tabular data of the input file as though it formed a matrix, and produces an output file with data corresponding to the transpose of the input file matrix. In other words, the columns of tabular data of the input file become rows in the output file. String column data are not transposed but are stored as string parameters in the output file. Operating on the output file with a second sddstranpose command essentially recovers the original input file.

  The column names of the input file are collected and made into a string column called OldColumnNames in the output file. The command line option -root allows one to generate column names for the data in the output file. If this option is not present, then the data of the first string column, if any string columns are present, of the input file are made into the column names of the output file. If no string column is present, then names are generated from an internal default.

  This command operates on the first data set of a file, and the following data sets until one is found with a row count differing from that of the first data set.

- **examples:** The data in file LTP.R12 (matrix of R_{12}'s in a beamline called LTP, say) is transposed to give file LTP.R12.trans:

  ```
  sddstranspose LTP.R12 LTP.R12.trans
  ```

- **synopsis:**

  sddstranspose [-pipe=[input][,output]] *inputfile outputfile* -root=*string* -symbol=*string* [-ascii] -verbose

- **files:** The input file contains the data for the matrix to be transposed. The output file contains the data for the transposed matrix. If only one file is specified, then the input file is overwritten by the output.

- **switches:**

  - -pipe[=input][,output] — The standard SDDS Toolkit pipe option.
  - -root=*string* — The string specified is used to generate columns names in the output file. The first data column is called *string*0, the second *string*1, etc.
  - -symbol=*string* — The string specified is assigned to the symbol field of data column definitions.
  - -ascii — Produces an output in ascii mode. Default is binary.
  - -verbose — Prints out incidental information to stderr.

- **author:** L. Emery ANL

4.37 sddsvslopes

- **description:** sddsvslopes makes straight line fits of vectorized data in the input file with respect to a selected parameter used as independent variable. The simplest example of vectorized data is a data set with one parameter and two columns, one string column of rootnames and one numerical column of data. The fitting is looped over rows across all the data sets in the input file (using a selected parameter as the independent vairable). The output file contains vectorized slopes and intercepts data for each column specified in the input file.

- **examples:** The file corrector.sdds contains vectorized beam position monitor (bpm) readbacks as a function of corrector setting. The defined parameter is CorrectorSetpoint. The defined columns are Rootname and x. Each row of the data set correspond to a different bpm. The bpm response to the corrector setpoints are calculated with

```
sddsvslopes corrector.sdds corrector.vslopes
-independentVariable=CorrectorSetpoint  -columns=x
```

- **synopsis:**

  sddsvslopes *SDDSinputfile SDDSoutputfile* -independentVariable=*parametername* [-columns=*list-of-names*] [-excludeColumns=*list-of-names*] -slopeErrors -verbose

- **files:** The input file contains the tabular data for fitting. The column Rootname must be present.

  The output file contains one data set of vectorized slopes and intercept data. The Rootname column from the input file is transfered to the output file. The column names are *name*Slope, and *name*Intercept where *name* is the name of the fitted data. If only one file is specified, then the input file is overwritten by the output. A string parameter called IndenpendentVariable is defined containing the name of the indepedent variable.

- **switches:**

  - -pipe[=input][,output] — The standard SDDS Toolkit pipe option.
  - -independentVariable=*parametername* — name of independent variable (default is the first valid column)
  - -columns=*list-of-names* — columns to be individually paired with independentVariable for straight line fitting
  - -excludeColumns=*list-of-names* — columns to exclude from fitting
  - -sigma[=generate] — calculates errors by interpreting column names *name*Sigma or Sigma*name* as sigma of column *name*. If these columns don't exist then the program generates a common sigma from the residual of a first fit, and refits with these sigmas. If option -sigma=generate is given, then sigmas are generated from the residual of a first fit for all columns, irrespective of the presence of columns *name*Sigma or Sigma*name*.
  - -ascii — make output file in ascii mode (binary is the default)
  - -verbose — prints some output to stderr

- **author:** L. Emery ANL

4.38 sddsxref

- **description:** 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. The user may specify which columns of the second data set to take and which to leave. The user may also transfer parameter and array data.

