User's Guide for SDDS Toolkit Version 1.21

Michael Borland
Advanced Photon Source

March 10, 2003

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

This document describes Version 1.20 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 they are many. Some of these 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 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 line is taken to end each data set. An invalid line (e.g., too few rows or invalid data) is treated as an error, and the rest of the file is ignored.

2.4 Structure of SDDS Binary Data Pages

Since the user may wish to read or write SDDS data sets without using the SDDS function library, a more detailed description of the structure of the data pages is provided.

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.

All binary data is stored in the machine representation, except for strings. Strings are stored in a variable-record format that consists of a long signed integers followed by a sequence of characters. The number of characters is equal to the value in the signed integer. Note that the SDDS library has features that allow recognition and interpretation of big- and little-endian data representations, which are not described here.

The first element in the data page is the row count, which is a long signed integer. This exists even in files that do not contain any columns.

If parameters have been defined, then their values follow in the order that the `parameter` definitions appear in the header. Note that if a parameter is define as "fixed-value" in the `parameter` definition, then its value will not appear.

If arrays have been defined, then they follow next, in the order that the `array` definitions appear in the header. For each array, a series of long signed integers is first given, one for each dimension of the array. For example, a two-dimensional array would have two integers, specifying the size of the array in the first and second dimension. If the two integers are, say, `n` and `m` in that order, then the declaration of the array in a C program would be, for example, `a[n][m]`. Elements of the array are put in the file in C storage order, which means that the outermost index varies fastest as the data is accessed in storage order.

If tabular-data columns have been defined, then the table data follows. Data is stored as rows, so that data for columns is intermixed. The order of the columns is the same as the order of the `column` definitions in the header.

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, LS-287, September 2001.

[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.

[8] S. Reiche, *NIM* A 429 (1999) 242.

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 Mathematical Operations Tools

- `sddsbaseline` (4.9) — Remove baselines from column data. Example application: determining the noise level in a video signal and subtracting it from the signal.

- `sddschanges` (4.12) — 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.

- `sddscliptails` (4.14) — Remove tails from column data, where a tail is dubious data on either side of a peak. Example application: removing halo or noise tails from video images of beam spots.

- `sddsderiv` (4.23) — Does numerical differentiation of multiple data columns versus a single column, with optional error propogation.

- `sddsinteg` (4.38) — 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` (4.39) — 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.

- `sddsnormalize` (4.44) — Normalizes data in multiple columns using various types of normalization factors, determined from the data.

- `sddspeakfind` (4.46) — 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` (4.50) — 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` (4.61) — Smooths columns of data using multipass nearest-neighbor averaging. Example application: reducing noise in a frequency spectrum prior to finding peaks.

- `sddszerofind` (4.68) — 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 Statistics Tools

- `sddscorrelate` (4.22) — 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.

- `sddsdistest` (4.26) — Performs statistical tests on data to determine whether the data is drawn from any of various distributions. Example application: determining if a component failure rate matches a Poisson distribution.

- `sddsenvelope` (4.28) — 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.

- `sddseventhist` (4.29) — Analyzes labeled events in a dataset to provide histograms of the occurences of each type of event. Can also histogram the overlap off all types of events with a single type of event. Example application: correlating the occurence times of alarm signals to determine which alarms usually occur together.

- `sddshist` (4.36) — 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` (4.37) — 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.

- `sddsmultihist` (4.41) — Does histograms of multiple columns of data. Example application: finding the distribution of a set of similar readbacks that are sampled many times.

- `sddsoutlier` (4.45) — 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`.

- `sddsprocess` (4.50) — 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.

- `sddsrowstats` (4.54) — Computes row-by-row statistics across multiple columns of data, creating new columns to contain the statistics. Example application: finding the mean value of a set of readout values from time-series data collection, where each readout is in a separate column.

- `sddsrunstats` (4.55) — Computes running or blocked statistics of multiple columns. Example applications: smoothing noisy data; finding running averages and error bars for time-series data.

- `sddsshiftcor` (4.59) — Computes correlation coefficients between column data as a function of shift position of a reference column. Example application: finding correlations among time series data collected from process variables, including the possibility of time-lags between the process variables due to physical or data collection effects.

3.1.3 Digital Signal Processing Tools

- `sddsconvolve` (4.21) — Does FFT convolution, deconvolution, and correlation. Example application: computing the ideal impulse response of a system after you've measured the response to a pulse.

- `sddsdigfilter` (4.25) — 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.

- `sddsfdfilter` (4.32) — Performs frequency-domain filtering of column data. Example application: applying a filter that is specified as a table of attenuation and phase as a function of frequency.

- `sddsfft` (4.33) — 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.

- `sddsnaff` (4.43) — Does Numerical Analysis of Fundamental Frequencies, a more accurate method of determining principle frequencies in signals than the FFT.

3.1.4 Data Fitting Tools

- `sddsexpfit` (4.31) — 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.

- `sddsgenericfit` (4.34) — Does generic fits to column data. Example application: fitting the sum of two gaussians.

- `sddsgfit` (4.35) — Does gaussian fits to column data. Example application: finding the width of a resonance, or the rms size of a beam profile.

- `sddsmpfit` (4.40) — Does polynomial fits to multiple columns data, including error analysis. Will do fits to specified orders, fits of specified symmetry, and adaptive fitting. Use `sddspfit` as a simpler and somewhat more capable alternative for fitting a single column of data.

- `sddspfit` (4.47) — Does polynomial fits to column data, including error analysis. Will do fits to specified orders, fits of specified symmetry, and adaptive fitting.

3.1.5 Data Manipulation Tools

- `sddsbreak` (4.10) — 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.

- `sddscast` (4.11) — change the data type of the elements in an SDDS file.

- `sddscollapse` (4.15) — 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`.

- `sddscollect` (4.16) — Reorganizes tabular data from the input file to bring data from several groups of similarly named columns together into a single column per group. Example application: collecting several statistical analyses of many columns into a single column per analysis type.

- `sddscombine` (4.17) — 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` (4.20) — 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. N.B.: it is *not* recommended to use `sddsconvert` to convert a binary SDDS file to ASCII, then strip the header off and read the ASCII file. This completely bypasses the self-describing aspects of the SDDS file and is not robust. If the program that creates the SDDS file is changed so that the columns are created in a different order, the program that reads the ASCII file will produce unexpected results. Use `sdds2plaindata` (4.6), `sddsprintout` (4.49), or `sdds2stream` (4.8) for conversion to non-self-describing files. In this way, you can assure the order of the data is fixed.

- `sddsderef` (4.24) — Allows dereferencing (i.e., de-indexing) of array and column data. Example application: converting a column of integer indexes into a column of equivalent text messages, where the text messages are stored once each in an array in the input file (for space-savings).

- `sddsendian` (4.27) — Converts from little-endian to big-endian and vice-versa. Example application: converting binary data from the native format to a format used on another type of computer prior to transferring the data to the other computer.

- `sddsexpand` (4.30) — Expands a data set into one page for each row, with column data promoted to parameter data. Essentially the inverse of `sddscollapse` (4.15).

- `sddsregroup` (4.53) — Swaps the row indexing and page indexing of data in an SDDS file. That is, the ith row of the jth data page in the input file becomes the jth row of the ith data page of the output file. Example application: viewing the long-term evolution of a repeatedly-sampled waveform at each point in the waveform.

- `sddstranspose` (4.65) — Transposes the tabular data in the input file, so that the output file contains one column for each row in the input. Example usage: tranpose an orbit response matrix as part of preparing to use it for feedback.

- `sddsmatrixmult` (4.42) — Multiplies the tabular data in the two input files to produce a file containing a matrix of the product. Example usage: Multiply a vector of errors with a correction matrix to obtain a vector of corrections to apply in a step-by-step feedback system.

- `sddsselect` (4.57) — 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` (4.62) — Sorts the tabular data section of a data set by the values in named columns. Optionally eliminates duplicate rows.

- `sddssplit` (4.63) — 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` (4.67) — 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.

3.1.6 Graphics Tools

- `sddscontour` (4.19) — 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` (4.48) — 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.

3.1.7 Miscellaneous Tools

- `elegant2genesis` (4.2) — Processes particle output from the particle tracking code `elegant`[2] and makes a file suitable for use as the BEAMFILE with the FEL code GENESIS[8].

- `sddssampledist` (4.56) — Draws samples from one or more probability distributions. Suitable for making input particle distributions for tracking codes, for example, using user-defined probability distributions.

- `sddssequence` (4.58) — Creates an SDDS data set of arithmetic sequences. Example application: generating values for an independent variable, whose values can be used by `sddsprocess` to produce a mathematical function.

- `sddscongen` (4.18) — 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`.

- `sddstimeconvert` (4.64) — Converts time data between seconds-since-epoch and calendar breakdown formats. Example application: finding the year, month, and day corresponding to a system time value.

3.1.8 File Protocol Conversion Tools

- `csv2sdds` (4.1) — Converts CSV (Comma-Separated-Values) data 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.
- `hdf2sdds` (4.3) — Converts Hierarchical Data Format (HDF) to SDDS.
- `lba2sdds` (??) — Converts Spiricon Laser Beam Analyzer files to SDDS.
- `plaindata2sdds` (4.4) — Converts plain data file with simple formatting to SDDS.
- `sdds2math` (4.5) — 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].
- `sdds2plaindata` (4.6) — Converts SDDS data to a plain data file with simple formatting. This is one of the recommended ways to convert SDDS data to plain ASCII or binary data for input to non-compliant programs. The advantage of using `sdds2plaindata` is that you can guarantee that the data is emitted in a fixed order.
- `sdds2spreadsheet` (4.7) — Converts SDDS data to a format accepted by the Excel and Wingz spreadsheets. Obsolete. Use `sddsprintout` (4.49) instead.
- `TFS2sdds` (??) — Converts MAD/LEP TFS files to SDDS.

3.1.9 Text-based Data-review Tools

- `sdds2stream` (4.8) — 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. `sdds2stream` may be used to convert SDDS data to plain ASCII text.

- `sddsprintout` (4.49) — Makes customized printouts from column, parameter, and array data in an SDDS data set. Also makes spreadsheet-compatible data and plain ASCII text. Example application: making a nicely-formatted printout of data that needs to be reviewed manually.

- `sddsquery` (4.52) — 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, ÂPS0.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*.
- `P`*quadrant*`P`*number*`y` — 16 single-precision readouts of the vertical 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 csv2sdds

- **description:** Converts Comma-Separated-Values (CSV) data and similar data to SDDS. CSV data is commonly used by spreadsheet programs.

- **example:**

```
csv2sdds data.csv -columnData=name=x,type=float,units=m
-columnData=name=Name,type=string data.sdds
```

- **synopsis:**

```
csv2sdds [CSVfile] [SDDSfile] [-pipe[=in][,out]] [-asciiOutput] [-spanLines]
[-maxRows=integer] [-delimiters=start=character,end=character]
[-separator=character] -schFile=SCHfilename |
[-columnData=name=string,type=string[,units=string] ...]
```

- **files:** *CSVfile* is a comma-separated-values file. Such a file consists of M rows each containing N items of data, forming N columns. The items on each row are separated by commas (or by a specified separator). The items may also be delimited by double quotation marks (or by specified delimiters).

  SDDSfile is the SDDS output that is created.

  The optional *SCHfilename* is a way of specifying the column headers. The file is expected to contain a series of lines of the form *tag=valueList*, where *valueList* is a comma-separated list of one or more items. Lines not matching this format are ignored. The *tag* may be one of the following:

  - `Filetype`: optional. If given, must have *valueList* of `Delimited`.
  - `Delimiter`: optional. If given, the first character of *valueList* is used for the start and end delimiters.
  - `Separator`: optional. If given, the first character of *valueList* is used for the separator.
  - `CharSet`: optional. If given, must have *valueList* of `ascii`.
  - `Field`*N*, where *N* is an integer: one or more required. The integers *N* must be consecutive. The first item in *valueList* is taken as the column name. The second item is interpreted as the data type. At present, only the `Float` data type is actually interpreted as anything other than character data. All others are treated as character string types. If needed, `sddsprocess` may be used to process the resulting string columns to produce other data types.

- **switches:**

  - `-pipe[=in][,out]` — The standard SDDS Toolkit pipe option.
  - `-asciiOutput` — Specifies ASCII output.
  - `-delimiters=start=`*character*`,end=`*character* — Specifies start and end delimiters for data. The default is to use a double-quotation mark for both.
  - `-separator=`*character* — Specifies separator to use. The default is a comma.

- -schFile=*filename* — Specifies the name of a SCH file specifying the format of the CSV file. I don't know what SCH stands for, but apparently some PC programs generate such files.
- -columnData=name=*string*,type=*string*[,units=*string*] — Specifies the name and data type of a column of data in the CSV file. One of these options should be given for each column in the input file, in the same order as the columns appear in that file.
- -spanLines — Specifies that the program should ignore line breaks in parsing the input data.
- maxRows=*integer* — The maximum number of rows expected. This allows optimization of the program, but isn't essential.

- **see also:**
  - sdds2stream (4.8)
  - sddsprocess (4.50)
  - plaindata2sdds (4.4)
- **author:** M. Borland, ANL/APS.

4.2 elegant2genesis

- **description:** `elegant2genesis` analyzes particle output data from `elegant` and prepares a "beamfile" for input to GENESIS[8], a 3-D time-dependent FEL code by S. Reiche. The beamfile is contains slice analysis of the particle data, and may be useful in other applications as well.

- **synopsis:**

```
elegant2genesis inputfile outputfile [-pipe=[in][,out]] [-textOutput]
[-totalCharge=coulombs | -chargeParameter=name] [-wavelength=meters |
-slices=integer] [-steer]
[-removePTails=deltaLimit=value[,fit][,beamOutput=filename]]
```

- **files:**
  - *inputfile* — A particle output file from `elegant` or any other program that uses the same column names and units.
  - *outputfile* — Contains the slice analysis, suitable for use with SDDS-compliant GENESIS.

- **switches:**
  - `-pipe[in][,out]` — The standard SDDS toolkit pipe option.
  - `-textOutput` — Requests text output instead of SDDS output, which may be useful for input to non-SDDS-complaint versions of GENESIS.
  - `-totalCharge=`*coulombs* — Gives the total charge of the beam in Coulombs.
  - `-chargeParameter=`*name* — Gives the name of a parameter in *inputfile* where the total charge in the beam is given.
  - `-wavelength=`*meters* — This option is misnamed. It is actually the slice length in meters.
  - `-slices=`*integer* — The number of analysis slices to use.
  - `-steer` — If given, then the transverse centroids for the bulk beam are all set to zero. The relative centroid offsets of the slices are, of course, unchanged.
  - `removePTails=deltaLimit=`*value*`[,fit][,beamOutput={\em filename}\verb`]— — Remvoes the momentum tails from the beam. `deltaLimit` is the maximum absolute value of $(p - \langle p \rangle)/\langle p \rangle$ that will be accepted. If `fit` is given, then a linear fit to p as a function of t is performed, and removal is based on the residuals from that fit. If `beamOutput` is given, then the filtered beam data is written to the named file for review.

- **author:** R. Soliday, ANL/APS.

4.3 hdf2sdds

- **description:** Converts Hierarchical Data Format (HDF) to SDDS.
- **example:**

```
hdf2sdds ./ data.hdf data.sdds -withIndex
```

- **synopsis:**

hdf2sdds [*HDF filepath*] [*HDF filename*] [*SDDSfile*] [{-arraysPreferred | -columnsPreferred}] [{-binary | -ascii}] [-withIndex] [-verbose]

- **files:**

  HDF filepath is the name of the directory containing the HDF file.

  HDF filename is the filename of the HDF file excluding the path.

  SDDSfile is the SDDS output that is created.

- **switches:**
  - `{-arraysPreferred | -columnsPreferred}` — Requests that the output be in the form of an SDDS array or SDDS columns. By default, -columnsPreferred is used.
  - `{-binary | -ascii}` — Requests that the output be binary or ASCII.
  - `-withIndex` — An index column is added to the output file.
  - `-verbose` — Prints out steps as the program runs.
- **author:** R. Soliday, ANL/APS.

