Application Programmer's Guide for SDDS Version 1.5

Michael Borland and Robert Soliday
Advanced Photon Source

June 29, 2006

SDDS is a file protocol for Self Describing Data Sets. This document describes Version 1.5 of the SDDS protocol and the function library that supports it. It is intended for those who wish to develop programs that use SDDS files.

1 Definition of SDDS Protocol

1.1 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 1.5.

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 1.5. 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.

1.1.1 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.

1.1.2 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
.
.
.
```

1.1.3 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 correponding 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
.
.
.
```

1.1.4 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
.
.
.
```

1.1.5 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:
.
.
.
```

1.1.6 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.

1.2 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.

1.3 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.

2 Overview of Library Routines

This section gives a brief description of the library routines grouped by functions. The reader should refer to the manual pages for more detailed information.

2.1 Input Routines

2.1.1 Setup

- `SDDS_InitializeInput` (4.63) — Opens a file and initializes the SDDS_TABLE structure used to access the data set.

- `SDDS_InitializeHeaderlessInput` (4.62) — Opens a file and initializes the SDDS_TABLE structure used to access the data set, but does not attempt to read the header, which is assumed not to exist. The caller must define the header using the SDDS_Define X routines. This method of using SDDS is *not* recommended, as it defeats the purpose of having a self-describing data set by imbedding the description in a program.

- SDDS_ReadTable (4.71) — Reads the next data table from a data set.

2.1.2 Determining File Contents

- SDDS_GetArrayInformation (4.37) — Returns information about the fields of an array definition. This is the preferred way to get this information.
- SDDS_GetColumnInformation (4.45) — Returns information about the fields of a column definition. This is the preferred way to get this information.
- SDDS_GetParameterInformation (4.53) — Returns information about the fields of a parameter definition. This is the preferred way to get this information.
- Other routines:
  - SDDS_GetArrayDefinition (4.35) — Returns a pointer to the ARRAY_DEFINITION structure for a named array.
  - SDDS_GetArrayNames (4.38) — Returns a pointer to an array of strings giving the names of all arrays.
  - SDDS_GetArrayIndex (4.36) — Returns the index of a named array in the table structure. This can be used for fast access to the data.
  - SDDS_GetArrayType (4.39) — Returns the data type of a named array in the table structure.
  - SDDS_GetColumnDefinition (4.41) — Returns a pointer to the COLUMN_DEFINITION structure for a named column.
  - SDDS_GetColumnNames (4.46) — Returns a pointer to an array of strings giving the names of all columns.
  - SDDS_GetColumnIndex (4.44) — Returns the index of a named column in the table structure. This can be used for fast access to the data.
  - SDDS_GetColumnType (4.47) — Returns the data type of a named column in the table structure.
  - SDDS_GetParameterDefinition (4.51) — Returns a pointer to the PARAMETER_DEFINITION structure for a named parameter.
  - SDDS_GetParameterNames (4.54) — Returns a pointer to an array of strings giving the names of all parameters.
  - SDDS_GetParameterIndex (4.52) — Returns the index of a named parameter in the table structure. This can be used for fast access to the data.
  - SDDS_GetParameterType (4.55) — Returns the data type of a named parameter in the table structure.
  - SDDS_VerifyArrayExists (4.88) — Returns the index of a named array if it exists as the specified data type.
  - SDDS_VerifyColumnExists (4.89) — Returns the index of a named column if it exists as the specified data type.
  - SDDS_VerifyParameterExists (4.90) — Returns the index of a named Parameter if it exists as the specified data type.

2.1.3 Column Selection

- `SDDS_SetColumnFlags` (4.77) — Sets all column-selection flags to a given value, indicating whether columns are initially "of interest" or not). By default, all columns are initially of interest.
- `SDDS_SetColumnsOfInterest` (4.80) — Modifies the "of interest" status of columns using lists of column names, wildcard matching, and so forth. General two-term logic is possible using the previous status of each column and the results of newly-requested matching.
- `SDDS_DeleteColumn` (4.26) — Deletes a named column from the current data table of a data set.
- `SDDS_DeleteUnsetColumns` (4.28) — Deletes all "unset" columns–i.e., those not declared as "of interest".

2.1.4 Row Selection

- `SDDS_SetRowFlags` (4.83) — Set all row-selection flags to a given value, indicating whether all rows are initially "of interest" or not. By default, all rows are initially of interest.
- `SDDS_SetRowsOfInterest` (4.84) — Modifies row-selection flags based on string matching to the data in a named column, with general two-term logic supported using the previous status of each row and the results of the newly-requested matching.
- `SDDS_FilterRowsOfInterest` (4.30) — Modifies row-selection flags based on whether the numeric entries in a named column are within a specified interval, with general two-term logic supported using the previous status of each row and the results of the newly-requested matching.
- `SDDS_CountRowsOfInterest` (4.16) — Counts the number of rows in the current data table that are currently "of interest", i.e., those with non-zero row-selection flags.
- `SDDS_DeleteUnsetRows` (4.29) — Deletes all rows of the current data table that are not currently of interest.

2.1.5 Access to Tabular Data

- `SDDS_GetColumn` (4.40) — Returns a pointer to an array of data values from a single column of the tabular portion of the current data table. Only rows that are "of interest" are returned.
- `SDDS_GetColumnInDoubles` (4.42) — Returns a pointer to an array of doubles containing the numeric data values (converted to double if necessary) from a single column of the tabular portion of the current data table. Only rows that are "of interest" are returned.
- `SDDS_GetMatrixFromColumn` (4.48) — Returns a pointer to a two-dimensional array of data values from a single column of the tabular portion of the current data table. Only rows that are "of interest" are included. The dimensions of the array are supplied by the caller (from parameters stored in the data set, for example).
- `SDDS_GetMatrixOfRows` (4.49) — Returns a pointer to a two-dimensional array of data values from the rows and columns "of interest". The columns are ordered as defined by calls to SDDS_SetColumnsOfInterest, and must have the same data type.

- SDDS_GetRow (4.56) — Returns a pointer to a single row containing data values from the columns "of interest". Only the rows "of interest" are returned. The columns are ordered as defined by calls to SDDS_SetColumnsOfInterest, and must have the same data type.
- SDDS_GetValue (4.58) — Returns a pointer to a single value from the tabular data. The data is requested by giving the column name and the row number. Only values from the rows "of interest" are returned.

2.1.6 Access to Array Data

- SDDS_GetArray (4.34) — Returns a pointer to an SDDS_ARRAY structure containing the current data and other information for a named array of a data set.

2.1.7 Access to Parameter Data

- SDDS_GetParameter (4.50) — Returns a pointer to the current value of a named parameter of a data set.

2.2 Output Routines

2.2.1 Setup

- SDDS_InitializeCopy (4.61) — Initializes a SDDS_TABLE structure in preparation for internally copying the current data table of another data set.
- SDDS_InitializeOutput (4.64) — Initializes a SDDS_TABLE structure in preparation for output of data to a data set. This includes opening the file but not writing the SDDS header, which is yet to be defined.
- SDDS_StartTable (4.86) — Signals the beginning of a new data table of a data set that is being created in a program. Usually calls to this routine are interleaved with calls to SDDS_WriteTable.
- SDDS_CopyTable (4.15) — Copies the current data table from one data set structure to another. The target data set is usually set up using SDDS_InitializeCopy.

2.2.2 Defining Data Set Elements

- SDDS_DefineArray (4.18) — Defines a new array for a data set.
- SDDS_DefineColumn (4.19) — Defines a new tabular-data column for a data set.
- SDDS_DefineParameter (4.20) — Defines a new data table parameter for a data set.

2.2.3 Defining Data Table Elements

- SDDS_SetArray (4.75) — Sets the values in an array for the current table of a data set.
- SDDS_SetParameters (4.81) — Sets the values of parameters for the current table of a data set.
- SDDS_SetRowValues (4.85) — Sets the values of tabular data for the current table of a data set.

2.2.4 Output

- `SDDS_WriteLayout` (4.91) — Writes the SDDS header to the file defined by a previous call to SDDS_InitializeOutput or SDDS_InitializeCopy. The contents of the header must have been previously defined by calls to the SDDS_Define X routines.
- `SDDS_WriteTable` (4.92) — Writes the current data table to the data set file. The contents of the table must have been previously defined by calls to SDDS_SetArray, SDDS_SetParameters, and SDDS_SetRowValues.