- **synopsis:**

  sddsxref [-pipe[=input][,output]] [*input*] [*xRefFile*] [*output*]
  [-equate=*columnName* | -match=*columnName*] [-reuse[=rows][,page]]
  [-take=*columnName*,...] [-leave=*columnName*,...] [-transfer={parameter |
  array},*name*[,*name*...] [-ifis={column | parameter | array},*name*[,*name*...]
  [-ifnot={column | parameter | array},*name*[,*name*...]

- **files:** *input* is the data set to which data is being added. *xRefFile* is the data set from which data is being taken. Warning: if *output* is not given and if -pipe=out is not specified, *input* is overwritten. For pipe input, the first file listed is taken to be *xRefFile*. For pipe input and output, the only file listed is *xRefFile*.

- **switches:**

  - -equate=*columnName*, -match=*columnName*— These options specify the name of a column that exists in both *input* and *xRefFile*. For match, the column must contain string data, while for the equate the column must contain numeric data. For each row in *input*, sddsxref searches *xRefFile* to find the first row for which the match column is identical or the equate column is equal, as appropriate. This row is the one from which any data is taken for addition to the row in *input*. If neither of these options is given, then rows are taken sequentially from *xRefFile* for each row of *input*.
  - -reuse[=rows][,page] — By default, each row from *xRefFile* is matched to one row in *input*. If -reuse=rows is given, each row from *xRefFile* may be matched to any number of rows in *input*. Also by default, each page of *input* is matched with the corresponding *page* of *xRefFile*. If -reuse=page is given, then each page of *input* is matched anew to the first page of *xRefFile*. The two qualifiers may be given together.
  - -take=*columnName*,..., -leave=*columnName*,...— These options specify which columns of *xRefFile* to extract from a matching or equal row of *xRefFile* for addition to a row of *input*. Wildcards may be given in the column names. By default, all columns not in *input* are taken. If take is employed, only the named columns will be taken. In either case, no column specified under leave will be taken. -leave=* causes no columns to be taken.
  - -transfer={parameter | array},*name*[,*name*...] — This option, which may be given multiple times, specifies the names of parameters and arrays to be transfered. Wildcards are not presently supported in this option.
  - -ifis={column | parameter | array},*name*[,*name*...], -ifnot={column | parameter | array},*name*[,*name*...] — These options allow conditional execution. If any column that is named under a ifis option is not present, execution aborts. If any column that is named under a ifnot option is present, execution aborts.

- **sddsmxref**— `sddsmxref` is a variant of `sddsxref` that permits multiple `-match` and `-equate` options for more sophisticated cross-referencing. In other respects, the program is used just like `sddsmxref`. `sddsxref` is much faster, however, for single-criterion matching or equating.
- **see also:**
  - Data for Examples (see 3.3)
  - `sddsxref` (4.38)
- **author:** M. Borland, ANL/APS.

4.39 sddszerofind

- **description:**

  sddszerofind finds the locations of zeroes in a single column of an SDDS file. This is done by finding successive rows for which a sign change occurs in the "dependent column", or any row for which an exact zero is present in this column. For each of the "independent columns", the location of the zero is determined by linear interpolation. Hence, the program is really interpolating multiple columns at locations of zeros in a single column. This single column is in a sense being looked at as a function of each of the interpolated columns.

- **examples:** Find zeroes of a Bessel function, $J_0(z)$:

  ```
  sddszerofind J0.sdds J0.zero -zero=J0 -column=z
  ```

  Find zeroes of a Bessel function, $J_0(z)$, and simultaneously interpolate $J_1(z)$ at the zero locations:

  ```
  sddszerofind J0.sdds J0.zero -zero=J0 -column=z,J1
  ```

  (This isn't the most accurate way to interpolate $J_1(z)$, of course.)

- **synopsis:**

  sddszerofind [-pipe=[input][,output]] [*inputfile*] [*outputfile*]
  -zeroesOf=*columnName* [-columns=*columnNames*] [-slopeOutput]

- **files:** *inputFile* contains the data to be searched for zeroes. *outputFile* contains columns for each of the independent quantities and a column for the dependent quantity. Normally, each dependent quantity is represented by a single column of the same name. If output of slopes is requested, additional columns will be present, having names of the form *columnName*Slope.