4.4 plaindata2sdds

- **description:** Converts plain data files with a simple format to SDDS.
- **example:**

```
plaindata2sdds data.input data.output -inputMode=ascii "-separator= "
-parameter=time,long -column=x,double -column=y,double
```

- **synopsis:**

```
plaindata2sdds [Inputfile] [Outputfile] [-pipe[=in][,out]]
[-inputMode=<ascii|binary>] [-outputMode=<ascii|binary>]
[-separator=character] [-noRowCount] [-order=<rowMajor|columnMajor>]
[-parameter=name,type[,units=string][,desc=string][,symbol=string] ...]
[-column=name,type[,units=string][,desc=string][,symbol=string] ...]
[-nowarnings]
```

- **files:** *Inputfile* is a file that is similar to SDDS files in that it contains parameter and column data. However this file does not contain SDDS header information. The column data does not need to be preceded by a row count but it is recommended. Also the column data can be separated by a user supplied character. White space on either side of the separator is allowed. Binary plaindata files are also allowed.

  Outputfile is the SDDS output that is created.

- **switches:**
  - `-pipe[=in][,out]` — The standard SDDS Toolkit pipe option.
  - `-inputMode=<ascii|binary>` — The plain data file can be read in ascii or binary format.
  - `-outputMode=<ascii|binary>` — The SDDS data file can be written in ascii or binary format.
  - `-separator=`*`character`* — In ascii mode the columns of the plain data file are separated by the given character.
  - `-noRowCount` — The row count is not included prior to the beginning of the column data. If the plain data file is a binary file then the row count must be included.
  - `-order=<rowMajor|columnMajor>` — Row major order is the default. Here each row of the plain data file consists of one element from each column. In column major order each column is located entirely on one row.
  - `-parameter=name,type[,units=string][,description=string][,symbol=string]` — Add this option for each parameter in the plain data file.
  - `-column=name,type[,units=string][,description=string][,symbol=string]` — Add this option for each column in the plain data file.

- **see also:**
  - `sdds2plaindata` (4.6)
  - `csv2sdds` (4.1)

- **author:** R. Soliday, ANL/APS.

4.5 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*] [-pipe[=input][,output]] [-comments] [-verbose] [-format=*printfString*]

- **switches:**
  - pipe — The standard SDDS Toolkit pipe option.
  - 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.

4.6 sdds2plaindata

- **description:** Converts SDDS data to a plain data file with simple formatting.
- **example:**

```
sdds2plaindata data.input data.output -outputMode=binary "-separator= "
-parameter=time -column=x -column=y
```

- **synopsis:**

```
sdds2plaindata [Inputfile] [Outputfile] [-pipe[=in][,out]]
[-outputMode=<ascii|binary>] [-separator=string] [-noRowCount]
[-order=<rowMajor|columnMajor>] [-parameter=name,[,format=string] ...]
[-column=name,[,format=string] ...]  [-nowarnings]
```

- **files:** *Inputfile* is the SDDS input file.

  Outputfile is a file that is similar to SDDS files in that it contains parameter and column data. However this file does not contain SDDS header information. The column data does not need to be preceded by a row count but it is recommended. Also the column data can be separated by a user supplied string. Binary plaindata files are also allowed.

- **switches:**
  - -pipe[=in][,out] — The standard SDDS Toolkit pipe option.
  - -outputMode=<ascii|binary> — The plain data file can be written in ascii or binary format.
  - -separator=*string* — In ascii mode the columns can be separated by the given string.
  - -noRowCount — The number of rows will not be included in the plain data file. If binary mode is used the number of rows will always be written to the file.
  - -order=<rowMajor|columnMajor> — Row major order is the default. Here each row consists of one element from each column. In column major order each column is written entirely on one row.
  - -parameter=*name*,[,format=*string*] — Add this option for each parameter to add to the plain data file.
  - -column=*name*,[,format=*string*] — Add this option for each column to add to the plain data file.

- **see also:**
  - plaindata2sdds (4.4)

- **author:** R. Soliday, ANL/APS.

4.7 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.

  Notes:

  1. Excel lines must be shorter than 255 characters. The Wingz delimiter can only be `\t`.
  2. The program `sddsprintout` with the `-spreadSheet` option is intended to replace the function of `sdds2spreadsheet`. It allows greater control of what data is output and how it is formatted.

- **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] [-pipe[=input][,output]]
[-delimiter=string] [-all] [-verbose]
```

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

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

4.8 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*...] |
  -arrays=*arrayName*[,*arrayName*...] } [-filenames] [-rows] [-noquotes]
  [-ignoreFormats] [-description]

- **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.
  - `arrays=`*arrayName*`[,`*arrayName*`...]` — Specifies the names of the arrays for which output is desired.
  - `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.
  - `ignoreFormats` — Specifies that the format data supplied in the file is to be ignored. Guarantees printing of floating point data to full precision.
  - `description` — Specifies printing of the description data for the data set.

- **see also:**
  - Data for Examples (see 3.3)
  - `sddsprintout` (4.49)
  - `sddsconvert` (4.20)
  - `sddsprocess` (4.50)
- **author:** M. Borland, ANL/APS.

4.9 sddsbaseline

- **description:** `sddsbaseline` performs baseline removal for SDDS column data. Several methods of determining the baseline are provided.
- **examples:** Remove baselines from a video image organized with each scan line in a separate column. The baseline is determined by looking at 10 points at either end of each line and averaging the pixel count for these points.

```
sddsbaseline image.sdds image1.sdds -columns=VideoLine* -select=endpoints=10
-method=average
```

- **synopsis:**

```
sddsbaseline [input] [output] [-pipe=[in][,out]] [-columns=listOfNames]
-select={endPoints=number | -outsideFWHA=multiplier | -antiOutlier=passes}
-method={fit | average} [-nonnegative
[-despike=passes=number,widthlimit=value] [-repeats=count]]
```

- **files:** *input* is an SDDS file containing one or more pages of data to be processed. *output* is an SDDS file in which the result is placed. Columns that are not processed are copied from *input* to *output* without change.
- **switches:**
  - `-pipe=[input][,output]` — The standard SDDS Toolkit pipe option.
  - `-columns=`*`listOfNames`* — Specifies an optionally-wildcarded list of names of columns from which to remove baselines.
  - `-select={endPoints=`*`number`*` |`
    `-outsideFWHA=`*`multiplier`*` | -antiOutlier=`*`passes`*`}` — Specifies how to select the points from which to determine the baseline. `endPoints` specifies selecting *number* values from the start and end of the column. `outsideFWHA` specifies selecting all values that are outside *multiplier* times the full-width-at-half-amplitude (FWHA) of the pixel count distribution. —verb—antiOutlier— specifies selecting all values that are *not* deemed outliers in the 2-sigma sense in any of *passes* inspections. These last two options implicitly assume that the statistical distribution of the pixel counts is baseline dominated.
  - `-method={fit | average}` — Specifies how to compute the baseline from the selected points. `fit` specifies fitting a line to the values (as a function of index). `average` specifies taking a simple average of the values.
  - `-nonnegative [-despike=passes=`*`number`*`,widthlimit=`*`value`*`] [-repeats=`*`count`*`]` — Specifies that the resulting function after baseline removal must be nonnegative. Any negative values are set to 0. In addition, despiking (as in `sddssmooth`) may be applied after removal of negative values; this can result in the removal of positive noise spikes. Giving `-repeats` allows applying the baseline removal procedure iteratively to the data.
- **see also:**
  - `sddssmooth` (4.61)

  - `sddscliptails` (4.14)

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

4.10 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` (4.17)

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

4.11 sddscast

- **description:** `sddscast` can be used to change the data type of the elements in an SDDS file.
- **example:** Convert the double columns of `APS.twi` to float (note that col1, col2,col3 are the column names in APS.twi file):

```
sddscast APS.twi -cast=col,*,double,float

sddscast APS.twi -cast=col,'(col1,col2,col3)',double,float

sddscast APS.twi -cast=col,'(col1,col4,col5)','(double,float,float)',long

or:

sddscast APS.twi '-cast=col,(col1,col4,col5),(double,float,float),long'
```

- **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:**
  - `-cast=column|parameter|array,<names>,<typeNames>,<newType>` names is of the form 'name' (with optional wildcards) or '(name,name,....)'; typeNames is of the form '(long,short,double,float,*)' ; newType is long, short,double, or float.
  - `-pipe=[input][,output]` –pipe flages.
  - `-noWarnings` — Suppresses warning messages, such as file replacement warnings.
- **author:**H. Shang ANL/APS.

4.12 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 -changesIn=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
  [-parallelPages]]
  ```

- **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.
  - `-parallelPages` — Valid only with `-baseline`. Specifies that the "baseline" data for each page of the input file shall be taken from the corresponding page of the baseline file. This results in page-by-page subtraction of the two files.

- **see also:**

  - Data for Examples (see 3.3)
  - `sddsenvelope` (4.28)

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

4.13 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` (4.20) 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.
- **switches:**
  - `-printErrors` — Causes the SDDS error traceback to be printed if the file is not `ok`. This may be helpful in determining the problem with the file.
- **see also:**
  - progrefsddsconvert
- **author:** M. Borland, ANL/APS.

4.14 sddscliptails

- **description:** `sddscliptails` removes the tails from functions, where a tail is a dubious feature extending to the left or right of a peak.

- **examples:** Remove tails from profiles of beam spots after baseline removal and prior to determining rms spot properties. This command clips the tails when the function falls to 1% of its peak value.

```
sddscliptails input.sdds output.sdds -columns=VideoLine* -fractional=0.01
```

- **synopsis:**

```
sddscliptails [input] [output] [-pipe=[in][,out]] [-columns=listOfNames]
[-fractional=value] [-absolute=value] [-fwhm=multiplier]
[-afterzero[=bufferWidth]]
```

- **files:** *input* is an SDDS file containing one or more pages of data to be processed. *output* is an SDDS file in which the result is placed. The output file will generally have fewer rows than the input file, corresponding the the number of rows clipped.

- **switches:**
  - `-pipe=[input][,output]` — The standard SDDS Toolkit pipe option.
  - `-columns=`*`listOfNames`* — Specifies an optionally-wildcarded list of names of columns from which to remove tails.
  - `-fractional=`*`value`* — Clip a tail if it falls below this fraction of the peak value of the column.
  - `-absolute=`*`value`* — Clip a tail if it falls below this absolute value. This value might correspond, say, to a known noise level.
  - `-fwhm=`*`multiplier`* — Clip a tail if it is beyond *multiplier* times the full-width-at-half-maximum (FWHM) from the peak of the column.
  - `-afterzero[=`*`bufferWidth`*`]` — Clip a tail if it is separated from the peak by values equal to zero. If *bufferWidth* is specified, then a region *bufferWidth* wide is kept on either side of the peak, if possible.

- **see also:**
  - `sddsbaseline` (4.9)

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

4.15 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*.

- **switches:**
  - `-pipe[=input][,output]` — The standard SDDS Toolkit pipe option.
  - `-noWarnings` — Suppresses warnings about file overwrites.

- **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` (4.50)
  - `sddscombine` (4.17)
  - `sddsexpand` (4.30)

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

4.16 sddscollect

- **description:** `sddscollect` reorganizes tabular data from the input file to bring data from several groups of similarly named columns together into a single column per group. In doing so, it creates one page of output for every row on the input file. It also creates parameters to hold data from columns that are not included in any group.

- **examples:** Take a data set with several columns of PAR BPM x and y readings (one set of readings per row) and create a new file with one column for x readings and one column for y readings, with each page containing one set of readings.

  ```
  sddscollect par.bpm par.orbits -collect=suffix=x -collect=suffix=y
  ```

  The output file has three columns, called `x`, `y`, and `Rootname`. The latter column contains the original column names less the "x" (or "y") suffix.

  Do statistics on PAR BPM x and y readbacks, then collect the statistics into columns, one column for each type of statistic:

  ```
  sddsprocess par.bpm -pipe=out -process=P?P?[xy],spread,%sSpread
  -process=P?P?[xy],ave,%sMean -process=P?P?[xy],stand,%sStDev | sddscollapse
  | sddscollect -pipe=in parbpm.stat  -collect=suffix=xSpread
  -collect=suffix=xMean -collect=suffix=xStDev -collect=suffix=ySpread
  -collect=suffix=yMean -collect=suffix=yStDev
  ```

  The output file has columns named `xSpread`, `ySpread`, `xMean`, `yMean`, `xStDev`, and `yStDev`, plus an additional column named `Rootname`. The latter column contains the remnants of each original column name after the suffix is removed. Note that in the example, the remnant names are the same for all the collections specified. If this were not true, `sddscollect` would abort and give an error message.

- **synopsis:**

  `sddscollect [`*input*`] [`*output*`] [-pipe[=input][,output]]`
  `-collect={suffix | prefix |`
  `match}=`*match*`[,column=`*newName*`][,editCommand=<string>]`

- **switches:**

  - `-pipe[=input][,output]` — The standard SDDS Toolkit pipe option.
  - `-collect={suffix | prefix`
    `| match}=`*match*`[,column=`*newName*`][,editCommand=<string>]` — Specifies a string, *match*, to look for at the end of (suffix mode), beginning of (prefix mode), or anywhere in (match mode) column names in the input file. Data from all matching columns is transferred into a single column in the output file. For prefix and suffix modes, the default name of this column is the suffix or prefix string, while for match mode it is created by applying the edit command to the matching column names. This may be changed with the `column` qualifier.

    This option may be given any number of times. However, all collections must produce the same number of matches. Further the set of name remainders (i.e., the original column name less the prefix or suffix, or following editing) must be the same for each collection.

- **see also:**
  - `sddsregroup` (4.53)
  - `sddstranspose` (4.65)
  - `sddsEditing` (**??**)
- **author:** M. Borland, ANL/APS.

4.17 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] [-sparse=integer] [-collapse]
[-delete={columns | parameters | arrays},matchingString[,...]]
[-retain={columns | parameters | arrays},matchingString[,...]]
```

- **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.
  - `sparse={\em integer}` — Specifies sparsing the tabular data in the input to retain only every *integer*-th row.
  - `collapse` — Specifies collapsing the data, as done by `sddscollapse`.
  - `-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.

- **see also:**
  - Data for Examples (see 3.3)
  - `sddscollapse` (4.15)
- **author:** M. Borland, ANL/APS.

4.18 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 $\sin(4\pi(x2 + y2))$ 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` (4.19)
  - `rpn` (4.70)
- **author:** M. Borland, ANL/APS.

4.19 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.
- **example:** This will generate a two-dimensional color-shaded map of the function $\sin(4\pi(\mathrm{x}^2+\mathrm{y}^2))$ 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`*` |`
    `-columnMatch=`*`indepColumnName,matchingExpression`*`]`

    * `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.
    * `columnMatch` — Specifies plotting of all columns matching *matchingExpression* as a function of the column *indepColumnName*. Each matching column is displayed as a horizontal color bar.

    In the case of the first two choices, 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. The data must be arranged so that y varies fastest as the row in the file increases. Put another way, variable 1 is the row index and variable 2 is the column index.
    2. The numeric parameters `NumberOfRows` and `NumberOfColumns` contain the values of R and C, respectively.
  - `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 *intervalth* contour line, starting with the *offsetth* 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.
- Data scaling:

  [-deltas[={fractional | normalize}]] [-logscale[=*floor*]]

  * deltas — For use with -columnMatch option only. Specifies plotting only differential values (relative to the mean of each column). If the fractional qualifier is given, then the differential values normalized to the individual means are plotted. If the normalize qualifier is given, then all differential values are normalized to the range [-1, 1] before plotting.
  * logscale — Specifies plotting the base-10 logarithm of the values. If a *floor* value is given, it is added to each value prior to taking the logarithm; this can help prevent taking the log of zero, for example.
- 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.
  * 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 (4.18)
  - sddshist2d (4.37)
  - sddsplot (4.48)
  - rpn (4.70)
- **author:** M. Borland, ANL/APS.

4.20 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. N.B.: it is *not* recommended to use `sddsconvert` to convert a binary SDDS file to ASCII, then strip the header off and read the ASCII file. This completely bypasses the self-describing aspects of the SDDS file and is not robust. If the program that creates the SDDS file is changed so that the columns are created in a different order, the program that reads the ASCII file will produce unexpected results. Use `sdds2plaindata` (4.6), `sddsprintout` (4.49), or `sdds2stream` (4.8) for conversion to non-self-describing files. In this way, you can assure the order of the data is fixed.

- **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[=clip]] [-linesPerRow=number] [-nowarnings] [-acceptAllNames]
  ```

- **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` (4.69).
- `-description=`*`text`*`,`*`contents`* — Sets the description fields for the output.
- `-recover[=clip]` — Asks for attempted recovery of corrupted data. If the qualifier is given, then all data from a corrupted page is ignored. Otherwise, `sddsconvert` tries to save as much data from the corrupted page as it can; typically, it is able to save part of the tabular data and all of the array and parameter 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.
- `-acceptAllNames` — Forces acceptance of any name for an SDDS data element (e.g., a column). This can be used with the `rename` or `editNames` options to fix invalid names in SDDS files. This option is provided for backward compatibility to the original version of SDDS, which allowed arbitrary characters in element names.