2.3 Error Handling and Utilities

- `SDDS_CastValue` (4.2) — Casts a numeric value from one SDDS data type to another.
- `SDDS_ConvertToDouble` (4.8) — Converts a numeric value from any SDDS numeric data type to double precision.
- `SDDS_GetTypeSize` (4.57) — Returns the size in bytes of a given SDDS data type.
- `SDDS_Logic` (4.66) — Performs general two-term logic for SDDS row and column selection operations.
- `SDDS_NumberOfErrors` (4.67) — Returns the number of errors recorded since the last call to SDDS_PrintErrors.
- `SDDS_PrintErrors` (4.69) — Prints the errors recorded since the last call to itself.
- `SDDS_PrintTypedValue` (4.70) — Prints any SDDS data type.
- `SDDS_Terminate` (4.87) — Terminates an SDDS data set, freeing all memory and closing the data set file.

3 Two Templates for SDDS Application Organization

This section gives pseudo-code templates for two typical SDDS applications. One shows how to access data stored in a SDDS file, while the other shows how to make an SDDS file from data generated internally (or read from another source).

3.1 Accessing Data Stored in an SDDS File

```
SDDS_TABLE SDDS_table;

/* open the file and read the SDDS header */
SDDS_InitializeInput(&SDDS_table, filename)

/* see if arrays that are needed are present */
if ((SDDS_CheckArray(&SDDS_table, array_I_need, NULL,
                     SDDS_ANY_NUMERIC_TYPE, stderr)) != SDDS_CHECK_OKAY) {
    fprintf(stderr, "array %s is not in the data file",
        array_I_need);
    exit(1);
```

```
}
...

/* see if parameters that are needed are present */
if ((SDDS_CheckParameter(&SDDS_table, parameter_I_need, NULL,
                         SDDS_ANY_NUMERIC_TYPE, stderr)) != SDDS_CHECK_OKAY) {
    fprintf(stderr, "parameter %s is not in the data file",
        parameter_I_need);
    exit(1);
}
...

/* see if columns that are needed are present */
if ((SDDS_CheckColumn(&SDDS_table, column_I_need, NULL,
                    SDDS_ANY_NUMERIC_TYPE, stderr)) != SDDS_CHECK_OKAY) {
    fprintf(stderr, "column %s is not in the data file",
        column_I_need);
    exit(1);
}
...

/* read and process each data table in the data set */
while (SDDS_ReadTable(&SDDS_table)>0) {
    /* set all rows and all columns to initially be "of interest" */
    SDDS_SetColumnFlags(&SDDS_table, 1);
    SDDS_SetRowFlags(&SDDS_table, 1);

    /* access array data */ SDDS_GetArray(...)

    /* access parameter data */ SDDS_GetParameter(...)

    /* optional row and column selection operations */
    SDDS_SetColumnsOfInterest(...);
    SDDS_SetRowsOfInterest(...);
    SDDS_FilterRowsOfInterest(...);

    /* access tabular data values */
    SDDS_GetValue(...)
    SDDS_GetRow(...)
    SDDS_GetColumn(...)
    ...
}
```

3.2 SDDS Output of Internally-Generated Data

```
SDDS_TABLE SDDS_table;

/* open the file and set a few data set properites */
```

```
SDDS_InitializeOutput(&SDDS_table, ...)

/* define columns */
SDDS_DefineColumn(&SDDS_table, column_name, ...);
...

/* define arrays */
SDDS_DefineArray(&SDDS_table, array_name, ...);
...

/* define parameters */
SDDS_DefineParameter(&SDDS_table, parameter_name, ...);
...

/* save the header */
SDDS_SaveLayout(&SDDS_table);

/* write the header */
SDDS_WriteLayout(&SDDS_table);

/* generate and output the data */
while (DataGenerated(my_data_structure)) {
    /* start a new SDDS data table */
    SDDS_StartTable(&SDDS_table, rows);

    /* put data into SDDS parameters and columns */
    SDDS_SetParameters(&SDDS_table, ...);
    ...
    SDDS_SetArray(&SDDS_table, ...);
    ...
    SDDS_SetColumn(&SDDS_table, ...);
    ...

    /* write the new data table */ SDDS_WriteTable(&SDDS_table);
}

SDDS_Terminate(&SDDS_table);
```

4 Manual Pages

4.1 SDDS_ArrayCount

- **name:**
  SDDS_ArrayCount
- **description:**
  Used to retrieve the number of arrays.

- **synopsis:** #include "SDDS.h"
long SDDS_ArrayCount(SDDS_TABLE *SDDS_table)
- **arguments:**
  - **SDDS_table:** Address of the SDDS_TABLE structure for the data set.
- **return value:**
Returns the number of arrays.
- **see also:**
  - `SDDS_ColumnCount` (4.7)
  - `SDDS_ParameterCount` (4.68)
  - `SDDS_RowCount` (4.73)

4.2 SDDS_CastValue

- **name:**
SDDS_CastValue
- **description:**
Casts a numeric value from one specified type (presumed to be the actual type) into another specified type.
- **synopsis:** #include "SDDS.h"
void *SDDS_CastValue(void *data, long index, long data_type, long desired_type, void *memory)
- **arguments:**
  - **data:** The reference address of the data to be converted.
  - **index:** The offset of the address of the item to be converted from the reference address, in units of the size of the declared type.
  - **data_type:** The declared type of the data. Must be one of the constants (defined in SDDS.h) SDDS_DOUBLE, SDDS_FLOAT, SDDS_LONG, SDDS_SHORT, or SDDS_CHARACTER.
  - **desired_type:** The desired type to cast the data to.
  - **memory:** The address of the location in which to store the result. If NULL, space is allocated.
- **return value:**
The address of the location in which the result was stored. Returns NULL on error and records an error message.
- **see also:**
  - `SDDS_ConvertToDouble` (4.8)
  - `SDDS_GetColumnInDoubles` (4.42)
  - `SDDS_PrintErrors` (4.69)
  - `SDDS_PrintTypedValue` (4.70)
  - `SDDS_NumberOfErrors` (4.67)

4.3 SDDS_CheckArray

- **name:**
  SDDS_CheckArray
- **description:**
  Check to see if an array exists with the specified name, units and type.
- **synopsis:** #include "SDDS.h"
  long SDDS_CheckArray(SDDS_TABLE *SDDS_table, char *name, char *units, long type, FILE *fp_message);
- **arguments:**
  - **SDDS_table:** Address of the SDDS_TABLE structure for the data set.
  - **name:** Name of array.
  - **units:** Units of array, may be NULL.
  - **type:** Valid types are SDDS_ANY_NUMERIC_TYPE, SDDS_ANY_FLOATING_TYPE, SDDS_ANY_INTEGER_TYPE, and 0. If 0 is used this is ignored.
  - **fp_message:** Error messages are sent here. Usually stderr.
- **return value:**
  Returns SDDS_CHECK_OK on success.
- **see also:**
  - `SDDS_CheckColumn` (4.4)
  - `SDDS_CheckParameter` (4.5)

4.4 SDDS_CheckColumn

- **name:**
  SDDS_CheckColumn
- **description:**
  Check to see if an column exists with the specified name, units and type.
- **synopsis:** #include "SDDS.h"
  long SDDS_CheckColumn(SDDS_TABLE *SDDS_table, char *name, char *units, long type, FILE *fp_message);
- **arguments:**
  - **SDDS_table:** Address of the SDDS_TABLE structure for the data set.
  - **name:** Name of column.
  - **units:** Units of column, may be NULL.
  - **type:** Valid types are SDDS_ANY_NUMERIC_TYPE, SDDS_ANY_FLOATING_TYPE, SDDS_ANY_INTEGER_TYPE, and 0. If 0 is used this is ignored.
  - **fp_message:** Error messages are sent here. Usually stderr.

- **return value:**
  Returns SDDS_CHECK_OK on success.

- **see also:**

  - `SDDS_CheckArray` (4.3)
  - `SDDS_CheckParameter` (4.5)

4.5 SDDS_CheckParameter

- **name:**
  SDDS_CheckParameter

- **description:**
  Check to see if an parameter exists with the specified name, units and type.