  If *inputFile* contains multiple pages, each is treated separately and is delivered to a separate page of *outputFile*.

- **switches:**

  - -pipe[=input][,output] — The standard SDDS Toolkit pipe option.
  - -zeroesOf=*columnName* — Specifies the name of the dependent quantity, for which zeroes will be found.
  - -columns=*columnNames* — Specifies the names of the independent quantities, for which zero locations will be interpolated. Generally, there is only one of these. *columnNames* is a comma-separated list of optionally wildcarded names.
  - -slopeOutput — Specifies that additional columns will be created containing the slopes of the dependent quantity as a function of each independent quantity. This can be useful, for example, if one wants to pick out only positive-going zero-crossings.

- **see also:**

  - sddsinterpolate (??)

- **author:** M. Borland, ANL/APS.

4.40 SDDS Editing

This manual page does not describe a program, but rather a facility that is common to several programs. In particular, several SDDS programs use a common syntax for specifying editing of string data. The editing commands for these programs are composed of a series of subcommands of the form [*count*]commandLetter[*commandSpecificData*] As indicated, the *count* and *commandSpecificData* are optional.

The commands are as follows:

[*n*]f — move forward 1 or *n* characters.

[*n*]b — move backward 1 or *n* characters.

[*n*]d — delete the next character or *n* characters.

[*n*]F — move forward 1 or *n* words.

[*n*]B — move backward 1 or *n* words.

[*n*]D — delete the next word or *n* words.

a — Go to the beginning of the string.

e — Go to the end of the string.

[*n*]i-*delim-text-delim-* — Insert *text*, delimited by the character *-delim-* 1 or *n* times. For example, "i/thisString/" would insert "thisString" once.

[*n*]s-*delim-text-delim-* — Search for *text*, delimited by the character *-delim-* 1 or *n* times. The position is left at the end of the search string. *-delim-* may be any character except a question mark.

S-*delim-text-delim-* — Search for *text*, delimited by the character *-delim-*, leaving the position at the start of the search string. *-delim-* may be any nonspace character except a question mark.

[*n*]s?-*delim-text-delim-* — Search for *text*, delimited by the character *-delim-* 1 or *n* times. Abort all subsequent editing if the search fails. If the search suceeds, leave the position at the end of the search string. *-delim-* may be any nonspace character except a question mark.

S?-*delim-text-delim-* — Search for *text*, delimited by the character *-delim-*. Abort all subsequent editing if the search fails. If the search suceeds, leave the position at the start of the search string. *-delim-* may be any nonspace character except a question mark.

[*n*]k — Delete forward from the present position 1 or *n* characters, placing them in the kill buffer.

[*n*]K — Delete forward from the present position 1 or *n* words, placing them in the kill buffer.

z*char* — Delete forward from the present position up to the first occurence of the character *char*, placing the deleted text in the kill buffer.

[*n*]Z*char* — Delete 1 or *n* times up to and including the character *char*, placing the deleted text in the kill buffer.

[*n*]y — Yank the kill buffer into the string 1 or *n* times.

[*n*]%-*delim-text1-delim-text2-delim*- — Replace *text1* with *text2* 1 or *n* times starting at the present position. -*delim*- may be any nonspace character. For example, "10%/c/C/" would capitalize the next 10 occurences of the character 'c'.

- **see also:**
  - `sddsprocess` (4.27)
  - `sddsplot` (4.25)
  - `sddsconvert` (4.11)

4.41 rpn Calculator Module

- **description:**

  Many of the SDDS toolkit programs employ a common Reverse Polish Notation (RPN) calculator module for equation evaluation. This module is based on the rpn programmable calculator program. It is also available in a commandline version called rpnl for use in shell scripts. This manual page discusses the programs rpn and rpnl, and indicates how the rpn expression evaluator is used in SDDS tools.