- **see also:**
  - Data for Examples (see 3.3)
  - `sddsprocess` (4.50)
  - `SDDS editing` (4.69)
- **author:** M. Borland, ANL/APS.

4.21 sddsconvolve

- **description:** `sddsconvolve` performs discrete Fourier convolution/deconvolution/correlation of signals in two files. It assumes that spacing of points is the same in both input files.
- **example:** Compute the result of a signal applied to a system with a known impulse response.

  ```
  sddsconvolve signal.sdds impulseResponse.sdds signalResponse.sdds
  -signalColumns=t,VSignal -responseColumns=t,VImpulse
  -outputColumns=t,VOutput
  ```

- **synopsis:**

  ```
  sddsconvolve signal-file response-file output [-pipe[=in][,out]]
  -signalColumns=indepColumn,dataName -responseColumns=indepColumn,dataName
  -outputColumns=indepColumn,dataName [-deconvolve [-noiseFraction=value] |
  -correlate]
  ```

- **files:** The meaning of the files depends on whether the `-deconvolve` or `-correlate` options are given. If neither option is given, then *signal-file* is the file containing the signal that is imposed on the system, *response-file* is the impulse response of the system, and *output* is the computed response of the system to the signal. If `-deconvolve` is given, then *signal-file* is the response of the system to the signal, *response-file* is the impulse response of the system, and *output* is the computed signal imposed on the system. If `-correlate` is given, then *signal-file* and *response-file* contain two equivalent signals, while *output* contains the computed Fourier correlation; physically, this tells over what time scale the two functions have correlated values.
- **switches:**
  - `-pipe=[input][,output]` — The standard SDDS Toolkit pipe option.
  - `-signalColumns=`*`indepColumn,dataName`* — Specifies the names of the data columns from *signal-file* (the first data file).
  - `-responseColumns=`*`indepColumn,dataName`* — Specifies the names of the data columns from *response-file* (the second data file).
  - `-outputColumns=`*`indepColumn,dataName`* — Specifies the desired names of the result in the file *output*.
  - `-deconvolve` — Specifies deconvolution instead of convolution.
  - `-noiseFraction=`*`value`* — Specifies the amount of noise to allow in the deconvolution to prevent division by zero, as a fraction of the maximum power in the impulse response function.
  - `-correlate` — Specifies correlation instead of convolution.
- **author:** M. Borland, ANL/APS.

4.22 sddscorrelate

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

$$\mathrm{C_{ij}} = \frac{\langle(\mathrm{x_i} - \langle \mathrm{x_i}\rangle)(\mathrm{x_j} - \langle \mathrm{x_j}\rangle)\rangle}{\sqrt{\langle(\mathrm{x_i} - \langle \mathrm{x_i}\rangle)^2\rangle\langle(\mathrm{x_j} - \langle \mathrm{x_j}\rangle)^2\rangle}}$$

  If $\mathrm{C_{ij}} = 1$, then the variables are perfectly correlated, whereas if $\mathrm{C_{ij}} = -1$, 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]
[-rankOrder] [-stDevOutlier[=limit=factor][,passes=integer]]
```

- **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.
  - `-rankOrder` — Specifies computing rank-order correlations rather than standard correlations. This is considered more robust that standard correlations.

- `-stDevOutlier[=limit=`*factor*`][,passes=`*integer*`]` — Specifies standard-deviation-based outlier elimination on each pair of columns prior to computation of the correlation coefficient. Any pair of values is ignored if one or both values are outliers relative to the column from which they come. The `limit` qualifier specifies the allowed deviation from the mean in standard deviations; the default is 1. The `passes` qualifier specifies how many times the outlier elimination (including recomputation of the mean and standard deviation) is performed; the default is 1.

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

4.23 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] [-SavitzkyGolay=left,right,fitOrder[,derivOrder]]
[-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):

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

  - `[-SavitzkyGolay=`*`left,right,fitOrder`*`[,`*`derivOrder`*`]]`
    — Specifies using a Savitzky-Golay smoothing filter to perform the derivative, which involves fitting a polynomial of *fitOrder* through *left*+*right*+1 points and then giving the derivative of the fitted curve. *derivOrder* is 1 by default and gives the order of derivative to take.
  - `-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` (4.38)

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

4.24 sddsderef

- **description:** `sddsderef` allows array and column dereferencing based on constants or on data in parameters and columns.
- **examples:** Let `arrayData` be a file containing a string array named `MessageText` and a column named `MessageIndex` containing integers. The integers are indices into the string array. To create a new column called `Message` giving the message text for each index, the following command would be used:

```
sddsderef arrayData -column=Message,arraySource=MessageText,MessageIndex
```

- **synopsis:**

```
sddsderef [input] [output] [-pipe[=input][,output]]
-column=newName,{columnSource | arraySource}=name,indexName[,indexName1...]
-parameter=newName,{columnSource | arraySource}=name,
     indexName[,indexName1...]
-constant=newName,{columnSource | arraySource}=name,
      indexValue[,indexValue...]
 [-outOfBounds={exit | delete}]
```

- **switches:**
  - `-pipe[=input][,output]` — The standard SDDS Toolkit pipe option.
  - `-column=`*newName*`,{columnSource | arraySource}=`*name*`,`*indexName* `[,`*indexName1*`...]` — Creates a new column named *newName* containing values found by dereferencing the array (or column) *name* using the index or indices in the named columns.
  - `-parameter=`*newName*`,{columnSource | arraySource}=`*name*`,`*indexName* `[,`*indexName1*`...]` — Creates a new parameter named *newName* containing values found by dereferencing the array (or column) *name* using the index or indices in the named parameters. A new value is thus generated for the parameter for each page.
  - `-constant=`*newName*`,{columnSource | arraySource}=`*name*`,`*indexValue* `[,`*indexValue*`...]` — Creates a new parameter named *newName* containing values found by dereferencing the array (or column) *name* using the index or indices given (as explicit values). Unless the array (or column) varies from page to page, the new parameter will have the same value on each page.
  - `-outBounds={exit | delete}` — Specifies behavior in the event that an index value is out of bounds (i.e., less than 0, or greater than or equal to the number of array elements or rows). If `exit` is given, the program aborts with an error; this is the default behavior. If `delete` is given, then the source row or page for the index is omitted from the output.
- **author:** M. Borland, ANL/APS.

4.25 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

4.26 sddsdistest

- **description:** `sddsdistest` performs the Kolmogorov-Smirnov (K-S) test on a set of numbers to determine how likely those numbers are to have been drawn from a specified statistical distribution (e.g., gaussian, poisson).
- **example:** Try the K-S test on random numbers generated by `sddsprocess`

  ```
  sddssequence -pipe=out -define=i,type=long
  -sequence=begin=0,end=9999,delta=1 | sddsprocess -pipe
  -define=column,gaussRN,grnd -define=column,uniformRN,rnd | sddsdistest -pipe
  -test=ks -gaussian -column=gaussRN -column=uniformRN | sddsprintout -pipe
  -column=ColumnName -column=distestSigLevel
  ```

  The result is

  ```
  ColumnName distestSigLevel ----------------------------------- gaussRN
  4.019061e-01 uniformRN 1.598565e-32
  ```

  which shows that the K-S test accurately distinguishes between numbers drawn from the two distributions. The probability that the numbers in column `uniformRN` are from a gaussian distribution is very small, whereas the probability that the numbers in column `gaussRN` are from a gaussian distribution is 40%.
- **synopsis:**

  ```
  sddsdistest [input] [output] [-pipe=[in][,out]] -column=name[,sigma=name]
  ... -exclude=name[,name...] ... -gaussian | -poisson | -student |
  -chisquared [-degreesOfFreedom=value | @parameterName]
  ```

- **switches:**
  - `-pipe=[input][,output]` — The standard SDDS Toolkit pipe option.
  - `-column=`*`name`*`[,sigma=`*`name`*`]` — Specifies the name of a column to test, and optionally the name of the column with the measurement error for the each test value. *name* may contain wildcards. The sigma name may contain "%s", for which each column name is substituted to obtain the corresponding sigma name. Multiple `column` options may be given.
  - `-exclude=`*`name`*`[,`*`name...`*`]` — Specifies the names of columns to exclude from testing.
  - `-gaussian | -poisson | -student | -chisquared` — Specifies the model distribution against which to test the data.
  - `-degreesOfFreedom=`*`value`*` | @`*`parameterName`* — Specifies the number of degrees of freedoms to assume for the model distribution in the case of student and chi-squared distribution. The first form specifies a fixed *value*, whereas the second specifies taking the value for each page from the named parameter.
- **author:** M. Borland, ANL/APS.

4.27 sddsendian

- **description:**

  `sddsendian` converts numerical data in an SDDS file from big-endian to little-endian or vice versa. This is needed prior to transferring binary data from one computer to another if the two computers have different endianess.

- **synopsis:**

  sddsendian [*inputFile*] [*outputFile*] [-pipe[=input][,output]]

- **switches:**

  - `-pipe[=input][,output]` — The standard SDDS Toolkit pipe option.

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

4.28 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] [-median=columnNames]
[-decileRange=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`*, `-median=`*`columnNames`*, `-decileRange=`*`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. Decile range is the spread between the 90% and 10% points on the distribution.
  - `-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` (4.12)

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

4.29 sddseventhist

- **description:** `sddseventhist` analyzes labeled events in a dataset to provide histograms of the occurences of each type of event. It can also histogram the overlap off all types of events with a single type of event.

- **synopsis:**

  ```
  sddseventhist [-pipe=[input][,output]] [inputFile] [outputFile]
  -dataColumn=columnName -eventIdentifier=columnName
  [-overlapEvent=eventValue] [-bins=number | -sizeOfBins=value]
  [-lowerLimit=value] [-upperLimit=value] [-sides] [-normalize[={sum | area |
  peak}]]
  ```

- **files:** *inputFile* is a file containing at least two columns of data. One column must contain string entries that serve as "event identifiers"; for example, these might be the names of channels that issued an alarm. The other column must contain numerical data that will be histogrammed; for example, these might be the times at which alarms occured. The *outputFile* contains one histogram of this numerical data for each unique value in of the event identifier; the histogram contains only the data that matches that identifier.

- **switches:**
  - `-pipe[=input][,output]` — The standard SDDS Toolkit pipe option.
  - `-dataColumn=`*columnName* — Specifies the name of the data column to be histogrammed.
  - `-eventIdentifier=`*columnName* — Specifies the name of the string column that identifies events.
  - `-overlapEvent=`*eventValue* — Requests computation of the overlap of the histograms of each event with the histogram of event *eventValue*. Useful in determining which events always occur at the same time as event *eventValue*.
  - `-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.
  - `-sides` — Specifies that zero-height bins should be attached to the lower and upper ends of the eventhistogram. 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.

- **see also:**
  - `sddscorrelate` (4.22)

  - sddshist (4.36)
  - sddshist2d (4.37)

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

4.30 sddsexpand

- **description:**

  `sddsexpand` reads data pages from an SDDS file and writes a new SDDS file containing a separate data page for every row in the input file. All column definitions from the input file are turned into parameter definitions in the output file. In addition all parameter definitions from the input file are copied to the output file. Each output data page contains the values of the columns from a single row of the input file, along with the values of the parameters from the same page. The output file contains no column or array definitions.

  `sddsexpand` is essentially the inverse of `sddscollapse` (except that the column data thrown out in collapsing a file is not recoverable).

- **synopsis:**

  ```
  sddsexpand [inputFile] [outputFile] [-pipe[=input][,output]]
  ```

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

- **switches:**
  - `-pipe[=input][,output]` — The standard SDDS Toolkit pipe option.
  - `-noWarnings` — Suppresses warnings about name clashes between parameters and columns. If such a clash occurs, the parameter data is ignored.

- **see also:**
  - `sddscollapse` (4.15)
  - `sddsbreak` (4.10)

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

4.31 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

  $$E(x) = C + F * eR * x,$$

  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` (4.47)
  - `sddsgfit` (4.35)
  - `sddsoutlier` (4.45)
- **author:** M. Borland, ANL/APS.

4.32 sddsfdfilter

- **description:**
- **synopsis:**

```
sddsfdfilter [-pipe[=input][,output]] [inputfile] [outputfile]
[-columns=indep-variable[,depen-quantity][,depen-quantity...]]
[-exclude=depen-quantity[,depen-quantity]]
[-clipFrequencies=[high=number][,low=number]]
[-threshold=level=value[,fractional][,start=freq][,end=freq]]
[-highpass=start=freq,end=freq] [-lowpass=start=freq,end=freq]
[-notch=center=center,flatWidth=width1,fullWidth=width2]
[-bandpass=center=center,flatWidth=width1,fullWidth=width2]
[-filterFile=filename=filename,frequency=columnName,
{real=columnName,imaginary=columnName | magnitude=columnName}]
[-cascade] [-newColumns] [-differenceColumns]
```

- **switches:**
  - `-pipe[=input][,output]` — The standard SDDS Toolkit pipe option.
  - `-columns=`*indepVariable*`[,`*depenQuantity*`][,`*depenQuantity*`...]` — Gives the name of the independent variable (typically the time variable) with respect to which filtering is done. If no *depenQuantity* qualifiers are given, then all numerical columns are filtered; otherwise, only the named columns are filtered.
  - `-exclude=`*depenQuantity*`[,`*depenQuantity*`]` — Specifies the names of columns to exclude from time filtering, altering whatever selections are made by the `-columns` option.
  - `-clipFrequencies[=low=`*frequency*`][,high=`*frequency*`]` — Specifies clipping frequencies below a given frequency (`low` qualifier) and/or above a given frequency (`high` qualifier). Any frequencies in the signals that are clipped are set to zero.
  - `-threshold=level=`*value*`[,fractional][,start=`*freq*`][,end=`*freq*`]` — Specifies a threshold level below which Fourier components are set to zero. If the `fractional` qualifier is given, the level is interpreted as a fraction of the largest component. The `start` and `end` qualifiers allow restricting the frequency range over which the threshold is applied.
  - `-highpass=start=`*freq*`,end=`*freq* — Specifies a highpass filter. Frequency components below the `start` value are set to zero, while those above the `end` value are unaffected. Those in between are multiplied by a value that varies linearly from 0 to 1.
  - `-lowpass=start=`*freq*`,end=`*freq* — Specifies a lowpass filter. Frequency components above the `end` value are set to zero, while those below the `start` value are unaffected. Those in between are multiplied by a value that varies linearly from 0 to 1.
  - `-notch=center=`*center*`,flatWidth=`*width1*`,fullWidth=`*width2* — Specifies a notch filter centered on a given frequency, attenuating completely within a band *width1* wide. Frequencies outside a band *width2* are unattenuated. Frequencies between the two widths are attenuated by values varying linearing from 0 to 1 as the frequency becomes more distant from the center frequency.

- `-bandpass=center=`*center*`,flatWidth=`*width1*`,fullWidth=`*width2* — Specifies a bandpass filter centered on a given frequency, passing a band *width1* wide without attenuation. Frequencies outside a band *width2* are completely attenuated. Frequencies between the two widths are attenuated by values varying linearing from 1 to 0 as the frequency becomes more distant from the center frequency.
- `-filterFile=filename=`*filename*`,frequency=`*columnName*`,`
  {`real=`*columnName*`,imaginary=`*columnName* `|` `magnitude=`*columnName*} — Specifies a filter explicitly as a function of frequency, using either a single value (simple attenuation) or a real and imaginary value. The name of the column containing frequency values must be given with the `frequency` qualifier. In addition, one must either give the name of the column containing the `magnitude` (attenuation factor) or else both the names of the columns containing the `real` and `imaginary` components. The function thus specified is interpolated linearly to obtain values at any required frequencies. Fourier components at frequency beyond the range in the file are unaffected.
- `-cascade` — By default, if several filters are specified using the above options, their output is added. When the `cascade` option is given, a new sequence is started, with the original signal as input. The output from all cascades is summed to obtain the final result.
- `-newColumns` — Specifies that new columns be created in the output file to hold the result of the filtering. By default, the filtered data is placed in columns with the same names as those in the input file. Using this option, both the filtered and unfiltered data will appear in the output file. The filtered data will have names of the form *columnName*`Filtered`, where *columnName* is the name of the source column in the input file.
- `-differenceColumns` — Specifies that new columns be created in the output file that contain the difference between each original column and the corresponding filtered column. The new columns have names of the form *columnName*`Difference`, where *columnName* is the name of the source column in the input file.

• **see also:**

  - `sddsdigfilter` (4.25)
  - `sddssmooth` (4.61)

• **author:** M. Borland, ANL/APS.

4.33 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] [-psdOutput]
```

- **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*. If power-spectral-density output is requested, then a column `PSD`*indepName* is also created.

  outputFile also contains two parameters, `fftFrequencies` and `fftFrequencySpacing`, giving the number of frequencies and the frequency spacing, respectively.