- **synopsis:** #include "SDDS.h"
  long SDDS_CheckParameter(SDDS_TABLE *SDDS_table, char *name, char *units, long type, FILE *fp_message);

- **arguments:**

  - **SDDS_table:** Address of the SDDS_TABLE structure for the data set.
  - **name:** Name of parameter.
  - **units:** Units of parameter, may be NULL.
  - **type:** Valid types are SDDS_ANY_NUMERIC_TYPE, SDDS_ANY_FLOATING_TYPE, SDDS_ANY_INTEGER_TYPE, and 0. If 0 is used this is ignored.
  - **fp_message:** Error messages are sent here. Usually stderr.

- **return value:**
  Returns SDDS_CHECK_OK on success.

- **see also:**

  - `SDDS_CheckArray` (4.3)
  - `SDDS_CheckColumn` (4.4)

4.6 SDDS_ClearErrors

- **name:**
  SDDS_ClearErrors

- **description:**
  Resets the error history.

- **synopsis:** #include "SDDS.h"
  void SDDS_ClearErrors(void)

- **arguments:**
  None.

- **return value:**
  None.
- **see also:**
  - `SDDS_PrintErrors` (4.69)
  - `SDDS_NumberOfErrors` (4.67)

4.7 SDDS_ColumnCount

- **name:**
  SDDS_ColumnCount
- **description:**
  Used to retrieve the number of columns.
- **synopsis:** #include "SDDS.h"
  long SDDS_ColumnCount(SDDS_TABLE *SDDS_table)
- **arguments:**
  - **SDDS_table:** Address of the SDDS_TABLE structure for the data set.
- **return value:**
  Returns the number of columns.
- **see also:**
  - `SDDS_ArrayCount` (4.1)
  - `SDDS_ParameterCount` (4.68)
  - `SDDS_RowCount` (4.73)

4.8 SDDS_ConvertToDouble

- **name:**
  SDDS_ConvertToDouble
- **description:**
  Converts an item of data from a specified type into a double precision value.
- **synopsis:** #include "SDDS.h"
  double SDDS_ConvertToDouble(long type, void *data, long index)
- **arguments:**
  - **type:** The type of the data. Must be one of the constants (defined in SDDS.h) SDDS_DOUBLE, SDDS_FLOAT, SDDS_LONG, SDDS_SHORT, or SDDS_CHARACTER.
  - **data:** The reference address of the data to be converted.
  - **index:** The offset of the address of the item to be converted from the reference address, in units of the size of the declared type.

- **return value:**
  The double precision value is returned. If an error occurs, zero is returned and an error message is recorded.

- **see also:**
  - `SDDS_CastValue` (4.2)
  - `SDDS_GetColumnInDoubles` (4.42)
  - `SDDS_PrintErrors` (4.69)
  - `SDDS_PrintTypedValue` (4.70)
  - `SDDS_NumberOfErrors` (4.67)

4.9 SDDS_CopyArrays

- **name:**
  SDDS_CopyArrays

- **description:**
  Copies array values from one structure into another.

- **synopsis:** #include "SDDS.h"
  long SDDS_CopyArrays(SDDS_TABLE *SDDS_target, SDDS_TABLE *SDDS_source)

- **arguments:**
  - **SDDS_target:** Address of SDDS_TABLE structure into which to copy data.
  - **SDDS_source:** Address of SDDS_TABLE structure from which to copy data.

- **return value:**
  Returns 1 on success. On failure, returns 0 and records an error message.

- **see also:**
  - `SDDS_CopyParameters` (4.12)
  - `SDDS_CopyColumns` (4.10)
  - `SDDS_CopyTable` (4.15)

4.10 SDDS_CopyColumns

- **name:**
  SDDS_CopyColumns

- **description:**
  Copies columns values from one structure into another.

- **synopsis:** #include "SDDS.h"
  long SDDS_CopyColumns(SDDS_TABLE *SDDS_target, SDDS_TABLE *SDDS_source)

- **arguments:**
  - **SDDS_target:** Address of SDDS_TABLE structure into which to copy data.

  - **SDDS_source:** Address of SDDS_TABLE structure from which to copy data.

- **return value:**
  Returns 1 on success. On failure, returns 0 and records an error message.

- **see also:**
  - `SDDS_CopyArrays` (4.9)
  - `SDDS_CopyParameters` (4.12)
  - `SDDS_CopyRowDirect` (4.13)
  - `SDDS_CopyRowsOfInterest` (4.14)
  - `SDDS_CopyTable` (4.15)

4.11 SDDS_CopyLayout

- **name:**
  SDDS_CopyLayout

- **description:**
  Copies the layout in an SDDS table from one structure into another.

- **synopsis:** #include "SDDS.h"
  long SDDS_CopyLayout(SDDS_TABLE *SDDS_target, SDDS_TABLE *SDDS_source)

- **arguments:**
  - **SDDS_target:** Address of SDDS_TABLE structure into which to copy data.
  - **SDDS_source:** Address of SDDS_TABLE structure from which to copy data.

- **return value:**
  Returns 1 on success. On failure, returns 0 and records an error message.

- **see also:**
  - `SDDS_CopyTable` (4.15)
  - `SDDS_InitializeCopy` (4.61)
  - `SDDS_NumberOfErrors` (4.67)
  - `SDDS_PrintErrors` (4.69)

4.12 SDDS_CopyParameters

- **name:**
  SDDS_CopyParameters

- **description:**
  Copies parameter values from one structure into another.

- **synopsis:** #include "SDDS.h"
  long SDDS_CopyParameters(SDDS_TABLE *SDDS_target, SDDS_TABLE *SDDS_source)

- **arguments:**

- **SDDS_target:** Address of SDDS_TABLE structure into which to copy data.
- **SDDS_source:** Address of SDDS_TABLE structure from which to copy data.

- **return value:**
Returns 1 on success. On failure, returns 0 and records an error message.

- **see also:**
  - `SDDS_CopyArrays` (4.9)
  - `SDDS_CopyColumns` (4.10)
  - `SDDS_CopyTable` (4.15)

4.13 SDDS_CopyRowDirect

- **name:**
SDDS_CopyRowDirect

- **description:**
Copies row values from one structure into another.

- **synopsis:** #include "SDDS.h"
long SDDS_CopyRowDirect(SDDS_TABLE *SDDS_target, long target_row, SDDS_TABLE *SDDS_source, long source_row)

- **arguments:**
  - **SDDS_target:** Address of SDDS_TABLE structure into which to copy data.
  - **target_row:** Row to place copied data.
  - **SDDS_source:** Address of SDDS_TABLE structure from which to copy data.
  - **source_row:** Row of data to copy.

- **return value:**
Returns 1 on success. On failure, returns 0 and records an error message.

- **see also:**
  - `SDDS_CopyColumns` (4.10)
  - `SDDS_CopyRowsOfInterest` (4.14)

4.14 SDDS_CopyRowsOfInterest

- **name:**
SDDS_CopyRowsOfInterest

- **description:**
Copies row values from one structure into another.

- **synopsis:** #include "SDDS.h"
long SDDS_CopyRowsOfInterest(SDDS_TABLE *SDDS_target, SDDS_TABLE *SDDS_source)

- **arguments:**
  - **SDDS_target:** Address of SDDS_TABLE structure into which to copy data.
  - **SDDS_source:** Address of SDDS_TABLE structure from which to copy data.
- **return value:**
  Returns 1 on success. On failure, returns 0 and records an error message.
- **see also:**
  - `SDDS_CopyColumns` (4.10)
  - `SDDS_CopyRowDirect` (4.13)

4.15 SDDS_CopyTable

- **name:**
  SDDS_CopyTable
- **description:**
  Copies the data in an SDDS table from one structure into another.
- **synopsis:** #include "SDDS.h"
  long SDDS_CopyTable(SDDS_TABLE *SDDS_target, SDDS_TABLE *SDDS_source)
- **arguments:**
  - **SDDS_target:** Address of SDDS_TABLE structure into which to copy data.
  - **SDDS_source:** Address of SDDS_TABLE structure from which to copy data.
- **return value:**
  Returns 1 on success. On failure, returns 0 and records an error message.
- **see also:**
  - `SDDS_CopyLayout` (4.11)
  - `SDDS_InitializeCopy` (4.61)
  - `SDDS_NumberOfErrors` (4.67)
  - `SDDS_PrintErrors` (4.69)

4.16 SDDS_CountRowsOfInterest

- **name:**
  SDDS_CountRowsOfInterest
- **description:**
  Counts and returns the number of rows marked as "of interest" in a table of data.
- **synopsis:** #include "SDDS.h"
  long SDDS_CountRowsOfInterest(SDDS_TABLE *SDDS_table);
- **arguments:**

  - **SDDS_table:** Address of the SDDS_TABLE structure for the data set.