- **examples:**
  Do some floating-point math using shell variables: (Note that the asterisk (for multiplication) is escaped in order to protect it from interpretation by the shell.)

```
set pi = 3.141592
set radius = 0.15
set area = 'rpnl $pi $radius 2 pow \*'
```

  Use rpn to do the same calculation:

```
rpn> 3.141592 sto pi
rpn> 0.15 sto radius
rpn> radius 2 pow pi *
0.070685820000000
rpn> quit
```

- **synopsis:**

  rpn [*filenames*]
  rpnl *rpnExpression*

- **Overview of rpn and rpnl:**

  rpn is a program that places the user in a Reverse Polish Notation calculator shell. Commands to rpn consist of generally of expressions in terms of built-in functions, user-defined variables, and user-defined functions. Built-in functions include mathematical operations, logic operations, string operations, and file operations. User-defined functions and variables may be defined "on the fly" or via files containing rpn commands.

  The command rpn *filename* invokes the rpn shell with *filename* as a initial command file. Typically, this file would contain instructions for a computation. Prior to execution of any files named the commandline, rpn first executes the instructions in the file named by the environment variable RPN_DEFNS, if it is defined. This file can be used to store commonly-used variable and function definitions in order to customize the rpn shell. This same file is read by rpnl and all of the SDDS toolkit programs that use the rpn calculator module. An example of such a file is included with the code.

  As with any RPN system, rpn uses stacks. Separate stacks are maintained for numerical, logical, string data, and command files.

  rpnl is essentially equivalent to executing rpn, typing a single command, then exiting. However, rpnl has the advantage that it evaluates the command and prints the result to the screen

without any need for user input. Thus, it can be used to provide floating point arithmetic in shell scripts. Because of the wide variety of operations supported by the rpn module and the availability of user-defined functions, this is a very powerful feature even for command shells that include floating point arithmetic.

Built-in commands may be divided into four broad categories: mathematical operations, logical operations, string operations, and file operations. (There are also a few specialized commands such as creating and listing user-defined functions and variables; these will be addressed in the next section). Any of these commands may be characterized by the number of items it uses from and places on the various stacks.

- Mathematical operations:
  * Using rpn variables:
    The `sto` (store) function allows both the creation of rpn variables and modification of their contents. rpn variables hold double-precision values. The variable name may be any string starting with an alphabetic character and containing no whitespace. The name may not be one used for a built-in or user-defined function. There is no limit to the number of variables that may be defined.
    For example, `1 sto one` would create a variable called one and store the value 1 in it. To recall the value, one simply uses the variable name. E.g., one could enter `3.1415925 sto pi` and later enter `pi` to retrieve the value of π.
  * Basic arithmetic: `+ - * /`
    These operations all take two values from the numeric stack and push one result onto the numeric stack. For example, `5 2 -` would push 5 onto the stack, push 2 onto the stack, then push the result (3) onto the stack.
  * Basic scientific functions: `sin cos acos asin atan atan2 sqrt sqr pow exp ln`

    With the exception of `atan2` and `pow`, these operations all take one item from the numeric stack and push one result onto that stack.
    `sin` and `cos` are the sine and cosine functions, while `asin`, `acos`, and `atan` are inverse trigonometic functins. `atan2` is the two-argument inverse tangent: `x y atan2` pushes the value atan(y/x) with the result being in the interval $[-\pi, \pi]$.
    `sqrt` returns the positive square-root of nonnegative values. `sqr` returns the square of a value. `pow` returns a general power of a number: `x y pow` pushes x^t onto the stack. Note that if y is nonintegral, then x must be nonnegative.
    `exp` and `ln` are the base-e exponential and logarithm functions.
  * Special functions: `Jn Yn cei1 cei2 erf erfc lngam`
    `Jn` and `Yn` are the Bessel functions of integer order of the first and second kind[7]. Both take two items from the stack and place one result on the stack. For example, `x i Jn` would push $J_i(x)$ onto the stack. Note that $Y_n(x)$ is singular at x=0.
    `cei1` and `cei2` are the 1st and 2nd complete elliptic integrals. The argument is the modulus k, as seen in the following equations (the functions K and E are those used by Abramowitz[7]).

$$\text{cei1(k)} = \text{K}(\text{k}^2) = \int_0^{\pi/2} \frac{\text{d}\theta}{\sqrt{1 - \text{k}^2 \sin^2\theta}}$$

$$\text{cei2(k)} = \text{E}(\text{k}^2) = \int_0^{\pi/2} \sqrt{1 - \text{k}^2 \sin^2\theta}\,\text{d}\theta$$

`erf` and `erfc` are the error function and complementary error function. By definition, erf(x) + erfc(x) is unity. However, for large x, `x erf 1 -` will return 0 while `x erfc` will return a small, nonzero value. The error function is defined as[7]:

$$\mathrm{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$$

Note that $\mathrm{erf}(x/\sqrt{2})$ is the area under the normal Gaussian curve between $-x$ and x.