- **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.
  - `-exclude=`*depenQuantity*`,...` — Specifies optionally wildcarded names of columns to exclude from analysis.
  - `-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 product of primes from 2 to 19 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.
- psdOutput — Specifies that in in addition to ordinary FFT data, the power-spectral-densities will also be included in the output. The units of the PSD are of the form x^2/t, where x (t) represents the units of the independent (dependent) variable. These units are conventional with PSDs, which are normalized to the frequency spacing so that integrating the PSD gives the signal power.

- **see also:**
  - Data for Examples (see 3.3)
  - sddsdigfilter (4.25)
  - sddsnaff (4.43)
- **author:** M. Borland, ANL/APS.

4.34 sddsgenericfit

- **description:** `sddsgenericfit` does fits to an arbitrary functional form specified by the user.
- **examples:** Fit a gaussian to a beam profile to get the rms beam size. In this example, the file `beamProfile.sdds` is assumed to contain data to be fit in columns `x` and `Intensity`.

```
sddsgenericfit beamProfile.sdds beamProfile.gfit -column=x,Intensity
 ‘‘-equation=height x center - sqr 2 / sigma sqr / exp / baseline + y -
sqr’’
 -variable=name=height,start=1,lower=0,upper=10,step=0.1
 -variable=name=center,start=0,lower=-10,upper=10,step=0.1
 -variable=name=sigma,start=1,lower=0.1,upper=10,step=0.1
 -variable=name=baseline,start=0,lower=-1,upper=1,step=0.1
```

- **synopsis:**

```
sddsgenericfit [-pipe=[input][,output]] [inputfile] [outputfile]
-columns=x-name,y-name[,ySigma=sy-name] -equation=string[,algebraic]
[-target=value] [-tolerance=value]
[-simplex=[restarts=integer][,cycles=integer,][evaluations=integer]]
-variable=name=name,lowerLimit=value,upperLimit=value,stepsize=value,startingValue=value[
[-variable=...]  [-verbosity=integer] [-startFromPrevious]
```

- **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 column is constructed by appending the string `Fit` to the name of the dependent variable. The name of the residual column is constructed by appending the string `Residual` to the name of the dependent variable. *outputFile* also contains parameters giving the values of the fit parameters.
- **switches:**
  - `-pipe=[input][,output]` — The standard SDDS Toolkit pipe option.
  - `-columns=`*x-name*`,`*y-name*`[,ySigma=`*name*`]` — Specifies the names of the independent and dependent columns of data, and optionally the name of the column containing the errors in the dependent column.
  - `-equation=`*string*`[,algebraic]` — Specifies the penalty function for performing the fit. If you are fitting the form $y = f(x)$, then the penalty function has the form $(y - f(x))^2$.
  - `-tolerance=`*value* — Specifies the minimum change in the (weighted) rms residual that is considered significant enough to justify continuing optimization.
  - `-simplex=[restarts=`*nRestarts*`][,cycles=`*nCycles*`,][evaluations=`*nEvals*`]` — Specifies parameters of the simplex optimization used to perform the fit. Each start or restart allows *nCycles* cycles with up to *nEvals* evaluations of the function. Defaults are 10 restarts, 10 cycles, and 5000 evaluations.

– -variablename=*name*,lowerLimit=*value*,upperLimit=*value*,stepsize=*value*,startingValue=*val*
— Specifies a parameter of the fitting fnuction. If the heat qualifier is given, then prior to each restart the code "heats" the values by adding random numbers to the result of the last iteration. This can help avoid getting stuck in a local minimum. You must give one variable option for every parameter of the fit.

– -startFromPrevious — Meaningful for multipage input files only. If given, then the optimization for each page starts from the parameter values from the fit to the previous page. By default, fitting for each page starts from the values specified on the commandline.

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

- **see also:**
  - sddsexpfit (4.31)
  - sddsgfit (4.35)
  - sddsoutlier (4.45)
  - sddspfit (4.47)
- **author:** M. Borland, ANL/APS.

4.35 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]]
[-fixValue=[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 one or more parameters.
  - `-fixValue=[baseline=`*value*`][,mean=`*value*`][,height=`*value*`][,sigma=`*value*`]` — Gives `sddsgfit` a fixed value for one or more parameters. If given, then `sddsgfit` will not attempt to fit the parameters in question.
  - `-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` (4.47)
  - `sddsexpfit` (4.31)
  - `sddsoutlier` (4.45)
- **author:** M. Borland, ANL/APS.

4.36 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 4.50) 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` (4.37)
  - `sddsprocess` (4.50)

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

4.37 sddshist2d

- **description:**

  `sddshist2d` makes two-dimensional histograms of data, producing output that is suitable for plotting with `sddscontour`. The two-dimensional histogram may include data from two columns, or may show the histograms of a single column versus page number.

- **examples:** Make a two-dimensional histogram of two PAR bpm readouts, then plot the result:

```
sddshist2d par.bpm par.bpm.h2d -column=P1P1x,P1P2x -xparam=50 -yparam=50
sddscontour -shade=32 par.bpm.h2d -quantity=frequency
```

- **synopsis:**

```
sddshist2d [-pipe[=input][,output]] [inputfile] [outputfile]
-columns={xName,yName | yName} [-weights=columnName[,average]]
[-xParameters=bins[,lower,upper]] [-yParameters=bins[,lower,upper]]
[-outputName=string] [-sameScale] [-combine] [-normalize[=sum]]
[-smooth[=passes]] [-verbose]
```

- **switches:**

  – `-pipe[=input][,output]` — The standard SDDS Toolkit pipe option.

  – `-columns={`*`xName,yName | yName`*`}` — Specifies the data from the input to histogram. If both *xName* and *yName* are given, then `sddshist2d` does a two-dimensional histogram of the values in the named columns. If only `yName` is given, then `sddshist2d` does a series of one-dimensional histograms of the named column, one for each data pages; these histograms are then assembled as a two-dimensional histogram with one axis being the page number.

  – `-weights=`*`columnName`*`[,average]` — Specifies the name of a column of data with which to proportionally weight the count value of points in the histogram. If the `average` qualifier is given, then each bin value is normalized to contain the average value of the weight for all points in the bin.

  – `-xParameters=`*`bins`*`[,`*`lower,upper`*`]` — Specifies the number of bins and optionally the histogrammed region for the x values. Ignored if only *yName* is given. By default, 21 bins are used encompassing all of the data points.

  – `-yParameters=`*`bins`*`[,`*`lower,upper`*`]` — Specifies the number of bins and optionally the histogrammed region for the y values. By default, 21 bins are used encompassing all of the data points.

  – `-outputName=`*`string`* — Specifies the name of the histogram data. The default is `frequency`.

  – `-sameScale` — Specifies that for multipage input files, the histogram region should be the same for all pages. The region is set to encompass all data points from all pages.

  – `-combine` — Specifies that for multipage input files, the data from all pages should be placed in a single histogram.

- `-normalize[=sum]` — Specifies normalization of the histogram. If the `sum` qualifier is not given, the histogram is normalized to unit amplitude; otherwise, it is normalized so that the sum of all frequencies is unity.
- `-smooth[=`*`passes`*`]` — Specifies smoothing by nearest-neighbor-averaging. If *passes* is omitted, only one pass is performed.
- `-verbose` — Requests informational output during processing.

- **see also:**
  - `sddshist` (4.36)
  - `sddscontour` (4.19)
  - `sddscongen` (4.18)
- **author:** M. Borland, ANL/APS.

4.38 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` (4.23)
- **author:** M. Borland, ANL/APS.

4.39 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 |
 -fillIn |
 -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.
  - `-fillIn`—Somewhat like `-sequence=`*`points`*, except the number of points is chosen so that the spacing of the interpolation points is equal to the minimum point spacing in the file. In other words, if you have a data file with non- equidistant points, this option interpolates to give you equidistant points with the same minimum spacing as the original data. This tends to fill in the space between widely-spaced points, hence the name.

- -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 4.47)
  - Data for Examples (see 3.3)

• **author:** M. Borland, ANL/APS.

4.40 sddsmpfit

- **description:** `sddsmpfit` 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. The options for `sddsmpfit` are very similar to those for `sddspfit`.

- **synopsis:**

```
sddsmpfit [-pipe=[input][,output]] [inputFile] [outputFile]
-independent=xName [-sigmaIndependent=xSigmaName]
-dependent=yName[,yName...]  [-sigmaDependent=templateString] -terms=number
[-symmetry={none | odd | even}] | -orders=number[,number...]
[-reviseOrders[=threshold=chiValue][,verbose]] [-chebyshev[=convert]]
[-xOffset=value] [-xFactor=value] [-sigmas=value,{absolute | fractional}]
[-modifySigmas] [-generateSigmas[=keepLargest | keepSmallest]]
[-sparse=interval] [-range=lower,upper] [-normalize[=termNumber]] [-verbose]
[-evaluate=filename[,begin=value][,end=value][,number=integer]]
[-fitLabelFormat=sprintfString] [-infoFile=filename]
```

- **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`. In addition, various parameters having names beginning with *yName* are created that give reduced chi-squared, slope, intercept, and so on.

- **switches:**
  - `-pipe[=input][,output]` — The standard SDDS Toolkit pipe option.
  - `-evaluate=`*`filename`*`[,begin=`*`value`*`][,end=`*`value`*`][,number=`*`integer`*`]` — Specifies creation of an SDDS file called *filename* containing points from evaluation of the fit. The fit is normally evaluated over the range of the input data; this may be changed using the `begin` and `end` qualifiers. Normally, the number of points at which the fit is evaluated is the number of points in the input data; this may be changed using the `number` qualifier.
  - `infoFile=`*`filename`* — Specifies creation of an SDDS file containing results of the fits in columns. A column called *yName*Coefficient is created for each column that is fitted.
  - 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]` — Asks for adaptive fitting to be performed on the first data page to determine what orders to use. Any term that does not improve the reduced chi-squared by *value* is discarded. Similar to but much less capable than the adaptive fitting feature of `sddspfit`.
  * `[-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 → (x − Offset)/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.
- `sddsmpfit` 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=`*`value`*{`absolute | fractional`} — 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 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 `sddsmpfit` 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)
  - `sddspfit` (4.47)
  - `sddsoutlier` (4.45)
- **author:** M. Borland, ANL/APS.

4.41 sddsmultihist

- **description:** `sddsmultihist` does one-dimensional histograms of multiple columns of data from an SDDS file. All columns are histogrammed on the same interval and with the same number of bins. It is similar to `sddshist`, except that the latter program only histograms a single column at a time. Unlike `sddshist`, `sddsmultihist` does not presently do statistical analyses or filtering.

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

```
sddshist par.bpm par.bpmhis -column=P?P?x -bins=20 -abscissa=xReadout
```

- **synopsis:**

```
sddsmultihist [-pipe=[input][,output]] [inputFile] [outputFile]
-columns=columnName[,columnName...]  -abscissa=newName [-separate]
[-exclude=columnName[,columnName...]]  [-bins=number | -sizeOfBins=value]
[-lowerLimit=value] [-upperLimit=value] [-sides]
```

- **files:** *inputFile* is the name of an SDDS file containing data to be histogrammed. If *inputFile* contains multiple data pages, each is treated separately. The histograms are placed in *outputFile*, which has one column of histogram frequencies for each histogrammed input column, plus a column giving the abscissa values for the frequency distributions. The former columns have names of the form *columnName*`Frequency`, containing the number of points in each bin. The latter column has a name given by the user.

- **switches:**
  - `-pipe[=input][,output]` — The standard SDDS Toolkit pipe option.
  - `-columns=`*columnName*`[,`*columnName*`...]` — Specifies the names of the data columns to be histogrammed. The *columnName* items may contain wildcards.
  - `-separate` — Specifies that a separate abscissa shall be created for each histogrammed column. If `-abscissa` is not given, then the abscissa names are the names of the columns being histogrammed.
  - `-abscissa=`*newName*`[,`*newName*`...]` — Specifies the name or names of the abscissa columns for the histogram output. If `-separate` is not given, then only one name is permitted. The units taken from the units of the columns being histogrammed.
  - `-exclude=`*columnName*`[,`*columnName*`...]` — Specifies the names of data columns to exclude from histogramming. The *columnName* items may contain wildcards.
  - `-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.
  - `-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.

- **see also:**
  - Data for Examples (see 3.3)
  - `sddshist` (4.36)
  - `sddshist2d` (4.37)
- **author:** M. Borland, ANL/APS.

4.42 sddsmatrixmult

- **description:** `sddsmatrixmult` multiplies the matrices represented in the two input files and puts the results in the output file.

  String columns are ignored and not copied to the output file.

- **examples:** In an accelerator beamline a linear relationship exists between the corrector dipole setpoints and the beam position monitor (BPM) readbacks. The matrix data in file response is multiplied with the columns of file corrector to produce a new file containing values of expected bpm change:

  ```
  sddsmatrixmult response correctorChange bpmExpectedChange
  ```

- **synopsis:**

  ```
  sddsmatrixmult [-pipe=[input][,output]] [file1] file2 [output] [-commute]
  [-reuse] [-verbose] [-ascii]
  ```

- **files:** The first file (*file1*) is the SDDS file for left-hand matrix of product. The second file (*file2*) is the SDDS file for right-hand matrix of product. The third file contains the product matrix data.

- **switches:**
  - `-pipe[=input][,output]` — The standard SDDS Toolkit pipe option.
  - `-commute` — Use *file1* for right-hand matrix and *file2* for left-hand matrix. Useful with -pipe option
  - `-reuse` — If one file runs out of data pages, then reuse the last one.
  - `-ascii` — Produces an output in ascii mode. Default is binary.
  - `-verbose` — Write diagnostic messages to stderr.

- **author: L. Emery** ANL

4.43 sddsnaff

- **description:** `sddsnaff` is an implementation of Laskar's Numerical Analysis of Fundamental Frequencies (NAFF) algorithm. This algorithm provides a way of determining the frequency components of a signal that is more accurate than Fast Fourier Transforms (FFT). FFTs are used as part of the analysis, so if an FFT is sufficient for an application, `sddsfft` should be used as it will be much faster.