- **return value:**
  Returns the number of rows for which the acceptance flags are non-zero. On error, returns -1.

- **see also:**

  - `SDDS_DeleteUnsetRows` (4.29)
  - `SDDS_FilterRowsOfInterest` (4.30)
  - `SDDS_GetMatrixOfRows` (4.49)
  - `SDDS_GetRow` (4.56)
  - `SDDS_SetRowFlags` (4.83)
  - `SDDS_SetRowsOfInterest` (4.84)

4.17 SDDS_DisconnectFile

- **name:**
  SDDS_DisconnectFile

- **description:**
  Allows "temporarily" closing a file. Updates the present page and flushes the table.

- **synopsis:** #include "SDDS.h"
  long SDDS_DisconnectFile(SDDS_TABLE *SDDS_table)

- **arguments:**

  - **SDDS_table:** Address of the SDDS_TABLE structure for the data set.

- **return value:**
  Returns 1 on success. On failure, returns 0 and records an error message.

- **see also:**

  - `SDDS_ReconnectFile` (4.72)

4.18 SDDS_DefineArray

- **name:**
  SDDS_DefineArray

- **description:**
  Processes a definition of a data array.

- **synopsis:** #include "SDDS.h"
  long SDDS_DefineArray(SDDS_TABLE *SDDS_table, char *name, char *symbol, char *units, char *description, char *format_string, long type, long field_length, long dimensions, char *group_name)

- **arguments:**

- **SDDS_table:** Address of the SDDS_TABLE structure for the data set.
- **name:** A NULL-terminated character string giving the name of the array. Must be supplied.
- **symbol:** A NULL-terminated character string giving the symbol to be used to represent the array. If none is desired, pass NULL.
- **units:** A NULL-terminated character string giving the units of the array. If none is desired, pass NULL.
- **description:** A NULL-terminated character string giving a description of the array. If none is desired, pass NULL.
- **format_string:** A NULL-terminated character string giving a printf format string to be used to print the data for ASCII output. If NULL is passed, a reasonable default will be chosen.
- **type:** An integer value specifying the data type of the array. Must be one of the following constants, defined in SDDS.h: SDDS_DOUBLE, SDDS_FLOAT, SDDS_LONG, SDDS_SHORT, SDDS_CHARACTER, or SDDS_STRING.
- **field_length:** An integer value giving the length of the field allotted to the array for ASCII output. Ignored if zero. If negative, the field length is the absolute value, but leading and trailing white-space are eliminated from data of type SDDS_STRING upon readin.
- **dimensions:** An integer value giving the number of dimensions of the array. Must be greater than 0.
- **group_name:** A NULL-terminated character string giving the name of the array group to which the array belongs. This mechanism allows the user to indicate that different arrays are related (e.g., parallel to one another).

- **return value:**
  On success, returns the index of the newly-defined array. Returns -1 on failure and records an error message.

- **see also:**
  - `SDDS_GetArrayDefinition` (4.35)
  - `SDDS_GetArrayInformation` (4.37)
  - `SDDS_GetArrayNames` (4.38)
  - `SDDS_NumberOfErrors` (4.67)
  - `SDDS_PrintErrors` (4.69)

4.19 SDDS_DefineColumn

- **name:**
  SDDS_DefineColumn

- **description:**
  Processes a definition of a data column.

- **synopsis:** #include "SDDS.h"
  long SDDS_DefineColumn(SDDS_TABLE *SDDS_table, char *name, char *symbol, char *units, char *description, char *format_string, long type, long field_length)
- **arguments:**
  - **SDDS_table:** Address of the SDDS_TABLE structure for the data set.
  - **name:** A NULL-terminated character string giving the name of the column. Must be supplied.
  - **symbol:** A NULL-terminated character string giving the symbol to be used to represent the column. If none is desired, pass NULL.
  - **units:** A NULL-terminated character string giving the units of the column. If none is desired, pass NULL.
  - **description:** A NULL-terminated character string giving a description of the column. If none is desired, pass NULL.
  - **format_string:** A NULL-terminated character string giving a printf format string to be used to print the data for ASCII output. If NULL is passed, a reasonable default will be chosen.
  - **type:** An integer value specifying the data type of the column. Must be one of the following constants, defined in SDDS.h: SDDS_DOUBLE, SDDS_FLOAT, SDDS_LONG, SDDS_SHORT, SDDS_CHARACTER, or SDDS_STRING.
  - **field_length:** An integer value given the length of the field allotted to the column for ASCII output. Ignored if zero. If negative, the field length is the absolute value, but leading and trailing white-space are eliminated from data of type SDDS_STRING upon readin.
- **return value:**
  On success, returns the index of the newly-defined column. Returns -1 on failure and records an error message.
- **see also:**
  - `SDDS_DefineSimpleColumn` (4.22)
  - `SDDS_DefineSimpleColumns` (4.23)
  - `SDDS_GetColumnDefinition` (4.41)
  - `SDDS_GetColumnInformation` (4.45)
  - `SDDS_GetColumnNames` (4.46)
  - `SDDS_NumberOfErrors` (4.67)
  - `SDDS_PrintErrors` (4.69)

4.20 SDDS_DefineParameter

- **name:**
  SDDS_DefineParameter
- **description:**
  Processes a definition of a data parameter.

- **synopsis:** #include "SDDS.h"
  long SDDS_DefineParameter(SDDS_TABLE *SDDS_table, char *name, char *symbol, char *units, char *description, char *format_string, long type, char *fixed_value)
- **arguments:**
  - **SDDS_table:** Address of the SDDS_TABLE structure for the data set.
  - **name:** A NULL-terminated character string giving the name of the parameter. Must be supplied.
  - **symbol:** A NULL-terminated character string giving the symbol to be used to represent the parameter. If none is desired, pass NULL.
  - **units:** A NULL-terminated character string giving the units of the parameter. If none is desired, pass NULL.
  - **description:** A NULL-terminated character string giving a description of the parameter. If none is desired, pass NULL.
  - **format_string:** A NULL-terminated character string giving a printf format string to be used to print the data for ASCII output. If NULL is passed, a reasonable default will be chosen.
  - **type:** An integer value specifying the data type of the parameter. Must be one of the following constants, defined in SDDS.h: SDDS_DOUBLE, SDDS_FLOAT, SDDS_LONG, SDDS_SHORT, SDDS_CHARACTER, or SDDS_STRING.
  - **fixed_value:** A NULL-terminated character string giving the permanent value of the parameter. For data that is not of type SDDS_STRING, the string should be prepared using, for example, sprintf.
- **return value:**
  On success, returns the index of the newly-defined parameter. Returns -1 on failure and records an error message.
- **see also:**
  - `SDDS_DefineParameter1` (4.21)
  - `SDDS_DefineSimpleParameter` (4.24)
  - `SDDS_DefineSimpleParameters` (4.25)
  - `SDDS_GetParameterDefinition` (4.51)
  - `SDDS_GetParameterInformation` (4.53)
  - `SDDS_GetParameterNames` (4.54)
  - `SDDS_NumberOfErrors` (4.67)
  - `SDDS_PrintErrors` (4.69)

4.21 SDDS_DefineParameter1

- **name:**
  SDDS_DefineParameter1
- **description:**
  Processes a definition of a data parameter.