`lngam` is the natural log of the gamma function. For integer arguments, `x lngam` is $\ln((x-1)!)$. The gamma function is defined as[7]:

$$\Gamma(x) = \int_0^\infty t^{x-1} e^{-t} dt$$

* Numeric stack operations: `cle n= pop rdn rup stlv swap view ==`
  `cle` clears the entire stack, while `pop` simply removes the top element. `==` duplicates the top item on the stack, while `x n=` duplicates the top x items of the stack (excluding the top itself). `swap` swaps the top two items on the stack. `rdn` (rotate down) and `rup` (rotate down) are stack rotation commands, and are the inverse of one another. `stlv` pushes the stack level (i.e., the number of items on the stack) onto the stack. Finally, `view` prints the entire stack from top to bottom.
* Random number generators: `rnd grnd`
  `rnd` returns a random number from a uniform distribution on [0, 1]. `grnd` returns a random number from a normal Gaussian distribution.
* Array operations: `mal [ ]`
  `mal` is the Memory ALlocation command; it pops a single value from the numeric stack, and returns a "pointer" to memory sufficient to store the number of double-precision values specified by that value. This pointer is really just an integer, which can be stored in a variable like any other number. It is used to place values in and retrieve values from the allocated memory.
  `]` is the memory store operator. A sequence of the form *value index addr* `]` results in *value* being stored in the *index* position of address *addr*. *value*, *index*, and *addr* are consumed in this operation. Indices start from 0.
  Similarly, *index addr* `[` value pushes the value in the *index* position of address *addr* onto the stack. *index* and *addr* are consumed in this operation.
* Miscellaneous: `tsci int`
  `tsci` allows one to toggle display between scientific and variable-format notation. In the former, all numbers are displayed in scientific notation, whereas in the later, only sufficiently large or small numbers are so displayed. (See also the `format` command below.)
  `int` returns the integer part of the top value on the stack by truncating the noninteger part.

– **Logical operations:** `! && < == > ? $ vlog ||`

* Conditional execution: `?` The question-mark operator functions to allow program branching. It is meant to remind the user of the C operator for conditional evaluation of expressions. A conditional statement has the form

? *executeIfTrue* : *executeIfFalse* $

The colon and dollar sign function as delimiters for the conditionally-executed instructions. The ? operator pops the first value from the logic stack. It branches to the first set of instructions if this value is "true", and to the second if it is "false".

* Comparisons: < == >

These operations compare two values from the numeric stack and push a value onto the logic stack indicating the result. Note that the values from the numeric stack are left intact. That is, these operations push the numeric values back onto the stack after the comparison.

* Logic operators: && || !

These operators consume values from the logic stack and push new results onto that stack. && returns the logical and of the top two values, while || returns the logical or. ! is the logical negation operator.

* Miscellaneous: vlog

This operator allows viewing the logic stack. It lists the values on the stack starting at the top.

– **String operations:** "" =str cshs format getformat pops scan sprf vstr

* Stack operations: "" =str pops vstr

To place a string on the string stack, one simply encloses it in double quotation marks. =str duplicates the top of the string stack. pops pops the top item off of the string stack. vstr prints (views) the string stack, starting at the top.

* Format operations: format getformat

format consumes the top item of the string stack, and causes it to be used as the default printf-style format string for printing numbers. getformat pushes onto the string stack the default printf-style format string for printing numbers.

* Print/scan operations: scan sprf

scan consumes the top item of the string stack and scans it for a number; it pushes the number scanned onto the string stack, pushes the remainder of the string onto the string stack, and pushes true/false onto the logic stack to indicate success/failure. sprf consumes the top of the string stack to get a sprintf format string, which it uses to print the top of the numeric stack; the resulting string is pushed onto the string stack. The numeric stack is left unchanged.