  The algorithm starts by removing the average value of the signal and applying a Hanning window. Next, the signal is FFT'd and the frequency at which the maximum FFT amplitude occurs is found. This is taken as the starting frequency for a numerical optimization of the "overlap" between the signal and $e^{i\omega t}$, which allows determining ω to resolution greater than the frequency spacing of the FFT. Once ω is determined, the overlap is subtracted from the original signal and the process is repeated, if desired.

- **examples:** Find the first fundamental frequency for each of the BPM signals in `par.bpm`.

```
sddsnaff par.bpm par.naff -column=Time,'P?P?x'
-terminateSearch=frequencies=1
```

- **synopsis:**

```
sddsnaff [inputfile] [outputfile] [-pipe=[input][,output]]
[-columns=indep-variable[,depen-quantity[,...]]]
[-exclude=depen-quantity[,...]]
[-terminateSearch=changeLimit=fraction[,maxFrequencies=number] |
frequencies=number]
[-iterateFrequency=[cycleLimit=number][,accuracyLimit=fraction]] [-truncate]
[-noWarnings]
```

- **files:**

  inputFile contains the data to be NAFF'd. One column from this file must be chosen as the independent variable. If *inputFile* contains multiple pages, each is treated separately and is delivered to a separate page of *outputFile*.

  outputFile contains two columns for each selected column in *inputFile*. These columns have names like *origColumn*`Frequency` and *origColumn*`Amplitude`, giving the frequency and amplitude for *origColumn*.

- **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 NAFF'd as a function of the independent variable. By default, all numerical columns except the independent column are NAFF'd.
  - `-exclude=`*`depenQuantity`*`,...` — Specifies optionally wildcarded names of columns to exclude from analysis.

- `-terminateSearch={changeLimit=`*fraction*`[maxFrequencies=`*number*`]`
  `| frequencies=`*number*`}` — Specifies when to stop searching for frequency components. If `changeLimit` is given, then the program stops when the RMS change in the signal is less than the specified *fraction* of the original RMS value of the signal. The maximum number of frequencies that will be returned in this mode is specified with `maxFrequencies` (default is 4). If `frequencies` is given, then the program finds the given number of frequencies, if possible. By default, the program finds one frequency for each signal.
- `-iterateFrequency=[cycleLimit=`*number*`][,accuracyLimit=`*fraction*`]` — This option controls the optimization procedure that searches for the best frequency. By default, the procedure executes 100 passes and attempts to determine the frequency to a precision of 0.00001 of the Nyquist frequency. `cycleLimit` is used to change the number of passes, while `accuracyLimit` is used to specify the desired precision.
- `-truncate` — Specifies that the data should be truncated so that the number of points is the largest product of primes from 2 to 19 not greater than the original number of points. In some cases, this will result in significantly greater speed, by making the FFTs faster.
- `-noWarnings` — Suppresses warning messages.

• **see also:**
  - Data for Examples (see 3.3)
  - `sddsfft` (4.33)

• **author:** M. Borland, ANL/APS.

4.44 sddsnormalize

- **description:** `sddsnormalize` performs various normalizations of column data.
- **synopsis:**

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

  `-columns=[mode=`*mode*`][,suffix=`*string*`][,exclude=`*wildcardString*`],columnName[,columnName...]`

  where mode is one of `minimum`, `maximum`, `largest`, `signedLargest`, or `spread`, referring to the factor used for normalization (see below).
- **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:**
  - `-pipe=[input][,output]` — The standard SDDS Toolkit pipe option.
  - `-columns=[mode=`*mode*`][,suffix=`*string*`]`
    `[,exclude=`*wildcardString*`],columnName[,columnName...]` — Any number of these options may be given. Each specifies columns to normalize and in what *mode*. *mode* may be one of `minimum`, `maximum`, `largest`, `signedLargest`, or `spread`, referring to the factor used for normalization: largest (the default) is the maximum absolute value; signed largest is the same value, but with the sign restored; spread is the maximum minus the minimum. Each *columnName* qualifier gives a possibly wildcarded string specifying columns to normalize. `exclude` may be used to exclude columns from normalization that are matched by a *columnName*. The `suffix` qualifier optionally specifies a suffix to be appended to each column name, to create a new column for the output file; if not given, then the original data are replaced with the normalized data.
- **author:** M. Borland, ANL/APS.

4.45 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 three standard deviations from the mean:

  ```
  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] [-minimumLimit=value]
  [-maximumLImit=value] [-chanceLimit=value] [-invert] [-verbose]
  [-noWarnings] [{-markOnly | -replaceOnly={lastValue | nextValue |
  interpolatedValue | value=number}}]
  ```

- **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.
- `-minimumLimit=`*value*, `-minimumLimit=`*value* — Specify minimum or maximum values that data points may have without being considered outliers.
- `-chanceLimit=`*value* — Specifies placing a lower limit on the probability of seeing a data point as a means of removing outliers. Gaussian statistics are used to determine the probability that each point would be seen in sampling a gaussian distribution a given number of times (equal to the number of points in each page). If this probability is less than *value*, then the point is considered an outlier. Using a larger *value* results in elimination of more points.
- `-invert` — Specifies that only outlier points should be kept.
- `-markOnly` — Specifies that instead of deleting outlier points, they should be only marked as outliers. This is done by creating a new column (`IsOutlier`) in the output file that contains a 1 (0) if the row has (no) outliers. If `IsOutlier` is in the input file, rows with a value of 1 are treated as outliers and essentially ignored in processing. Hence, successive invocations of `sddsoutlier` in a data-processing pipeline make use of results from previous invocations even if `-markOnly` is given. Note: if `-markOnly` is not given, then the presence of `IsOutlier` in the input file has no effect.
- -tt -replaceOnly={lastValue — nextValue — interpolatedValue — value=*number*} — Specifies replacing outliers rather than removing them. `lastValue` (`nextValue`) specifies replacing with the previous (next) value in the column. `interpolatedValue` specifies interpolating a new value from the last and next value (with row number as the independent quantity). `value=`*number* specifies replacing outliers with *number*.
- `-verbose` — Specifies that informational printouts should be provided.
- `-noWarnings` — Specifies that warnings should be suppressed.

- **see also:**
  - Data for Examples (see 3.3)
  - `sddspfit` (4.47)
  - `sddsgfit` (4.35)
  - `sddsexpfit` (4.31)
  - `sddscorrelate` (4.22)
- **author:** M. Borland, ANL/APS.

4.46 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` (4.33)
  - `sddssmooth` (4.61)
  - `sddspeakfind` (4.46)

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

4.47 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]
[-evaluate=filename[,begin=value][,end=value][,number=integer]]
-columns=xName,yName[,xSigma=name][,ySigma=name] -terms=number
[-symmetry={none | odd | even}] | -orders=number[,number...]
[-reviseOrders[=threshold=chiValue][,verbose][,complete=<chiThreshold>][,goodEnough=<chiVa
[-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 *quantityName*`Sigma`.

- **switches:**
  - `-pipe[=input][,output]` — The standard SDDS Toolkit pipe option.
  - `-evaluate=`*`filename`*`[,begin=`*`value`*`][,end=`*`value`*`][,number=`*`integer`*`]` — Specifies creation of an SDDS file called *filename* containing points from evaluation of the fit. The fit is normally evaluated over the range of the input data; this may be changed using the `begin` and `end` qualifiers. Normally, the number of points at which the fit is evaluated is the number of points in the input data; this may be changed using the `number` qualifier.
  - `-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=`*`chiValue1`*`][,verbose]`
      `[,complete=`*`chiThreshold`*`][,goodEnough=`*`chiValue2`*`]` — Specifies adaptive fitting to eliminate spurious terms. When invoked, this switch causes `sddspfit` to repeatedly fit the first page of data with different numbers of terms in an attempt to find a minimal number of terms that gives an acceptable fit. This is done in up to three stages:
      1. The process starts by making a fit with all terms. Then, each term is eliminated individually and a new fit is made. If the new fit has a smaller reduced chi-squared by an amount of at least *chiValue1*, then the term is permanently eliminated and the process is repeated for each remaining term. By default, the criterion for an improvement is a change of 0.1 in the reduced chi-squared. This step eliminates terms that result in a bad fit due to numerical problems. If the `goodEnough=`*`chiValue2`* qualifier is given, then the first fit that has reduced chi-squared less than *chiValue2* is used.
      2. Next, the individual terms are tested for how well they improve reduced chi-squared. Any term that does not improve the reduced chi-squared by at least *chiValue1* is eliminated. This stage eliminates terms that do not sufficiently improve the fit to merit inclusion. Again, if the `goodEnough=`*`chiValue2`* qualifier is given, then the first fit that has reduced chi-squared less than *chiValue2* is used.

3. Finally, if `complete=`*`chiThreshold`* is given, then next stage involves repeating the above procedure with the remaining terms, but instead of eliminating one term at a time, the program tests each possible combination of terms. This can be very time consuming, especially if the `goodEnough=`*`chiValue2`* qualifier is not given.

  * `[-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 → (x − Offset)/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` (4.31)
  - `sddsgfit` (4.35)
  - `sddsplot` (4.48)
  - `sddsoutlier` (4.45)

• **author:** M. Borland, ANL/APS.

4.48 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, using different line types for each quantity:

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

  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
  ```

  Plot the Twiss functions for the APS design, using a common scale for the beta functions and another for eta:

  ```
  sddsplot -graphic=line,vary APS0.twi -columnNames=s,beta?
  -yScalesGroup=id=beta -columnNames=s,etax -yScalesGroup=id=eta
  ```

- **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 rapidly, 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 appear.

Any point on any plot panel can be referenced by unit coordinates 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. By default, the region the pspace occupies in these coordinates is [0.15, 0.90]x[0.15, 0.90]. The extent of the pspace may be changed explicitly, or it may be altered implicitly by certain switches (e.g., to make room for legends). The data or user's coordinates, referred to as (x, y), are mapped onto this space, as are the "pspace coordinates", (p, q). The latter are [0, 1]x[0, 1] coordinates.

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}]` — Rarely used. 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]` — Provided for compatibility with an older type of data file and rarely used. 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.

    * `-namescan={all | first}` — Specifies whether `sddsplot` should scan all input files when searching for matches to wildcard datanames, or only the first. The default is to scan all files, which may be slow for many files with large numbers of columns or parameters.

- 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. Some commonly-used devices that have device-specific arguments are:
        · `motif` — The device arguments are a single string of space-separated entries of the form `-`*resourceName value*. These are passed directly to the MOTIF "outboard-driver" without any interpretation. The resource names may be found in the help for the driver.
        · `mif` — Three qualifiers are presently accepted. `linesizeDefault=`*size* sets the default line thickness (normally 0.25). `dashsizeDefault=`*size* sets the default dash size (normally 1.0). `lineIncrement=`*value* sets the line thickness increment between different line types.
        · `gif`, `tgif`, `sgif`, `mgif`, `lgif` — Two qualifiers are presently accepted, both dealing with output file setup. `rootname=`*string* specifies a rootname for automatic filename generation; the resulting filenames are of the form *rootname*.DDD, where DDD is a three-digit integer. `template=`*string* provides a more general facility; one uses it to specify an sprintf-style format string to use in creating filenames. For example, the behavior obtained using `rootname=`*name* may be obtained using `template=`*name*`.%03ld`.
    * `-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 | fixByName}]` — Specifies the type of graphic element to use for data in the present plot request.
    element may be one of `line`, `symbol`, `errorBar`, `impulse`, `yimpulse`, `bar`, `ybar`, `dot`, or `continue`. 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. `yimpulse` and `ybar` are analogous except that the line extends from x=0 or from the left-hand vertical border of the plot.
    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, error bars, and dots. It specifies the line style or color to be used in making a symbol or error bar, and the size for a dot. 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.

If one desires automatic variation of the line color, symbol type, and so on, one may obtain this using the `vary` qualifier. By default, the type is varied. The `eachFile`, `eachPage`, `eachRequest`, or `fixByName` may be given to specify how to assign type or subtype. For `eachFile`, variation is done separately for data from different files. For `eachPage`, variation is done separately for data from different pages (hence, items from different pages would have the same line or symbol). For `eachRequest`, variation is done separately for each request. The `fixByName` qualifier in constrast assigns fixed graphic attributes to items according to the y name.

  * `-arrowSettings=[,autoScale][scale=`*`factor`*`][,linetype=`*`integer`*`]`
`[,centered][,singleBarb][,barbLength=`*`value`*`][,barbAngle=`*`value`*`]`
`[,{cartesianData | polarData | scalarData}]` —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,xmax,ymin,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.

  * `-range=[{x|y}Minimum=`*`value`*`][,{x|y}Maximum=`*`value`*`][,{x|y}Center=`*`value`*`]`

— Constrains the extent and center of the plot in user's coordinates. `xminimum` specifies the minimum allowable horizontal extent of the plot; if the autoscaled (or user-specified) range is less than this, the range is increased symmetrically to this value. Similarly, `xmaximum` specifies the maximum allowable horizontal extent of the plot. `xcenter` specifies the center of the horizontal range without affecting the extent. The `y` options are the same, but for the vertical coordinates.

* `-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][,global]`—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 from a request, rather than simply the data on a particular plot panel. If given without these qualifiers, both x and y are affected. `global` forces `sddsplot` to impose the desired condition across all plot requests.
* `-zoom=[{x|y}Factor=`*value*`][,{x|p}Center=`*value*`][,{y|q}Center=`*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=[,[{x|y}]grid][[{x|y}]spacing=`*value*`]`
`[,[{x|y}]factor=`*value*`][,[{x|y}]modulus=`*value*`]`
`[,[{x|y}]size=`*fraction*`][,[{x|y}]linetype=`*integer*`]`
`[,[{x|y}]logarithmic]` — 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=[[{x|y}]divisions=`*`integer`*`][,[{x|y}]grid]`
`[,[{x|y}]linetype=`*`integer`*`][,[{x|y}]size=`*`fraction`*`]`—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.
* `-yScalesGroup={ID=`*`string`*` | fileIndex | nameIndex | nameString |`
`page | request | tag | subpage | iNameString}` — Specifies multiple vertical scales. The most common form is `|-yscalesGroup=namestring|, which uses a separate scale for every separately-named quantity. Otherwise, one specifies a separate scale for items from different files (by file index), with different name index, different page, and so on. The tag is a quality of a dataset specified with the -tag option. iNameString is the name string in inverse order (i.e., so that one compares namestrings starting at the end rather than the beginning). These qualifiers are shared with the groupBy and separate options.`
* `-xScalesGroup --- Identical to yScalesGroup but for x axis scales.`
*
`alignZero[={xcenter|xfactor|pPin=`*`value`*`}][,{ycenter|yfactor|qPin=`*`value`*`}]`
`--- This option is provides a facility for lining up zeros on plots with multiple axes. You must give at least one of the qualifiers. The xfactor and yfactor qualifiers request multiplication of the upper and lower limits for each scale by the smallest factors that will line up the zeros. The xcenter and ycenter qualifiers position the zeros at the center of the plot space, which may result in empty regions on the plot. The pPin and qPin allow specifying the point at which to ‘‘pin’’ the zeros, in plot-space coordinates (0 to 1).`
* `-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`*`} | use={name | symbol | description}][,units][,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, or the user may specify with the `use` qualifier that the element name, symbol, or descrpition be used. The user may also force the appearance of the units on the label using the `units` qualifier. The label text may be edited using Toolkit editing commands (`SDDS editing` (4.69)).
  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 displayed 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`* — **Obsolete**: Specifies a common size for all labels, including numeric labels. In the original version of `sddsplot`, the *fraction* was the horizontal size of the characters as a fraction of the horizontal size of the plot region. This meaning is no longer precisely true because the new version doesn't used fixed character sizes. However, this option may still be used to scale character sizes up and down. The previous nominal value for *fraction* was 0.03, which is now used as the reference point for scaling. Hence, if you specify 0.06, the character sizes would be doubled.
* `-noLabels`—Specifies that no labels (i.e., x and y dimension labels, title, and topline label) will be made.
* `-string={@`*`parameterName`*` | `*`string`*`},{x|p}Coordinate=`*`value`* `{y|q}Coordinate=`*`value`*`[,scale=`*`factor`*`][,angle=`*`degrees`*`] [,justifyMode=`*`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 `justifyMode` 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` (4.69)).

* `-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:

* `-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.
* `-factor=[{x|y}Multiplier=`*`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.
* `-offset=[{x|y}Change=`*`value`*`][,{x|y}Parameter=`*`value`*`][,{x|y}Invert]` — Specifies that the x and/or the y data be offset by either specified values, qor by values in named numerical parameters. Normally, the offset is of the form $x \rightarrow x + x_o$. The `invert` qualifiers cause the offset to be subtracted rather than added.

  If `-factor` is given together with `-offset`, then the offset is applied first.
* `-mode={x | y}={linear | logarithmic | normalize | offset | coffset | center | meanCenter | specialScales}[,...]` —

  Invokes one or more standard transformations of data, independently for x and y values. The `linear` mode is the default. `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. The `offset`, `coffset`, `center`, and `meanCenter` qualifiers result in shifting of the data: `offset` directs that data be shifted so that the first value plotted is zero; `coffset` directs the data

to use a common offset from the first plot; `center` directs that data be shifted the center of the range is zero; `meanCenter` directs that the data be shifted so that the average plotted value is zero.
`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. One can also use the `-tickSettings` option for this, which is the preferred method.

  * `-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*`][,limit=`*integer*`][,scale=`*factor*`] [,allTicks][,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 Nth enumerated value; the default is 1. Also, `limit=`*M* specifies displaying and making a tick for only M enumerated values at equal spacing; the default is unlimited. If one of these options is employed but one desires to see all the ticks (even those without labels), the `allTicks` qualifier may be given. `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 and data labels:
  * `-legend[={ {x|y}Symbol | {x|y}Description | {x|y}Name | 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. `xName` and `yName` specify use of the indicated datanames. `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 commands`SDDS 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.

  * `-pointLabel=`*name*`[,edit=`*editCommand*`][,scale=`*number*`][,justifyMode={rcl}{bct}]` — Specifies labeling of individual data points using data from column or parameter *name*. The labels may be edited by specifying an *editCommand* with the `edit` qualifier. The `scale` qualifier may be used to scale the label size. The `justifyMode` qualifer is used to change the location of the label relative to the point; the first letter gives the horizontal justification (right, center, or left) and the second gives the vertical justification (bottom, center, or top).

- Creating overlays:

  The overlay feature allows displaying data that has different scales on the same plot. In most cases, it is superseded by the `-yScalesGroup` and `-xScalesGroup` options. The only exception is when one wants to overlay data without having scales shown for the data. (An example is plotting magnet layouts for Twiss parameter plots using the `magnets` output from `elegant`.)

  `-overlay=[{x|y}mode=`*mode*`][,{x|y}factor=`*value*`][,{x|y}offset=`*value*`] [,{x|y}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` qualifiers invoke a more sophisticated and more generally applicable form of severing. 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][,nameString][,iNameString]` — 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 | iNameString}]` — 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=[{x|y}Minimum=`*value*`][,{x|y}Maximum=`*value*`][,autoscaling]`— 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.

  The `autoscaling` qualifier specifies that `sddsplot` will not remove data outside the defined limits, but rather that it will ignore it for purposes of autoscaling. If lines are used to connect data points, this could result in lines being drawn to the boundary of the plot region, thus showing the presence of extreme points.
* `-sparse=`*interval*`[,`*offset*`]`—Specifies that only every *intervalth* point will be used. If *offset* is not given, the first point in the internal data set is the first taken; otherwise, the *offsetth* 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.
* `-presparse=`*interval*`[,`*offset*`]`—Similar to `-sparse`, except that sparsing is done at the time the data page is read and only once for all requests and datanames that draw data from the data page. This is faster, and is usually what is desired. However, if one wants to plot sparsed and unsparsed data from the same file at the same time, `-presparse` cannot be used. If both `presparse` and `sparse` are given, both are applied.
* `-filter={column | parameter},`*rangeSpec*`[,`*rangeSpec*`,`*logicOp*`...]` — 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={column | parameter},`*matchTest*`[,`*matchTest*`,`*logicOp*`]` — Specifies winnowing based on data in string parameters or columns. A *matchTest* is of the form *name*`=`*matchingString*`[,!]`, where the matching string may include wildcards. 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.

  The use of several match tests and logic is done just as for `-filter`. For example, to match all the rows for which the column `Name` starts with 'A' or 'B', one could use `-match=column,Name=A*,Name=B*,|`. (This could also be done with `-match=column,Name=[AB]*`.)

– Miscellaneous:

* `-repeat[=checkSeconds=`*number*`][,timeOut=`*seconds*`]` — Specifies repeated plotting of data from the files, with replotting occuring when any file is modified. By default, `sddsplot` checks the files every second and times out after 900s of no change. This is available on UNIX systems only. It is best used with the `motif` device type and the following device argument: `-device=motif,''-movie true -keep 1''`.
* `-drawLine=`
  {`x0Value=`*value*`|p0Value=`*value*`|x0parameter=`*name*`|p0parameter=`*name*}
  {`x1Value=`*value*`|p1Value=`*value*`|x1parameter=`*name*`|p1parameter=`*name*}
  {`y0Value=`*value*`|q0Value=`*value*`|y0parameter=`*name*`|q0parameter=`*name*}
  {`y1Value=`*value*`|q1Value=`*value*`|y1parameter=`*name*`|q1parameter=`*name*}
  — Specifies drawing of lines on the plot by giving the two endpoints of the line. For each endpoint (labeled '0' and '1'), one must specify the x or p coordinate (for horizontal) and the y or q coordinate (for vertical). Each coordinate name be specified explicitly (e.g., `x0Value=1.7`) or via a parameter (e.g., `x0parameter=alpha`). If a parameter is given, the coordinate can change as the parameter value changes in the file.

- **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 `$r` switches back to Roman mode. The correspondance between Greek characters and the alphabet is shown in Figure 1. For example, to make a lowercase alpha, one would use `$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 1 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:**
  - `SDDS_DEVICE` — Gives the name of the device type to use as the default.

Figure 1: Special character set

- **see also:**
  - Data for Examples (see 3.3)
  - `SDDS editing` (4.69)
  - `SDDS Wildcard Conventions` (4.71)
- **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 X-windows program (`mpl_motif`) was written by K. Evans of ANL/APS.

  The GIF drivers use the `gd 1.2` library by Thomas Boutell. The latter is copyrighted by the Quest Protein Database Center, Cold Spring Harbor Labs.

4.49 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]
[-columns[=nameList[,format=string][,label=string][,editLabel=command]
    [,useDefaultFormat][,endsLine][,blankLines=number]]]
[-parameters[=nameList[,format=string][,label=string][,editLabel=command]
    [,useDefaultFormat][,endsLine][,blankLines=number]]]
[-spreadsheet[=delimiter=string][,quoteMark=string][,noLabels][,schFile=filename]]
[-fromPage=number] [-toPage=number]
[-formatDefaults=SDDStype=formatString[,...]]  [-width=integer]
[-pageAdvance] [-paginate[=lines=number][,noTitle][,noLabels]]
[-postPageLines=number] [-title=string] [-noTitle] [-noWarnings]
```

- **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.
  - `-columns[=`*nameList*`[,format=`*string*`][,label=`*string*`][,editLabel=`*command*`]` `[,useDefaultFormat][,endsline][,blankLines=`*number*`]]` — 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`. The `useDefaultFormat` qualifier directs that `sddsprintout` use its own default format for the data type in question, as opposed to any format that might be specified in the SDDS header.

    The `label` qualifier can be used to specify the column label in the printout (by default, the column name is used); the label may be edited using the `editLabel` qualifier and a standard editing sequence.

    If the `endsLine` qualifier is given, a line break is issued after the last column of the list is printed. The `blankLies` qualifier may be used to specify that one or more blank lines be emitted following such a line break.

    Any number of `-columns` options may be given.
  - `-parameters...` — Identical to `-columns`, except that printout of parameters is specified.

–

`-spreadsheet[=delimiter=`*string*`][,quoteMark=`*string*`][,noLabels][,schFile=`*filename*`]` — Specifies spreadsheet compatible output, using the given delimiter between columns. In this mode, simplified header is printed and no line width limits are imposed. The default delimiter is a tab. The default quotation mark is ·: If the `schfile` qualifier is given, a header file for comma-separated-values data is generated. In this case, the delimiter should be a comma.

– `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*.

– `-width=`*integer* — Specifies the width of the output line in number of characters. The default is 130.

– `-pageAdvance` — Specifies that the page be advanced at the end of every data page of the SDDS file. This is done by emitting any ASCII page advance character, which will probably work only if the output is sent to a printer.

– `-paginate` — Specifies pagination of the output, using a default 66 line page. The `lines` qualifier may be used to change the page length. By default, the title and column labels are printed for each page. These may be disabled using the `notitle` and `nolabels` qualifiers.

– `-postPageLines` — Specifies that a number of blank lines shall be emitted at the end of the printout for each page. By default, there are no blank lines between pages.

– `-title=`*string* — Specifies the title for the printout.

– `-noTitle` — Specifies that no title be printed.

– `-noWarnings` — Suppresses warning messages, such as those concerning data elements requested in the printout that are not in the input file.

- **see also:**
  – Data for Examples (see 3.3)
  – `sdds2spreadsheet` (4.7)
  – `sdds2stream` (4.8)
- **author:** M. Borland, ANL/APS.

4.50 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`*`[,`*`logicOp`*`...]]` —
      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). For example, to select rows for which A is on [0, 1] and B is on [10, 20], one would use `-filter=column,A,0,1,B,10,20,&`.

* `-match={column | parameter},`*`matchTest`*`[,`*`matchTest`*`,`*`logicOp`*`]` — 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).
  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.
  In addition, if instead of `=` one uses `=+`, then matching is case-insensitive. The plus sign is intended to be mnemonic, as the case-insensitive matching results in additional matches.
  The use of several match tests and logic is done just as for `-filter`. For example, to match all the rows for which the column `Name` starts with 'A' or 'B', one could use `-match=column,Name=A*,Name=B*,|`. (This could also be done with `-match=column,Name=[AB]*`.)
* `-numberTest={column | parameter},`*`name`*`[,invert]` — Specifies testing the values of in a string column (parameter) to see if they can be (or cannot be, if `invert` is given) converted to numbers. If not, the corresponding row (page) is deleted.
* `-test={column | parameter},`*`test`*`[,autostop][,algebraic]` — 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.
* `-fclip=`*`head`*`,`*`tail`*`[,invert]` — Specifies the fraction 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 sparsing of each page with the indicated interval. That is, only every *intervalth* 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][,algebraic]` — 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*.
  If *sourceName* contains wildcards, then *newName* must contain at least one occurrence of the string "%s". In this case, for each name that matches *sourceName*, an additional element is created, with a name created by substituting the name for "%s" in *newName*.
* `-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 4.69).
  If *sourceName* contains wildcards, then *newName* must contain at least one occurrence of the string "%s". In this case, for each name that matches *sourceName*, an additional element is created, with a name created by substituting the name for "%s" in *newName*.
* `-reedit={column | parameter},`*`name`*`,`*`edit-command`* — Like `-edit`, except that the element *name* must already exist. Each value is replaced by the value obtained from applying *edit-command*.
* `-print={column | parameter},`*`newName`*`,`*`sprintfString`*`,`*`sourceName`*
`[,`*`sourceName...`*`][,`*`definitionEntries`*`]` — Specifies creation of a new string column (parameter) by formatted printing of one or more elements from other columns (parameters). The *sprintfString* is a C-style format string such as might be given to the routine `sprintf`. With the exception of the `name` field, any valid namelist command field and value may be given as part of the *definitionEntries*.
* `-reprint` — Identical in syntax and function to `-print`, except that if *newName* already exists, it is overwritten. No error or warning is issued.
* `-format={column | parameter},`*`newName`*`,`*`sourceName`*
`[,stringFormat=`*`sprintfString`*`][,doubleFormat=`*`sprintfString`*`]`
`[,longFormat=`*`sprintfString`*`]` — Reformats string data in different ways depending on the type of data the string contains. Each string is separated into tokens at space boundaries. Each token is separately formatted, either as a long integer, a double-precision floating point number, or a string, depending on what the token appears to be. The formatting is done using the specified format strings; the default format strings are %ld for longs, %21.15e for doubles, and %s for strings.
* `-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). If *commandName* contains wildcards, then *newName* must contain at least one occurrence of "%s". In this case, for each name that matches *commandName*, an additional element is created, with a name created by substituting the name for "%s" in *newName*.

– Creation and modification of numeric columns and parameters:

- `-convertUnits={column | parameter},`*`name,newUnits,oldUnits,factor`*
— Specifies units conversion for the column or parameter *name* (which may contain wildcards). 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`*`[,select=`*`matchString`*`]`
`[,editSelection=`*`editCommand`*`][,`*`definitionEntries`*`][,algebraic]`
— 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.
Using the `select` qualifier, it is possible to use a single `-define` option to specify many instances of new column definitions. If `select` is given, the input is searched for all the column names matching *matchString*. These are then optionally editted using the *editCommand* specified with `editSelection`. The resulting strings are then substituted one at a time into *name* and *equation*, replacing all occurances of "%s". For example, suppose a file contained a number of column-pairs of the form *Prefix*V1 and *Prefix*V2; to take the difference of each pair, one could use
`-define=column,%sDiff,%sV1 %sV2 -,select=*V1,edit=%/V1//`

  `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.
- `-cast={column | parameter},`*`newName,oldName,newType`* — This option allows casting data from one numerical data type to another. It is much faster than trying to do the same operation using `-define`. The string *newType* may be any of `double`, `float`, `long`, `short`, or `character`.
- `-process=`*`mainColumnName,analysisName,resultName`*
`[,description=`*`string`*`][,symbol=`*`string`*`][,weightBy=`*`columnName`*`]`
`[,functionOf=`*`columnName`*`[,lowerLimit=`*`value`*`][,upperLimit=`*`value`*`]]`
`[,head=`*`number`*`][,tail=`*`number`*`][,fhead=`*`fraction`*`][,ftail=`*`fraction`*`]`

[,topLimit=*value*] [,bottomLimit=*value*]
[,position] [,offset=*value*] [,factor=*value*]
[,match=*columnName*,value=*match-value*] — This option may be given any number of times. It specifies creation of a new parameter *resultName* by processing column *mainColumnName* using analysis mode *analysisName*. The column must contain numeric data, in general, except for a few analysis modes that take any type of data (see below). *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 "%s" embedded in it; if so, *mainColumnName* is substituted. If wildcards are given in *mainColumnName*, then "%s" must appear in *resultName*; in this case, the name of each selected column is substituted. Similarly, if the `description` field is supplied, it may contain an embedded "%s" for which the column name will be 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`, `drange`, `qrange` — The median value, i.e., the value which is both above and below 50the quartile-range, which is the range excluding the smallest and largest 25
- `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; the `functionOf` qualifer may name a string column.
- `first`, `last` — The values in the first and last rows of the page. Will accept non-numeric data.
- `pick` — The first value within the filter. Will accept non-numeric data.
- `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*.
- `sigma` — The standard deviation over the square-root of the number of points. This is an estimate of the uncertainty in the mean value.
- `slope`, `intercept`, `lfsd` — The slope and intercept of a linear fit. The `functionOf` qualifier must be given to specify the quantity to fit against. `lfsd`

is the Linear-Fit-Standard-Deviation, which is the standard deviation of the fit residuals.

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.

· `topLimit=`*value*, `bottomLimit=`*value* — Specifies winnowing of rows so that only those with *mainColumn* values above the `topLimit` or below the `bottomLimit` are included in the computations.

· `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.