- **synopsis:** #include "SDDS.h"
  long SDDS_DefineParameter1(SDDS_TABLE *SDDS_table, char *name, char *symbol, char *units, char *description, char *format_string, long type, void *fixed_value)

- **arguments:**
  - **SDDS_table:** Address of the SDDS_TABLE structure for the data set.
  - **name:** A NULL-terminated character string giving the name of the parameter. Must be supplied.
  - **symbol:** A NULL-terminated character string giving the symbol to be used to represent the parameter. If none is desired, pass NULL.
  - **units:** A NULL-terminated character string giving the units of the parameter. If none is desired, pass NULL.
  - **description:** A NULL-terminated character string giving a description of the parameter. If none is desired, pass NULL.
  - **format_string:** A NULL-terminated character string giving a printf format string to be used to print the data for ASCII output. If NULL is passed, a reasonable default will be chosen.
  - **type:** An integer value specifying the data type of the parameter. Must be one of the following constants, defined in SDDS.h: SDDS_DOUBLE, SDDS_FLOAT, SDDS_LONG, SDDS_SHORT, SDDS_CHARACTER, or SDDS_STRING.
  - **fixed_value:** A pointer to a numerical variable giving the permanent value of the parameter.

- **return value:**
  On success, returns the index of the newly-defined parameter. Returns -1 on failure and records an error message.

- **see also:**
  - `SDDS_DefineParameter` (4.20)
  - `SDDS_DefineSimpleParameter` (4.24)
  - `SDDS_DefineSimpleParameters` (4.25)
  - `SDDS_GetParameterDefinition` (4.51)
  - `SDDS_GetParameterInformation` (4.53)
  - `SDDS_GetParameterNames` (4.54)
  - `SDDS_NumberOfErrors` (4.67)
  - `SDDS_PrintErrors` (4.69)

4.22 SDDS_DefineSimpleColumn

- **name:**
  SDDS_DefineSimpleColumn

- **description:**
  Processes a definition of a data column.

- **synopsis:** #include "SDDS.h"
long SDDS_DefineSimpleColumn(SDDS_TABLE *SDDS_table, char *name, char *units, long type)

- **arguments:**
  - **SDDS_table:** Address of the SDDS_TABLE structure for the data set.
  - **name:** A NULL-terminated character string giving the name of the column. Must be supplied.
  - **units:** A NULL-terminated character string giving the units of the column. If none is desired, pass NULL.
  - **type:** An integer value specifying the data type of the column. Must be one of the following constants, defined in SDDS.h: SDDS_DOUBLE, SDDS_FLOAT, SDDS_LONG, SDDS_SHORT, SDDS_CHARACTER, or SDDS_STRING.

- **return value:**
Returns 1 on success. On failure, returns 0 and records an error message.

- **see also:**
  - `SDDS_DefineColumn` (4.19)
  - `SDDS_DefineSimpleColumns` (4.23)
  - `SDDS_GetColumnDefinition` (4.41)
  - `SDDS_GetColumnInformation` (4.45)
  - `SDDS_GetColumnNames` (4.46)
  - `SDDS_NumberOfErrors` (4.67)
  - `SDDS_PrintErrors` (4.69)

4.23 SDDS_DefineSimpleColumns

- **name:**
SDDS_DefineSimpleColumns

- **description:**
Processes a definition of multiple data columns.

- **synopsis:** #include "SDDS.h"
long SDDS_DefineColumns(SDDS_TABLE *SDDS_table, long number, char **name, char **units, long type)

- **arguments:**
  - **SDDS_table:** Address of the SDDS_TABLE structure for the data set.
  - **number:** The number of columns to be defined.
  - **name:** An array of NULL-terminated character strings giving the names of the columns. Must be supplied.
  - **units:** An array of NULL-terminated character strings giving the units of the columns. If none is desired, pass NULL.

- **type:** An integer value specifying the data type of the columns. Must be one of the following constants, defined in SDDS.h: SDDS_DOUBLE, SDDS_FLOAT, SDDS_LONG, SDDS_SHORT, SDDS_CHARACTER, or SDDS_STRING.

- **return value:**
  Returns 1 on success. On failure, returns 0 and records an error message.

- **see also:**
  - `SDDS_DefineColumn` (4.19)
  - `SDDS_DefineSimpleColumn` (4.22)
  - `SDDS_GetColumnDefinition` (4.41)
  - `SDDS_GetColumnInformation` (4.45)
  - `SDDS_GetColumnNames` (4.46)
  - `SDDS_NumberOfErrors` (4.67)
  - `SDDS_PrintErrors` (4.69)

4.24 SDDS_DefineSimpleParameter

- **name:**
  SDDS_DefineSimpleParameter

- **description:**
  Processes a definition of a data parameter.

- **synopsis:** #include "SDDS.h"
  long SDDS_DefineSimpleParameter(SDDS_TABLE *SDDS_table, char *name, char *units, long type)

- **arguments:**
  - **SDDS_table:** Address of the SDDS_TABLE structure for the data set.
  - **name:** A NULL-terminated character string giving the name of the parameter. Must be supplied.
  - **units:** A NULL-terminated character string giving the units of the parameter. If none is desired, pass NULL.
  - **type:** An integer value specifying the data type of the parameter. Must be one of the following constants, defined in SDDS.h: SDDS_DOUBLE, SDDS_FLOAT, SDDS_LONG, SDDS_SHORT, SDDS_CHARACTER, or SDDS_STRING.

- **return value:**
  Returns 1 on success. On failure, returns 0 and records an error message.

- **see also:**
  - `SDDS_DefineParameter` (4.20)
  - `SDDS_DefineParameter1` (4.21)
  - `SDDS_DefineSimpleParameters` (4.25)

- SDDS_GetParameterDefinition (4.51)
- SDDS_GetParameterInformation (4.53)
- SDDS_GetParameterNames (4.54)
- SDDS_NumberOfErrors (4.67)
- SDDS_PrintErrors (4.69)

4.25 SDDS_DefineSimpleParameters

- **name:**
SDDS_DefineSimpleParameters
- **description:**
Processes a definition of multiple data parameters.
- **synopsis:** #include "SDDS.h"
long SDDS_DefineSimpleParameters(SDDS_TABLE *SDDS_table, long number, char **name, char **units, long type)
- **arguments:**
  - **SDDS_table:** Address of the SDDS_TABLE structure for the data set.
  - **number:** The number of parameters to be defined.
  - **name:** An array of NULL-terminated character strings giving the names of the parameters. Must be supplied.
  - **units:** An array of NULL-terminated character strings giving the units of the parameters. If none is desired, pass NULL.
  - **type:** An integer value specifying the data type of the parameter. Must be one of the following constants, defined in SDDS.h: SDDS_DOUBLE, SDDS_FLOAT, SDDS_LONG, SDDS_SHORT, SDDS_CHARACTER, or SDDS_STRING.
- **return value:**
Returns 1 on success. On failure, returns 0 and records an error message.
- **see also:**
  - SDDS_DefineParameter (4.20)
  - SDDS_DefineParameter1 (4.21)
  - SDDS_DefineSimpleParameter (4.24)
  - SDDS_GetParameterDefinition (4.51)
  - SDDS_GetParameterInformation (4.53)
  - SDDS_GetParameterNames (4.54)
  - SDDS_NumberOfErrors (4.67)
  - SDDS_PrintErrors (4.69)

4.26 SDDS_DeleteColumn

- **name:**
  SDDS_DeleteColumn
- **description:**
  Deletes a named column from the current data table of a data set. If another data table is subsequently read in, the column reappears in the new data table.
- **synopsis:** #include "SDDS.h"
  long SDDS_DeleteColumn(SDDS_TABLE *SDDS_table, char *column_name)
- **arguments:**
  - **SDDS_table:** Address of the SDDS_TABLE structure for the data set.
  - **column_name:** A NULL-terminated character string giving the name of the column to delete.
- **return value:**
  Returns 1 on success. Returns 0 on failure, and records an error message.
- **see also:**
  - `SDDS_DeleteParameter` (4.27)
  - `SDDS_DeleteUnsetColumns` (4.28)
  - `SDDS_SetColumnFlags` (4.77)
  - `SDDS_SetColumnsOfInterest` (4.80)
  - `SDDS_NumberOfErrors` (4.67)
  - `SDDS_PrintErrors` (4.69)