– **File operations:** @ clos fprf gets open puts

* Command file input: @

The @ operator consumes the top item of the string stack, pushing it onto the command file stack. The command file is executed following completion of processing of the current input line. Command file execution may be nested, since the files are on a stack. The name of the command file may have options appended to it in the format *filename,option.* Presently, the only option recognized is 's', for silent execution. If not present, the command file is echoed to the screen as it is executed. Example: "commands.rpn,s" @ would silently execute the rpn commands in the file commands.rpn.

* Opening and closing files: clos open

open consumes the top of the string stack, and opens a file with the name given in

that element. The string is of the format *filename,option*, where *option* is either 'w' or 'r' for write or read. `open` pushes a file number onto the numeric stack. This should be stored in a variable for use with other file IO commands. The file numbers 0 and 1 are predefined, respectively, as the standard input and standard output.
`clos` consumes the top of the numeric stack, and uses it as the number of a file to close.

* Input/output commands: `fprf gets puts`
These commands are like the C routines with similar names. `fprf` is like fprintf; it consumes the top of the string stack to get a fprintf format string for printing a number. It consumes the top of the numeric stack to get the file number, and uses the next item on the numeric stack as the number to print. This number is left on the stack.
`gets` consumes the top of the numeric stack to get a file number from which to read. It reads a line of input from the given file, and pushes it onto the string stack. The trailing newline is removed. If successful, `gets` pushes true onto the logic stack, otherwise it pushes false.
`puts` consumes the top of the string stack to get a string to output, and the top of the numeric stack to get a file number. Unlike the C routine of the same name, a newline is *not* generated. Both `puts` and `fprf` accept C-style escape sequences for including newlines and other such characters.

– **author:** M. Borland, ANL/APS.

4.42 SDDS Wildcard Conventions

This manual page does not describe a program, but rather a facility that is common to several programs. In particular, several SDDS programs use a common convention for wildcards in element names.

The characters *, ?, [,], and ^ are used for wildcard operations.

* matches any zero or more characters. A sequence like *a matches zero or more characters up to the first occurence of a.

? matches any one character.

[*rangeSpec*] matches any one character in *rangeSpec*. *rangeSpec* is composed on any number of explicit characters, plus character ranges specified as *firstChar*-*lastChar*, which matches any character between *firstChar* and *lastChar* inclusive in the ASCII character set. For example, [a-z] would match a lower case alphabetic character, while [a-z][A-Z][0-9] would match any alphanumeric character.

[^*rangeSpec*] matches any one character not in *rangeSpec*.

- see also:
  - sddschanges (4.5)
  - sddsconvert (4.11)
  - sddscorrelate (4.12)
  - sddsenvelope (4.15)
  - sddsfft (4.17)
  - sddsoutlier (4.22)
  - sddsplot (4.25)
  - sddsprintout (4.26)
  - sddsprocess (4.27)
  - sddssmooth (4.33)
  - sddsxref (4.38)
  - sddszerofind (4.39)

5 Manual Pages for APS-Specific Programs

5.1 awe2sdds

- **description:** Converts a file in awe self-describing format to SDDS. This is of interest to only a few users at APS, as awe format has been superseeded by SDDS and is rarely used.
- **example:** To convert awe format Twiss parameter data from an old version of elegant:

```
awe2sdds APS.awe APS.sdds -labelColumnName=ElementName
```

- **synopsis:**

awe2sdds *inputFile outputFile* [-labelColumnName=*string*] [-asciiOutput]

- **files:** *inputFile* is an awe-format file, the SDDS equivalent of which is written to *outputFile*. The "auxiliary values" of the awe file are converted into SDDS parameters. The awe tables are converted into SDDS tabular data, all columns being double precision except the "row label", which becomes a string column.
- **switches:**
  - -labelColumnName=*string* — Requests that the awe row label be given the name *string*. By default, the row label is placed in a column named "row-label".
  - -asciiOutput — Requests that output be in ASCII. By default, the output is binary.
- **author:** M. Borland, ANL/APS.