· `head=`*number*, `fhead=`*fraction* — Specifies taking the head of the data prior to processing. `head` gives the number of points keep, while `fhead` gives the fraction of the points to keep. If *number* or *fraction* is less than 0, thenthe head points are deleted and the other points are kepts. If head and tail are both used, head is performed first. gives the fraction of the points to clip.

· `tail=`*number*, `ftail=`*fraction* — Specifies taking the tail of the data prior to processing. `tail` gives the number of points keep, while `ftail` gives the fraction of the points to keep. If *number* or *fraction* is less than 0, thenthe tail points are deleted and the other points are kepts. If head and tail are both used, head 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 $x \rightarrow f * (x + o)$.

· `match=`*controlName*, `value=`*match-value* — Specify the match column and the match value (may contain wildcard).

– 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.

∗ `-description=[text=`*string*`][,contents=`*string*`]` — Specifies the description fields for the SDDS dataset. Use of this feature is disparaged as these fields are not manipulated by any tools. Use of string parameters is suggested.

∗ `-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.
* `-delete={columns | parameters | arrays},`*matchingString*`[,...]`,
  `-retain={columns | parameters | arrays},`*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.

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

4.51 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 for the output file are generated either from the data in a selected string column in the input file, from the value of the command line option -root, or from an internal default.

  The column names of the input file are collected and made into a string column in the output file.

- **examples:** The matrix of R_{12} values for some accelerator beamline called LTP is stored in file LTP.R12. The pseudo-inverse (useful for trajectory correction), named LTP.InvR12, is created with:

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

- **synopsis:**

```
sddspseudoinverse [-pipe=[input][,output]] inputFile outputFile
[-minimumSingularValueRatio=value | -largestSingularValues=number]
[-oldColumnNames=string] [{-root=string [-digits=integer] |
-newColumnNames=column}] [-umatrix=file] [-vmatrix=file] [-removeDCVectors]
[-weights=file,name=columnName,value=columnName] [-symbol=string] [-ascii]
[-verbose] [-noWarnings]
```

- **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.

  Multiple data pages of the input file will be processed and written to the outptu file if all the data pages of the input file have the same number of rows. The processing will stop at the first data page which doesn't have the same number of rows as that of the first page. If applicable, the string column selected to generate column names for the output file is assumed to be the same in all input data sets. The string columns of only the first data set are read.

- **switches:**
  - `-pipe[=input][,output]` — The standard SDDS Toolkit pipe option.
  - `-minimumSingularValueRatio=`*realValue* — Used to remove small singular values from the calculation. The smallest singular value retained for the inverse calculation is determined by multiplying this ratio value with the largest singular value of the input matrix.
  - `-largestSingularValues=`*integer* — Used to remove small singular values from the calculation. The largest *integer* singular values are kept.

- `-oldColumnNames=`*string* — A string column of name *string* is created in the output file, containing the column names of the input files as string data. If this option is not present, then the default name of "OldColumnNames" is used for the string column.
- `-root=`*string* — A string used to generate columns names for the output file data. The first data column is named "*string*`000`", the second, "*string*`001`", etc.
- `-digits=`*integer* — minimum number of digits used in the number appended to *root* of the output file column names. (Default value is 3).
- `-newColumnNames=`*string* — Specifies a string column of the input file which will be used to define column names of the output file.
- `-umatrix=`*file* — writes the u column-orthogonal matrix to a file. The SVD decomposition follows the convention $A = uSv^T$. The "transformed" x are $v^T x$, and the "transformed" y are $u^T y$.
- `-vmatrix=`*file* — writes the v column-orthogonal matrix to a file.
- `-removeDCVectors` — Removes the eigenvectors which have an overall DC component.
- `-weights=`*file*`,name=`*columnName*`,value=`*columnName*`]` — Specifies a file which contains weights for each of the rows of the matrix, thus giving different weights for solving the linear equations of the pseudoinverse problem. The equation that is solved is $wAx = wy$ where w is the weight vector turned into a diagonal matrix and A is the input matrix. The matrix solution returned is $(wA)^I w$ where $()^I$ means taking the pseudoinverse. The u matrix now has a different interpretation: the "transformed" x are $v^T x$, as before, but the "transformed" y are $u^T wy$.
- `-symbol=`*string* — The string for 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

4.52 sddsquery

- **description:**

  `sddsquery` prints a summary of the SDDS header for a data set. It 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. Finally, it will create an SDDS file containing information about what is in the header.

- **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=" "`
  ```

  Get a list of the column names into an SDDS file

  ```
  sddsquery APS.twi -columnList -sddsOutput=APS.twi.names
  ```

- **synopsis:**

  ```
  sddsquery SDDSfilename [SDDSfilename...]  [-sddsOutput[=filename]]
  [{-arrayList | -columnList | -parameterList | -version}]
  [-delimiter=delimitingString] [-appendUnits[=bare]] [-readAll]
  ```

- **files:** The input filenames may name arbitrary SDDS files.

  If `-sddsOutput` is given, the output normally contains one page for each data class (i.e., array, parameter, and column). The following elements are defined:

  - Columns (all string type):
    - `Name` — The name of the element.
    - `Units` — The units of the data.
    - `Symbol` — The symbol for the element.
    - `Format` — The format string for the element (e.g., "%f").
    - `Type` — The SDDS data type name (e.g., double, float, etc.).
    - `Description` — The description for the element.
    - `Group` — The group name (for array elements only).
  - Parameters:
    - `Class` — The SDDS class for the present page.
    - `Filename` — The filename being described by the present page.

- **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.

  - `sddsOutput[={\em filename}]]` — Requests that output be delivered in SDDS protocol. If no *filename* is given, the output is delivered to the standard output.
  - `arrayList` — In non-SDDS output mode, requests that a list of array names be printed to the standard output, one name per line. In SDDS output mode, requests that only array information be provided.
  - `columnList` — In non-SDDS output mode, requests that a list of column names be printed to the standard output, one name per line. In SDDS output mode, requests that only column information be provided.
  - `parameterList` — In non-SDDS output mode, requests that a list of parameter names be printed to the standard output, one name per line. In SDDS output mode, requests that only parameter information be provided.
  - `-version` — Requests that the SDDS version number of the file be printed to the standard output. Valid in non-SDDS output mode only.
  - `-delimiter=`*`delimitingString`* — Requests that listed items be separated by the given string. By default, the delimiter is a newline. Valid in non-SDDS output mode only.
  - `-appendunits[=bare]` — Requests that the units of each item be printed directly following the item name. Valid in non-SDDS output mode only. If the `bare` qualifier is not given, then the units are enclosed in parentheses.
  - `-readAll` — Forces `sddsquery` to read the entire file. On some operating systems this is necessary when querying compressed files to prevent "Broken Pipe" errors. For large files, use of this option will make `sddsquery` slower.

- **see also:**

  - Data for Examples (see 3.3)

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

4.53 sddsregroup

- **description:** `sddsregroup` swaps the row indexing and page indexing of data in an SDDS file. That is, the ith row of all data pages in the input file are collected and made into the ith 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

4.54 sddsrowstats

- **description:** `sddsrowstats` analyzes data across columns on a row-by-row basis to find minima, maxima, averages, standard-deviations, etc. The output is a copy of the input with additional columns that contain the desired statistics.
- **examples:** Find the mean x and y orbits from PAR beam-position-monitor data collected with one set of x and y values (in 32 columns) per row.

  ```
  sddsrowstats par.bpm par.bpm1 -mean=xMean,P?P?x -mean=yMean,P?P?y
  ```

- **synopsis:**

  ```
  sddsrowstats [-pipe=[input][,output]] [input] [output] [-nowarnings]
  [-mean=newName,[,limitOps],columnNameList]
  [-rms=newName,[,limitOps],columnNameList]
  [-median=newName[,limitOps],columnNameList]
  [-minimum=newName[,limitOps],columnNameList]
  [-maximum=newName[,limitOps],columnNameList]
  [-standardDeviation=newName[,limitOps],columnNameList]
  [-sigma=newName[,limitOps],columnNameList]
  [-mad=newName[,limitOps],columnNameList]
  [-sum=newName[,limitOps][,power=<integer>],columnNameList]
  [-drange=newName[,limitOps],columnNameList]
  [-qrange=newName[,limitOps],columnNameList]
  [-smallest=newName[,limitOps],columnNameList]
  [-largest=newName[,limitOps],columnNameList]
  [-count=newName[,limitOps],columnNameList]
  ```

  where *columnList* is a comma-separated list of one or more optionally wildcarded names and *limitOps* is of the form `[topLimit=`*`value`*`,][bottomLimit=`*`value`*`]`.

- **switches:**
  - `-pipe=[input][,output]` — The standard SDDS Toolkit pipe option.
  - `-mean`, `-rms`, `median`, `minimum`, `maximum`, `standardDeviation`, `sigma`, `mad`, `drange`, `qrange`, `smallest`, `largest`, `count` — Compute indicated statistic across the columns specified in *columnList*. If *limitOps* are given, then values above the `topLimit` or below the `bottomLimit` are excluded from computations. `sigma` is the standard deviation of the mean. `mad` is the mean-absolute-deviation. `smallest` (`largest`) is the minimum (maximum) absolute value. `drange` and `qrange` are the decile and quartile ranges, respectively.
  - `-sum=`*`newName`*`[,power=`*`integer`*`],`*`columnNameList`* — Specifies creation of a new column *newName* containing the row-by-row sums of the columns specified in *columnList*. The values are summed after being raised to the given power, which is 1 by default.

- **see also:**
  - Data for Examples (see 3.3)
  - `sddschanges` (4.12)

  - `sddsenvelope` (4.28)
  - `sddsprocess` (4.50)

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

4.55 sddsrunstats

- **description:** `sddsrunstats` computes running or blocked statistics on SDDS tabular data.
- **examples:** Smooth PAR x beam-position-monitor data by using a sliding window 32 points long:

  ```
  sddsrunstats par.bpm par.bpm.rs -mean=Time,P?P?x
  ```

  Same, but use nonoverlapping window for averages:

  ```
  sddsrunstats par.bpm par.bpm.rs -mean=Time,P?P?x -noOverlap
  ```

- **synopsis:**

  ```
  sddsrunstats [-pipe[=input][,output]] [input] [output] [-points=integer |
  -window=column=column,width=value] [-noOverlap] [-partialOk]
  [-mean=[limitOps],columnNameList] [-minimum=[limitOps],columnNameList]
  [-maximum=[limitOps],columnNameList]
  [-standardDeviation=[limitOps],columnNameList]
  [-sigma=[limitOps],columnNameList]
  [-sum=[limitOps][,power=integer],columnNameList]
  [-sample=[limitOps],columnNameList]
  ```

  where *columnNameList* is a comma-separated list of one or more optionally wildcarded names and *limitOps* is of the form `[topLimit={\em value},][bottomLimit={\em value}]`.

- **switches:**
  - `-pipe=[input][,output]` — The standard SDDS Toolkit pipe option.
  - `points=`*`integer`* — The number of points in the statistics window for each output row. If non-overlapping statistics are used, the last output row will be computed from fewer than the specified number of points if the input file number of rows is not a multiple of the specified number of points.
  - `window=column=`*`column`*`,width=`*`value`* — Specifies a column to use for determining statistics row boundaries. For example, one might want statistics for 60 second blocks of data when the data is not uniformly sampled in time. In this case, the column would be "Time" (say) and the width 60.
  - `partialOk` — Specifies that `sddsrunstats` should do computations even if the number of available rows is less than specified. By default, such data is simply ignored.
  - `-noOverlap` — Specifies non-overlapping statistics. The default is to compute running statistics with a sliding window.
  - `-mean=[`*`limitOps`*`],`*`columnNameList`*
    `-minimum=[`*`limitOps`*`],`*`columnNameList`*
    `-maximum=[`*`limitOps`*`],`*`columnNameList`*
    `-standardDeviation=[`*`limitOps`*`],`*`columnNameList`*
    `-sigma=[`*`limitOps`*`],`*`columnNameList`* — Specifies computation of the indicated statistic for the columns matching *columnNameList* (see above). The standard deviation is N-1 weighted. Sigma is the standard deviation of the sample mean. *limitOps* (see above for syntax) allows filtering the points in each window to exclude values above the `topLimit` or below the `bottomLimit`.

- `-sum=[`*limitOps*`][,power=`*integer*`],`*columnNameList* — Specifies computation of a general sum of powers of values. For example, to get the sum of squares you'd use `power=2`. *columnNameList* and *limitOps* are as for the last item.
- `-sample=[`*limitOps*`],`*columnNameList* — Results in extraction of a single set of values per group, namely, the first value in the group that passes the *limitOps* criteria.

- **see also:**
  - Data for Examples (see 3.3)
  - `sddssmooth` (4.61)
- **author:** M. Borland, ANL/APS.

4.56 sddssampledist

- **description:** `sddssampledist` provides for psuedo-random sampling of probability distributions. It also provides nonrandom sampling, using Halton sequences.
- **example:** Draw some random samples from a normal (gaussian) distribution, G(z), shifted to have a sigma of 10 and centroid of 5.

```
sddssampledist gaussian.sdds samples.sdds -samples=100
-columns=indep=z,df=G,output=zSample,factor=10,offset=5
```

- **synopsis:**

```
sddssampledist [input] [output] [-pipe=[in][,out]]
-columns=independentVariable=name,cdf=CDFName | df=DFName
[,output=name][,units=string][,factor=value]
[,offset=value][,datafile=filename]
[,haltonRadix=primeNumber[,randomize[,group=groupID]]] [-columns=...]
[-samples=integer] [-seed=integer]
```

- **files:**

  input is the default input file for distribution functions (DFs) and cumulative distribution functions (CDFs). *input* need not be given if all `-column` options give the `datafile` qualifier.

  output contains the samples. The names of the sampled data are by default the same as the names of the independent variable from the `-column` options. These names may be changed by the `output` qualifier of that option.

- **switches:**
  - `pipe[=input][,output]` — The standard SDDS Toolkit pipe option.
  - `columns=independentVariable=`*name*`,{cdf=`*CDFName* `| df=`*DFName*`}` `[,output=`*name*`][,units=`*string*`][,factor=`*value*`][,offset=`*value*`]` `[,datafile=`*filename*`][,haltonRadix=`*primeNumber*`[,randomize[,group=`*groupID*`]]]` — Any number this option may be given. Specifies the CDF or DF from which to draw samples (`cdf` or `df` qualifier), as well as the variable that the CDF or DF depends on (`independentVariable` qualifier). The samples are values of this variable. `output` allows specifying the name of the column for the samples, while `units` allows specifying the units. `factor` and `offset` may be used to perform a simple transformation of the sample values, according to $x \rightarrow x * f + o$. `datafile` allows specifying an alternate file as the source for the distribution function data. By default, the data is drawn from the main input file. `haltonRadix` allows specifying the radix for generation of a nonrandom Halton sequence, which provides a much smoother sampling of the distribution than does a pseudo-random sequence. The radix should be a small prime number. If you generate multiple sequences from the same radix, they will be correlated. Hence, the `randomize` qualifier should be used to remove the correlations. If there are multiple `column` options that should be randomized together (i.e., randomized relative to other data but not each other), the `group` qualifier can be used to assign these options to a specific *integer) group ID.

- `samples` — Specifies the number of samples to generate.
- `seed` — Specifies the seed for the random number generation. Should be a large, odd integer. If not given, the system clock is used to generate a seed.

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

4.57 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 APSnquad.twi -match=ElementName -reuse
  -invert
  ```

- **synopsis:**

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

- **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*.
  - `-noWarnings` — Specifies that no warning messages (about, e.g., file length mismatches or file overwrites) should be issued.

- **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` (4.67)
- **author:** M. Borland, ANL/APS.

4.58 sddssequence

- **description:** `sddssequence` generates an SDDS file with a single page and several columns of data of arithmetic sequences. An example application is generating values for an independent variable, whose values can be used by `sddsprocess` to produce a mathematical function.