4.27 SDDS_DeleteParameter

- **name:**
  SDDS_DeleteParameter
- **description:**
  Deletes a named parameter from the current data table of a data set. If another data table is subsequently read in, the parameter reappears in the new data table.
- **synopsis:** #include "SDDS.h"
  long SDDS_DeleteParameter(SDDS_TABLE *SDDS_table, char *parameter_name)
- **arguments:**
  - **SDDS_table:** Address of the SDDS_TABLE structure for the data set.
  - **parameter_name:** A NULL-terminated character string giving the name of the parameter to delete.
- **return value:**
  Returns 1 on success. Returns 0 on failure, and records an error message.
- **see also:**
  - `SDDS_DeleteColumn` (4.26)

4.28 SDDS_DeleteUnsetColumns

- **name:**
  SDDS_DeleteUnsetColumns
- **description:**
  Deletes "unset" columns from the current data table of a data set. An "unset" column is one which has not been declared to be "of interest" using SDDS_SetColumnsOfInterest.
- **synopsis:** #include "SDDS.h"
  long SDDS_DeleteUnsetColumns(SDDS_TABLE *SDDS_table)
- **arguments:**
  - **SDDS_table:** Address of the SDDS_TABLE structure for the data set.
- **return value:**
  Returns 1 on success. On failure, returns 0 and records an error message.
- **see also:**
  - `SDDS_SetColumnFlags` (4.77)
  - `SDDS_SetColumnsOfInterest` (4.80)
  - `SDDS_NumberOfErrors` (4.67)
  - `SDDS_PrintErrors` (4.69)

4.29 SDDS_DeleteUnsetRows

- **name:**
  SDDS_DeleteUnsetRows
- **description:**
  Deletes "unset" rows from the current data table of a data set. An "unset" row is one which has not been declared to be "of interest" using SDDS_SetRowsOfInterest or SDDS_FilterRowsOfInterest.
- **synopsis:** #include "SDDS.h"
  long SDDS_DeleteUnsetRows(SDDS_TABLE *SDDS_table)
- **arguments:**
  - **SDDS_table:** Address of the SDDS_TABLE structure for the data set.
- **return value:**
  Returns 1 on success. On failure, returns 0 and records an error message.
- **see also:**
  - `SDDS_FilterRowsOfInterest` (4.30)
  - `SDDS_NumberOfErrors` (4.67)
  - `SDDS_PrintErrors` (4.69)
  - `SDDS_SetRowFlags` (4.83)
  - `SDDS_SetRowsOfInterest` (4.84)

4.30 SDDS_FilterRowsOfInterest

- **name:**
  SDDS_FilterRowsOfInterest
- **description:**
  Sets which rows of a data table from a data set are "of interest".
- **synopsis:** #include "SDDS.h"
  long SDDS_FilterRowsOfInterest(SDDS_TABLE *SDDS_table, char *filter_column, double lower, double upper, long logic)
- **arguments:**
  - **SDDS_table:** Address of the SDDS_TABLE structure for the data set.
  - **filter_column:** A NULL-terminated character string giving the name of a column, the values in which will be used to filter the rows of the table.
  - **lower, upper:** The lower and upper limits of the filter window. A value inside the filter is said to "match" the filter.
  - **logic:** A word of bit flags indicating how to combine the previous status of each row with the acceptance status based on matching to the matching_string. See the manual entry for SDDS_Logic for details.
- **return value:**
  On success, returns the number of rows that are currently accepted. On failure, returns -1 and records an error message.

4.31 SDDS_FindArray

- **name:**
  SDDS_FindArray
- **description:**
  Determine if an array exists.
- **synopsis:** #include "SDDS.h"
  char *SDDS_FindArray(SDDS_TABLE *SDDS_table, long mode, ...);
- **arguments:**
  - **SDDS_table:** Address of the SDDS_TABLE structure for the data set.
  - **mode:** Valid modes are FIND_SPECIFIED_TYPE, FIND_ANY_TYPE, FIND_NUMERIC_TYPE, FIND_FLOATING_TYPE, FIND_INTEGER_TYPE
  - **...:** If mode is FIND_SPECIFIED_TYPE it expects an SDDS data type followed by a list of array names followed by NULL. Otherwise it expects a list of array names followed by NULL.
- **return value:**
  On success it returns the name of the last array specified. On error it returns a NULL and records an error message.

- **see also:**
  - `SDDS_FindColumn` (4.32)
  - `SDDS_FindParameter` (4.33)

4.32 SDDS_FindColumn

- **name:**
  SDDS_FindColumn
- **description:**
  Determine if a column exists.
- **synopsis:** #include "SDDS.h"
  char *SDDS_FindColumn(SDDS_TABLE *SDDS_table, long mode, ...);
- **arguments:**
  - **SDDS_table:** Address of the SDDS_TABLE structure for the data set.
  - **mode:** Valid modes are FIND_SPECIFIED_TYPE, FIND_ANY_TYPE, FIND_NUMERIC_TYPE, FIND_FLOATING_TYPE, FIND_INTEGER_TYPE
  - **...:** If mode is FIND_SPECIFIED_TYPE it expects an SDDS data type followed by a list of column names followed by NULL. Otherwise it expects a list of column names followed by NULL.
- **return value:**
  On success it returns the name of the last column specified. On error it returns a NULL and records an error message.
- **see also:**
  - `SDDS_FindArray` (4.31)
  - `SDDS_FindParameter` (4.33)

4.33 SDDS_FindParameter

- **name:**
  SDDS_FindParameter
- **description:**
  Determine if a parameter exists.
- **synopsis:** #include "SDDS.h"
  char *SDDS_FindParameter(SDDS_TABLE *SDDS_table, long mode, ...);
- **arguments:**
  - **SDDS_table:** Address of the SDDS_TABLE structure for the data set.
  - **mode:** Valid modes are FIND_SPECIFIED_TYPE, FIND_ANY_TYPE, FIND_NUMERIC_TYPE, FIND_FLOATING_TYPE, FIND_INTEGER_TYPE

- **...:** If mode is FIND_SPECIFIED_TYPE it expects an SDDS data type followed by a list of parameter names followed by NULL. Otherwise it expects a list of parameter names followed by NULL.

- **return value:**
  On success it returns the name of the last parameter specified. On error it returns a NULL and records an error message.

- **see also:**
  - `SDDS_FindArray` (4.31)
  - `SDDS_FindColumn` (4.32)

4.34 SDDS_GetArray

- **name:**
  SDDS_GetArray

- **description:**
  Returns a pointer to a structure containing the data and other information about an array in the current data table of a data set.

- **synopsis:** #include "SDDS.h"
  SDDS_ARRAY *SDDS_GetArray(SDDS_TABLE *SDDS_table, char *array_name, SDDS_ARRAY *memory);

```
typedef struct {
    /* pointer to the internally-stored structure: */
    ARRAY\_DEFINITION *definition;
    long *dimension;            /* sizes for each dimension */
    long elements;              /* total number of elements */
    /* Address of new copy of array of data values, stored contiguously: */
    void *data;
    /* Address of n-dimensional pointer array into the new copy of data: */
    void *pointer;
} SDDS\_ARRAY;
```

- **arguments:**
  - **SDDS_table:** Address of the SDDS_TABLE structure for the data set.
  - **array_name:** A NULL-terminated character string giving the name of the array to return.
  - **memory:** Address of a SDDS_ARRAY structure into which to place the information. Preferred usage is to pass back a pointer that was originally returned by SDDS_GetArray. Otherwise, the user must initialize the pointers in the structure to NULL.

- **return value:**
  On success, returns a pointer to a SDDS_ARRAY structure. The contents of the structure point to newly-created copies of the information, with the exception of the definition field,

which points to the internal copy. If memory is not NULL, the storage pointed to by the data and pointer fields of the structure *memory will be reused. The data pointer, should be cast to type *data-type**, where *data-type* is the data type of the array. The preferred method is to use the pointer pointer, which should be cast to type *data-type**n, where *n symbolizes a string of n asterisks, n being the number of dimensions of the array. Note that for type SDDS_STRING, *data-type* is char *.

On failure, returns NULL and records an error message.