5.2 col2sdds

- **description:** Converts a file in `column` self-describing format[?] to SDDS. This is of interest to APS users only, some of whom still have programs that generate `column`-format files.
- **synopsis:**

  `col2sdds` *inputFile outputFile* `[-fixMplNames]`

- **files:** *inputFile* is a `column`-format file, the SDDS equivalent of which is written to *outputFile*. The "auxiliary values" of the `columns` file are converted into SDDS parameters. The `column` table is converted into SDDS tabular data, all columns begin double precision except the "row label", which becomes a string column.
- **switches:**
  - `-fixMplNames` — Requests that any column or parameter names in the input file that contain `mpl` character set escape sequences be "fixed". This results in simpler names. The escape sequences are always retained in definition of the symbol for each column or parameter, and hence will appear on graphs as expected.
- **author:** M. Borland, ANL/APS.

5.3 sdds2mpl

- **description:** `sdds2mpl` extracts data columns or parameters from an SDDS data set and creates `mpl` data files. The program allows creation of `mpl` labels from SDDS parameters. This tool is primarily of interest to APS users, some of whom still use the older `mpl` Toolkit. It may be of interest to others who are interested in a simple format for use with programs that don't need the full power of SDDS protocol. Such applications can use `sdds2mpl` and `mpl2sdds` to mediate between themselves and SDDS-compliant programs.

- **example:**

```
sdds2mpl APS.twi -rootname=APS -output=column,z,betax -output=column,z,betay
```

- **synopsis:**

sdds2mpl [*SDDSfile*] [-pipe[=input]] [-rootName=*string*] [-separateTables]
-output={column | parameter},*xName*,*yName*[,{*syName* | *sxName*,*syName*}]
[-announceOpenings] [-labelParameters=*name*[=*format*]][...]

- **files:** *SDDSfile* is the name of an SDDS file from which `mpl`-format files will be made. Each `mpl` file contains two to four columns of data.

- **switches:**

  - `-pipe[=input]` — The standard SDDS Toolkit pipe option.
  - `-announceOpenings` — Requests that an informational message be printed whenever a new output file is opened.
  - `-rootName=`*string* — Gives the rootname for constructing output filenames.
  - `-separateTable` — Requests that tabular-data column output from separate pages in the SDDS data set go to separate files.
  - `-labelParameters=`*name*`[=`*format*`]][...]` — Gives the names and optional `printf` format specifications for parameters that will be printed on the title line of the `mpl` files.
  - `-output{column | parameter},`*xName*`,`*yName*`[,{`*syName* | *sxName*`,`*syName*`}]` — Requests that the named columns or parameters be put into a `mpl` file or set of files. If `-separate` is not given or if the data is for parameters, the name of the file is *rootname*`_xName_yName.out`. For column output, if `-separate` is given, the names of the files are *rootname*`_`*N*`_xName_yName.out`, where *N* is the page number. This option may be given any number of times.

- **see also:**

  - Data for Examples (see 3.3)
  - `mpl2sdds` (5.4)

- **author:** M. Borland, ANL/APS.

5.4 mpl2sdds

- **description:** Adds `mpl[?]` data files to an SDDS data set. `mpl` is a simple data format used by the `mpl` Toolkit, which is now largely superseeded by SDDS and will not be supported in the future.

- **example:**

```
mpl2sdds APS_s_betax.out APS_s_betay.out -output=APSbetas.sdds
```

- **synopsis:**

`mpl2sdds` *mplFile* `[`*mplFile...*`]` `-output=`*SDDSFile* `[-erase]`

- **files:** Any number of *mplFile* arguments may be given. These name files in `mpl` format, which has between two and four columns of data. `sdds2mpl` attempts to add all of the columns from each `mpl` data file to the data set. However, a column that has the same name as an existing column will not be used. By default, the data in the `mpl` files is added to *SDDSFile*, if it exists already.

- **switches:**

  - `-output=`*SDDSfile* — Specifies that data be added to file *SDDSfile*. If the file does not exist, it is created.
  - `-erase` — Specifies that if *SDDSFile* exists already, it should be erased prior to adding any data to the data set. By default, the data in *SDDSFilename* is retained.

- **see also:**

  - `sdds2mpl` (5.3)

- **author:** M. Borland, ANL/APS.