- **examples:** Make a file example.sdds with column Index of type long with 100 rows. The value of Index in the first row is 1 and is incremented by 1 for each successive row.

```
sddssequence example.sdds -define=Index -sequence=begin=1,number=100,delta=1
```

- **synopsis:**

```
sddssequence [-pipe=[output]] [<outputfile>]
-define=<columnName>[,<definitionEntries>] [-repeat=<number>]
-sequence=begin=<value>[,number=<integer>][,end=<value>]
[,delta=<value>][,interval=<integer>]
[-sequence=[begin=<value>][,number=<integer>][,end=<value>]
[,delta=<value>][,interval=<integer> ...]
```

- **files:** *outputfile* is the name of the SDDS file containing data generated.

- **switches:**

  - `-pipe=[output]` — The standard SDDS Toolkit pipe option.
  - `-define=<columnName>[,<definitionEntries>]` — Defines a new column. One or more `-sequence` options should follow. Definition entries have the form `fieldName=value` where `fieldName` is the name of any namelist command field (except the name field) for a column. The default data type is double. To get a different type, use `type=<typeName>`.

    Multiple `-define` options can be used to create mutliple columns, each with their own set of `-sequence` options.
  - `-sequence=begin=<value>[,number=<integer>][,end=<value>]` `[,delta=<value>][,interval=<integer>]` — Defines the arithmetic sequence for the data column. More that one `sequence` option can be given for a column definition, thus allowing arithmetic sequences of different character in one column. The `begin` value must be given in the first `sequence` option. If any `sequence` options follows immediately, a default value equal to the previous `end` value plus the previous `delta` value is used. For the rest of the suboptions, the user must supply (`end`, `delta`), (`end`, `number`), or (`delta`, `number`). If `number` isn't supplied, then the set of `start`, `end`, `delta` must imply a positive number of rows.

    The `interval` field specifies the number of rows for which the value is frozen within the sequence. For instance,

    ```
    sddssequence example.sdds -def=Index,type=long -sequ=beg=1,num=10,end=10,inte=2
    sddsprintout -col=Index example.sdds
    ```

    produces

```
Printout for SDDS file example.sdds

  Index
------------
         1
         1
         3
         3
         5
         5
         7
         7
         9
         9
```

  - `-repeat=<number>` — Repeats the sequence identically for the given number of times.

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

4.59 sddsshiftcor

- **description:** `sddsshiftcor` computes correlation coefficients and correlation significance between column data as a function of shifting of the data columns relative to each other. 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. In some cases, signals are correlated but with a time-lag. Hence, computing

$$C_{ij}$$

as a function of the shifting of one of the signals may reveal relationships that are not apparent in a simple correlation, such as might be done with `sddscorrelate`.

- **synopsis:**

```
sddsshiftcor [-pipe=[input][,output]] [inputFile] [outputFile]
-with=columnName
[-scan[=start=startShift][,end=endShift][,delta=deltaShift]]
[-columns=columnNames] [-excludeColumns=columnNames] [-rankOrder]
[-stDevOutlier[=limit=factor][,passes=integer]] [-verbose]
```

- **files:** *inputFile* is an SDDS file containing two or more columns of data. *outputFile* contains one column (`ShiftedBy`) for the amount shifted, plus one column for each analyzed column in *inputFile*. The latter each contains the correlation coefficient with the shifted signal for the given shift value.

- **switches:**
  - `-pipe=[input][,output]` — The standard SDDS Toolkit pipe option.
  - `-with=`*`columnName`* — Specifies the column to be shifted, which is correlated with the other columns.
  - `-scan[=start=`*`startShift`*`][,end=`*`endShift`*`][,delta=`*`deltaShift`*`]` — Specifies the amount to shift and the step size. The values are all integers. By default *startShift*=-10, *endShift*=10, and *deltaShift*=1
  - `-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.
  - `-rankOrder` — Specifies computing rank-order correlations rather than standard correlations. This is considered more robust that standard correlations.
  - `-stDevOutlier[=limit=`*`factor`*`][,passes=`*`integer`*`]` — Specifies standard-deviation-based outlier elimination on each pair of columns prior to computation of the correlation coefficient. Any pair of values is ignored if one or both values are outliers relative to the

column from which they come. The `limit` qualifier specifies the allowed deviation from the mean in standard deviations; the default is 1. The `passes` qualifier specifies how many times the outlier elimination (including recomputation of the mean and standard deviation) is performed; the default is 1.

- **see also:**
  - `sddscorrelate` (4.22)
- **author:** M. Borland, ANL/APS.

4.60 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 of slopes and intercepts for each data set of the input file. Errors on the slope and intercept may be calculated as an option.
- **examples:** The file corrector.sdds below 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 the use of `sddsslopes`:

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

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

- **synopsis:**

```
sddsslopes [-pipe=[input][,output]] inputFile outputFile
-independentVariable=parameterName [-range=lower,upper]
[-columns=listOfNames] [-excludeColumns=listOfNames] [-sigma[=generate]]
[-residual=file] [-ascii] [-verbose]
```

- **files:** The input file contains the tabular data for fitting. Multiple data sets are processed one at a time. 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 a one-row data set for each data set in the input file. 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).
  - `-range=`*lower*`,`*upper* — The range of the independent variable where the fit is calculated. By default, all data points are used.
  - `-columns=`*listOfNames* — columns to be individually paired with independentVariable for straight line fitting.
  - `-excludeColumns=`*listOfNames* — 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*.

  - `-residual=`*file* — Specifies an output file into which the residual of the fits are written. The column names in the residual file are the same as they appear in the input file.
  - `-ascii` — make output file in ascii mode (binary is the default).
  - `-verbose` — prints some output to stderr.

- **author: L. Emery** ANL

4.61 sddssmooth

- **description:**

  `sddssmooth` smooths columns of data using multipass nearest-neighbor averaging and/or despiking. Any number of columns may be smoothed. The smoothed data may be put in place of the original data, or included as a new column.

  Nearest-neighbor averaging involves repeatedly replacing each point by the average of its N nearest-neighbors; this is the type of smoothing that is done if nothing is specified. Despiking consists of replacing the most extreme of N nearest neighbors with the average of the same points; the most extreme point is the one with the largest mean absolute difference from the other points.

- **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]
[-SavitzkyGolay=left,right,order,[derivativeOrder]]
[-despike[=neighbors=integer][,passes=integer]] [-newColumns]
[-differenceColumns]
```

- **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 nearest-neighbor-averaging smoothing passes to make over each column of data. The default is 1. If 0, no such smoothing is done. In the limit of an infinite number of passes, every point will tend toward the average value of the original data. If `-despike` is also given, then despiking occurs first.

  – `-SavitzkyGolay=`*`left,right,order,`*`[`*`derivativeOrder`*`]` — Specifies smoothing by use of a Savitzky-Golay filter, which involves fitting a polynomial of order *order* through *left*+*right*+1 points. Optionally, takes the *derivativeOrder*-th derivative of the data. If this option is given, the nearest-neighbor-averaging smoothing is not done. If `-despike` is also given, then despiking occurs first.

  - `-despike[=neighbors=`*integer*`][,passes=`*integer*`]` — Specifies smoothing by despiking, as described above. By default, 4 nearest-neighbors are used and 1 pass is done. If this option is not given, no despiking is done.
  - `-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.
  - `-differenceColumns` — Specifies that additional columns be created in the output file, containing the difference between the original data and the smoothed data. The new columns are given names of the form *columnName*`Unsmooth`, where *columnName* is the orignal name of the column.

- **see also:**
  - `sddsdigfilter` (4.25)
- **author:** M. Borland, ANL/APS.

4.62 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[=count]]
  [-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[=count]` — Specifies that for any rows that are identical in the sort column values, only the first should be included in the output file. If the `count` qualifier is given, then a count of the number of identical rows is supplied in a column called IdenticalCount.
  - `-noWarnings` — Suppresses warning messages.

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

4.63 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`*`] [-nameParameter=`*`paramName`*`]`
  `[-firstPage=`*`number`*`] [-lastPage=`*`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.
  - `-nameParameter=`*`paramName`* — Specifies that instead of composing names for the output files, `sddssplit` take the names from the string parameter *paramName* in the input file. This provides a limited capability to retrieve the original files from a file made with `sddscombine`. Note that if the named parameter takes the same value on two pages, the file created for the first of the pages will be overwritten.
  - `-firstPage=`*`number`* — Specifies the first page of data to use.
  - `-lastPage=`*`number`* — Specifies the last page of data to use.
  - `-interval=`*`number`* — Specifies the interval between pages that are used.

- **see also:**
  - `sddsbreak` (4.10)
  - `sddscombine` (4.17)

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

4.64 sddstimeconvert

- **description:** `sddstimeconvert` does conversions between calendar time in terms of (for example) day, month, and year, and "time-since-epoch". The latter is the time in seconds since a system-defined reference time (e.g., 0:00 on January 1, 1970).

- **examples:** Convert column data broken down as day, month, year, and hour to seconds-since-epoch:

  ```
  sddstimeconvert input.sdds output.sdds
  -epoch=column,Time,year=TheYear,month=TheMonth,day=DayOfMonth,hour=HourOfDay
  ```

  where `TheYear`, `TheMonth`, `DayOfMonth`, and `HourOfDay` are the names of the columns in the input file and `Time` is the column to be created containing the time-since-epoch.

- **synopsis:**

  ```
  sddstimeconvert [inputFile] [outputFile] [-pipe[=input][,output]]
  [-breakdown={column | parameter},timeName[,year=newName]
     [,julianDay=newName][,month=newName][,day=newName][,hour=newName][,text=newName]]
  [-epoch={column | parameter},newName,year=name
      [,julianDay=name | month=name,day=name],hour=name]
  ```

- **switches:**
  - `-pipe[=input][,output]` — The standard SDDS Toolkit pipe option.
  - `-breakdown={column | parameter},`*`timeName`*`[,year=`*`newName`*`]`
    `[,julianDay=`*`newName`*`][,month=`*`newName`*`][,day=`*`newName`*`][,hour=`*`newName`*`]`
    `[,text=`*`newName`*`]` — Specifies conversion of the column or parameter data named *timeName* to year, Julian day, month, day, hour, and/or a text string. *timeName* contains the time expressed as seconds-since-epoch. Any number of these options may be given.
  - `-epoch={column | parameter},`*`newName`*`,year=`*`name`*`,`
    `[julianDay=`*`name`*` | month=`*`name`*`,day=`*`name`*`],hour=`*`name`* — Specifies conversion of column or parameter data given as year, Julian day or month/day, and hour to seconds-since-epoch, with the result being placed in *newName*.

- **notes:** The hour data as used or created by `sddstimeconvert` contains the floating-point time-of-day in hours. That is, the minutes and seconds are folded into the hour value.

  Year values must be the full four-digit year; e.g., year 99 is *not* 1999, but rather 99 AD.

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

4.65 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 for the output file are generated either from the data in a selected string column in the input file, from the value of the command line option -root, or from an internal default.

  The column names of the input file are collected and made into a string column in the output file.

- **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
  [-oldColumnNames=string] [{-root=string [-digits=integer] |
  -newColumnNames=column}] [-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.
  - `-oldColumnNames=`*`string`* — A string column of name *`string`* is created in the output file, containing the column names of the input files as string data. If this option is not present, then the default name of "OldColumnNames" is used for the string column.
  - `-root=`*`string`* — A string used to generate columns names for the output file data. The first data column is named "*`string`*000", the second, "*`string`*001", etc. If the input file has only one row, the the root name alone (with no digits following) is used for the column name.
  - `-digits=`*`integer`* — minimum number of digits used in the number appended to *`root`* of the output file column names. (Default value is 3).
  - `-newColumnNames=`*`string`* — Specifies a string column of the input file which will be used to define column names of the output file.
  - `-symbol=`*`string`* — The string for 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

4.66 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 [-pipe=[input][,output]] inputFile outputFile
-independentVariable=parametername [-columns=listOfNames]
[-excludeColumns=listOfNames] [-sigma[=generate]] [-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 and Index columns from the input file is transfered to the output file. In the column Index doesn't exist in the input file, then it is created in the output file anyway. 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=`*`listOfNames`* — columns to be individually paired with independentVariable for straight line fitting
  - `-excludeColumns=`*`listOfNames`* — 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

4.67 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]] [-fillIn]
[-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.
  - `-fillIn` — Normally if there is no match in *xRefFile* for a row in *input*, that row will not appear in *output*. If `-fillIn` is given, the row will appear, but the values for the columns that are being transferred from *xRefFile* will be filled with zeros and empty strings, as appropriate.
  - `-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)
  - sddsselect (4.57)

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

4.68 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:**
  - `sddsinterp` (4.39)

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

4.69 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` (4.50)
  - `sddsplot` (4.48)
  - `sddsconvert` (4.20)

4.70 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: `+ - * / sum`

    With the exception of `sum`, 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.
    `sum` is used to sum the top `n` items on the stack, exclusive of the top of the stack, which gives the number of items to sum. For example, `2 4 6 8 4 sum` would put the value `20` on 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 xt 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 gamP gamQ 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 $\mathrm{J_i(x)}$ onto the stack. Note that $\mathrm{Y_n(x)}$ 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]).

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

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

`erf` and `erfc` are the error function and complementary error function. By definition, $\mathrm{erf(x)} + \mathrm{erfc(x)}$ 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]:

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

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

`gamP` and `gamQ` are, respectively, the incomplete gamma function and its complement [7]:

$$\mathrm{gamP(a,x) = 1 - gamQ(a,x)} = \frac{1}{\Gamma(\mathrm{a})} \int_0^{\mathrm{x}} \mathrm{e^{-t}t^{a-1}dt} \qquad \mathrm{a} > 0$$

These functions take two arguments; the 'a' argument is place on the stack first.
`lngam` is the natural log of the gamma function. For integer arguments, `x lngam` is $\mathrm{ln((x-1)!)}$. The gamma function is defined as[7]:

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

* 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.
  * Examples:
    Suppose that a quantity is tested for its sign. If the sign is negative, then have the conditional return a -1, if the sign is positive then return a +1.
    Suppose we are running in the rpn shell and that the quantity 4 is initially pushed onto the stack. The command "`0 < ? -1 : 1 $`" that accomplishes the sign test will be executed as follows.

```
command      stack    logical stack
0            0        stack empty
             4

<            0        false  <-- new value
             4

? 1 : -1 $   1        stack empty
             0
             4
```

    In order to keep the stack small, the command could have been written "`0 < pop pop ? -1 : 1 $`", where the pop commands would eliminate the 0 and 4 from the stack before the conditional is executed.

If the command is executed with rpnl command in a C-shell, then the $ character has to be escaped with single quotes:

C-shell> rpnl “`4 0 < pop pop ? -1 : 1 "'$'`

If the command is executed in a C-shell sddsprocess command to create a new column, then we write:

```
sddsprocess <infile> <outfile> \
   -def=col,NewColumn,"OldColumn 0 < pop pop ? -1 : 1 "'$'
```

which is similar to the rpnl command above.
If the sddsprocess command is run in a tcl/tk shell, then the $ character is escaped with a backslash:

```
sddsprocess <infile> <outfile> \
   "-def=col,NewColumn,OldColumn 0 < pop pop ? -1 : 1 \$"
```

Note that the double quotes enclose the whole command argument, not just the sub-argument.

- **String operations**: `"" =str cshs format getformat pops scan sprf vstr xstr`

  * 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.
  * Other operations: `cshs xstr`
    `cshs` executes the top string of the stack in a C-shell subprocess; note that if the command requires terminal input, `rpn` will hang. `xstr` executes the top string of the stack as an rpn command.

- **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.

4.71 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` (4.12)
  - `sddsconvert` (4.20)
  - `sddscorrelate` (4.22)
  - `sddsenvelope` (4.28)
  - `sddsfft` (4.33)
  - `sddsoutlier` (4.45)
  - `sddsplot` (4.48)
  - `sddsprintout` (4.49)
  - `sddsprocess` (4.50)
  - `sddssmooth` (4.61)
  - `sddsxref` (4.67)
  - `sddszerofind` (4.68)

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] [-separatePages]
-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.
  - `-separatePages` — 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 superseded 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.

Contents