- **see also:**
  - `SDDS_NumberOfErrors` (4.67)
  - `SDDS_PrintErrors` (4.69)
  - `SDDS_SetArray` (4.75)

4.35 SDDS_GetArrayDefinition

- **name:**
  SDDS_GetArrayDefinition
- **description:**
  Returns a pointer to a structure describing a named data array.
- **synopsis:** #include "SDDS.h"
  ARRAY_DEFINITION *SDDS_GetArrayDefinition(SDDS_TABLE *SDDS_table, char *name)

```
typedef struct {
    char *name, *symbol, *units, *description, *format\_string;
    long type, field\_length, dimensions;
    /* internal data may follow, and should not be accessed by users */
} ARRAY\_DEFINITION;
```

- **arguments:**
  - **SDDS_table:** Address of the SDDS_TABLE structure for the data set.
  - **name:** A NULL-terminated character string giving the name of the array for which the definition is desired.
- **return value:**
  On success, returns the address of the internally-stored structure containing the definition of the array. On failure, returns NULL and records an error message.
- **see also:**
  - `SDDS_GetArrayInformation` (4.37)
  - `SDDS_GetArrayNames` (4.38)
  - `SDDS_NumberOfErrors` (4.67)
  - `SDDS_PrintErrors` (4.69)

4.36 SDDS_GetArrayIndex

- **name:**
  SDDS_GetArrayIndex
- **description:**
  Returns the index of a named array in the data set. This is used with some routines for faster access to the data or to information about the data.
- **synopsis:** #include "SDDS.h"
  long SDDS_GetArrayIndex(SDDS_TABLE *SDDS_table, char *array_name)
- **arguments:**
  - **SDDS_table:** Address of the SDDS_TABLE structure for the data set.
  - **array_name:** A NULL-terminated character string giving the name of the array for which information is desired.
- **return value:**
  On success, returns a non-negative integer giving the index of the array. On failure, returns -1 and records an error message.
- **see also:**
  - `SDDS_GetArrayDefinition` (4.35)
  - `SDDS_GetArrayInformation` (4.37)
  - `SDDS_GetArrayNames` (4.38)
  - `SDDS_NumberOfErrors` (4.67)
  - `SDDS_PrintErrors` (4.69)

4.37 SDDS_GetArrayInformation

- **name:**
  SDDS_GetArrayInformation
- **description:**
  Returns information about a specified array. This routine is the preferred alternative to SDDS_GetArrayDefinition.
- **synopsis:** #include "SDDS.h"
  long SDDS_GetArrayInformation(SDDS_TABLE *SDDS_table, char *field_name, void *memory, long mode, name-or-index)
- **arguments:**
  - **SDDS_table:** Address of the SDDS_TABLE structure for the data set.
  - **field_name:** A NULL terminated string giving the name of the field that information is requested for. These field names are identical to the names used in the namelist commands.

- **memory:** The address in which to place the information. Should have type *data-type** in the user's program, where *data-type* is the data type of the information. Note that for STRING information, *data-type* is char *. If memory is NULL, the existence and type of the information will be verified; the type will be returned as usual.
- **mode:** One of the following constants (defined in SDDS.h):
  * SDDS_GET_BY_NAME - Indicates that the next argument is a NULL-terminated character string containing the name of the array for which information is desired.
  * SDDS_GET_BY_INDEX - Indicates that the next argument is a non-negative integer giving the index of the array for which information is desired. This index is obtained from SDDS_DefineArray or SDDS_GetArrayIndex.

- **return value:**
  On success, returns the SDDS data type of the information. This is one of the following constants (defined in SDDS.h): SDDS_DOUBLE, SDDS_FLOAT, SDDS_LONG, SDDS_SHORT, SDDS_CHARACTER, or SDDS_STRING.

  On failure, returns zero and records an error message.

- **see also:**
  - `SDDS_GetArrayDefinition` (4.35)
  - `SDDS_GetArrayIndex` (4.36)
  - `SDDS_NumberOfErrors` (4.67)
  - `SDDS_PrintErrors` (4.69)
  - `SDDS_VerifyArrayExists` (4.88)

4.38 SDDS_GetArrayNames

- **name:**
  SDDS_GetArrayNames

- **description:**
  Returns an array of character strings giving the names of the arrays for a data set.

- **synopsis:** #include "SDDS.h"
  char **SDDS_GetArrayNames(SDDS_TABLE *SDDS_table, long *number)

- **arguments:**
  - **SDDS_table:** Address of the SDDS_TABLE structure for the data set.
  - **number:** Address of a location in which to place the number of arrays.

- **return value:**
  On success, returns an array of NULL-terminated character strings giving the names of the arrays. On failure, returns NULL and records an error message.

- **see also:**
  - `SDDS_GetArrayDefinition` (4.35)

- SDDS_GetArrayInformation (4.37)
- SDDS_NumberOfErrors (4.67)
- SDDS_PrintErrors (4.69)

4.39 SDDS_GetArrayType

- **name:**
  SDDS_GetArrayType
- **description:**
  Returns the SDDS data type of a named array in the data set.
- **synopsis:** #include "SDDS.h"
  long SDDS_GetArrayType(SDDS_TABLE *SDDS_table, long array_index)
- **arguments:**
  - **SDDS_table:** Address of the SDDS_TABLE structure for the data set.
  - **array_index:** The integer index of the array in the data set. Returned by SDDS_DefineArray or obtained from SDDS_GetArrayIndex.
- **return value:**
  On success, returns a non-negative integer giving the SDDS data type; this will be one of the constants (defined in SDDS.h) SDDS_DOUBLE, SDDS_FLOAT, SDDS_LONG, SDDS_SHORT, SDDS_CHARACTER, or SDDS_STRING.

  On failure, returns -1 and records an error message.

4.40 SDDS_GetColumn

- **name:**
  SDDS_GetColumn
- **description:**
  Returns a pointer to the data for a named column for the current data table of a data set.
- **synopsis:** #include "SDDS.h"
  void *SDDS_GetColumn(SDDS_TABLE *SDDS_table, char *column_name);
- **arguments:**
  - **SDDS_table:** Address of the SDDS_TABLE structure for the data set.
  - **column_name:** A NULL-terminated character string giving the name of the column to return.
- **return value:**
  On success, returns the address of a newly-created copy of the "rows of interest" from the named column. This array has type data-type*, where data-type is the data type of the column. Note that for type SDDS_STRING, data-type is char *. The number of rows is available from SDDS_CountRowsOfInterest.

  On failure, returns NULL and records an error message.

4.41 SDDS_GetColumnDefinition

- **name:**
SDDS_GetColumnDefinition

- **description:**
Returns a pointer to a structure describing a named data column.

- **synopsis:** #include "SDDS.h"
COLUMN_DEFINITION *SDDS_GetColumnDefinition(SDDS_TABLE *SDDS_table, char *name)

```
typedef struct {
    char *name, *symbol, *units, *description, *format\_string;
    long type, field\_length;
    /* internal data follows that should not be accessed by users */
} COLUMN\_DEFINITION;
```

- **arguments:**
  - **SDDS_table:** Address of the SDDS_TABLE structure for the data set.
  - **name:** A NULL-terminated character string giving the name of the column for which the definition is desired.

- **return value:**
On success, returns the address of the internally-stored structure containing the definition of the column. On failure, returns NULL and records an error message.

- **see also:**
  - `SDDS_GetColumnInformation` (4.45)
  - `SDDS_GetColumnNames` (4.46)
  - `SDDS_NumberOfErrors` (4.67)
  - `SDDS_PrintErrors` (4.69)

4.42 SDDS_GetColumnInDoubles

- **name:**
SDDS_GetColumnInDoubles

- **description:**
Returns an array of double-precision values containing the data in a specified column, provided the column contains numerical data. Integer data types are properly cast to double precision.

- **synopsis:** #include "SDDS.h"
double *SDDS_GetColumnInDoubles(SDDS_TABLE *SDDS_table, char *column_name)

- **arguments:**
  - **SDDS_table:** Address of the SDDS_TABLE structure for the data set.

  - **column_name:** A NULL-terminated character string giving the name of the column for which data is desired.

- **return value:**
  On success, returns the address of a newly-allocated array of double's containing the data. On failure, returns NULL and records an error message.

- **see also:**

  - `SDDS_CastValue` (4.2)
  - `SDDS_ConvertToDouble` (4.8)
  - `SDDS_GetColumn` (4.40)
  - `SDDS_GetColumnInLong` (4.43)
  - `SDDS_NumberOfErrors` (4.67)
  - `SDDS_PrintErrors` (4.69)
  - `SDDS_PrintTypedValue` (4.70)

4.43 SDDS_GetColumnInLong

- **name:**
  SDDS_GetColumnInLong

- **description:**
  Returns an array of long-precision values containing the data in a specified column, provided the column contains numerical data. Float data types are properly cast to long precision.

- **synopsis:** #include "SDDS.h"
  long *SDDS_GetColumnInLong(SDDS_TABLE *SDDS_table, char *column_name)

- **arguments:**

  - **SDDS_table:** Address of the SDDS_TABLE structure for the data set.
  - **column_name:** A NULL-terminated character string giving the name of the column for which data is desired.

- **return value:**
  On success, returns the address of a newly-allocated array of long's containing the data. On failure, returns NULL and records an error message.

- **see also:**

  - `SDDS_CastValue` (4.2)
  - `SDDS_ConvertToDouble` (4.8)
  - `SDDS_GetColumn` (4.40)
  - `SDDS_GetColumnInDoubles` (4.42)
  - `SDDS_NumberOfErrors` (4.67)
  - `SDDS_PrintErrors` (4.69)
  - `SDDS_PrintTypedValue` (4.70)

4.44 SDDS_GetColumnIndex

- **name:**
  SDDS_GetColumnIndex
- **description:**
  Returns the index of a named column in the data set. This is used with some routines for faster access to the data or to information about the data.
- **synopsis:** #include "SDDS.h"
  long SDDS_GetColumnIndex(SDDS_TABLE *SDDS_table, char *column_name)
- **arguments:**
  - **SDDS_table:** Address of the SDDS_TABLE structure for the data set.
  - **column_name:** A NULL-terminated character string giving the name of the column for which information is desired.
- **return value:**
  On success, returns a non-negative integer giving the index of the column. On failure, returns -1 and records an error message.
- **see also:**
  - `SDDS_GetColumnDefinition` (4.41)
  - `SDDS_GetColumnInformation` (4.45)
  - `SDDS_GetColumnNames` (4.46)
  - `SDDS_NumberOfErrors` (4.67)
  - `SDDS_PrintErrors` (4.69)

4.45 SDDS_GetColumnInformation

- **name:**
  SDDS_GetColumnInformation
- **description:**
  Returns information about a specified column. This routine is the preferred alternative to SDDS_GetColumnDefinition.
- **synopsis:** #include "SDDS.h"
  long SDDS_GetColumnInformation(SDDS_TABLE *SDDS_table, char *field_name, void *memory, long mode, name-or-index)
- **arguments:**
  - **SDDS_table:** Address of the SDDS_TABLE structure for the data set.
  - **field_name:** A NULL terminated string giving the name of the field that information is requested for. These field names are identical to the names used in the namelist commands.

- **memory:** The address in which to place the information. Should have type *data-type** in the user's program, where *data-type* is the data type of the information. Note that for STRING information, *data-type* is char *. If memory is NULL, the existence and type of the information will be verified; the type will be returned as usual.
- **mode:** One of the following constants (defined in SDDS.h):
  * SDDS_GET_BY_NAME - Indicates that the next argument is a NULL-terminated character string containing the name of the column for which information is desired.
  * SDDS_GET_BY_INDEX - Indicates that the next argument is a non-negative integer giving the index of the column for which information is desired. This index is obtained from SDDS_DefineColumn or SDDS_GetColumnIndex.

- **return value:**
  On success, returns the SDDS data type of the information. This is one of the following constants (defined in SDDS.h): SDDS_DOUBLE, SDDS_FLOAT, SDDS_LONG, SDDS_SHORT, SDDS_CHARACTER, or SDDS_STRING.

  On failure, returns zero and records an error message.

- **see also:**
  - `SDDS_GetColumnDefinition` (4.41)
  - `SDDS_GetColumnIndex` (4.44)
  - `SDDS_NumberOfErrors` (4.67)
  - `SDDS_PrintErrors` (4.69)
  - `SDDS_VerifyColumnExists` (4.89)

4.46 SDDS_GetColumnNames

- **name:**
  SDDS_GetColumnNames

- **description:**
  Returns an array of character strings giving the names of the columns for a data set.

- **synopsis:** #include "SDDS.h"
  char **SDDS_GetColumnNames(SDDS_TABLE *SDDS_table, long *number)

- **arguments:**
  - **SDDS_table:** Address of the SDDS_TABLE structure for the data set.
  - **number:** Address of a location in which to place the number of columns.

- **return value:**
  On success, returns a pointer to an array of NULL-terminated character strings giving the names of the columns. On failure, returns NULL and records an error message.

- **see also:**
  - `SDDS_GetColumnDefinition` (4.41)

- SDDS_GetColumnInformation (4.45)
- SDDS_NumberOfErrors (4.67)
- SDDS_PrintErrors (4.69)

4.47 SDDS_GetColumnType

- **name:**
SDDS_GetColumnType
- **description:**
Returns the SDDS data type of a named column in the data set.
- **synopsis:** #include "SDDS.h"
long SDDS_GetColumnType(SDDS_TABLE *SDDS_table, long column_index)
- **arguments:**
  - **SDDS_table:** Address of the SDDS_TABLE structure for the data set.
  - **column_index:** The integer index of the column in the data set. Returned by SDDS_DefineColumn or obtained from SDDS_GetColumnIndex.
- **return value:**
On success, returns a non-negative integer giving the SDDS data type; this will be one of the constants (defined in SDDS.h) SDDS_DOUBLE, SDDS_FLOAT, SDDS_LONG, SDDS_SHORT, SDDS_CHARACTER, or SDDS_STRING. On failure, returns -1 and records an error message.
- **see also:**
  - SDDS_DefineColumn (4.19)
  - SDDS_GetColumnDefinition (4.41)
  - SDDS_GetColumnIndex (4.44)
  - SDDS_GetColumnInformation (4.45)
  - SDDS_GetColumnNames (4.46)
  - SDDS_NumberOfErrors (4.67)
  - SDDS_PrintErrors (4.69)

4.48 SDDS_GetMatrixFromColumn

- **name:**
SDDS_GetMatrixFromColumn
- **description:**
Returns a column of data arranged as a two-dimensional array.
- **synopsis:** #include "SDDS.h"
void *SDDS_GetMatrixFromColumn(SDDS_TABLE *SDDS_table, char *column_name, long dimension1, long dimension2, long mode)
- **arguments:**

- **SDDS_table:** Address of the SDDS_TABLE structure for the data set.
- **column_name:** A NULL-terminated character string giving the name of the column of data to retrieve.
- **dimension1, dimension2:** The desired dimensions of the array, the product of which must equal the number of rows in the column. These dimensions are typically stored in parameters of the data table.
- **mode:** One of the constants SDDS_ROW_MAJOR_DATA and SDDS_COLUMN_MAJOR_DATA, defined in SDDS.h. These indicate that the items in the column should be considered to be given in row-major or column-major order, respectively.

- **return value:**
  On success, returns a pointer to the array, which has type *data-type***, where *data-type* is the data type of the column. The array is newly-allocated, and may be modified as needed by the caller. Note that for type SDDS_STRING, *data-type* is char **.

  On failure, returns NULL and records an error message

- **see also:**
  - `SDDS_GetMatrixOfRows` (4.49)
  - `SDDS_GetColumn` (4.40)
  - `SDDS_NumberOfErrors` (4.67)
  - `SDDS_PrintErrors` (4.69)

4.49 SDDS_GetMatrixOfRows

- **name:**
  SDDS_GetMatrixOfRows

- **description:**
  Returns a two-dimensional array of the rows of interest from the columns of interest, as specified by such functions as SDDS_SetRowsOfInterest and SDDS_SetColumnsOfInterest. All selected columns must have the same data type.

- **synopsis:** #include "SDDS.h"
  void *SDDS_GetMatrixOfRows(SDDS_TABLE *SDDS_table, long *n_rows)

- **arguments:**
  - **SDDS_table:** Address of the SDDS_TABLE structure for the data set.
  - **n_rows:** The address of a location into which the number of rows in the array is written.

- **return value:**
  On success, returns a pointer to the array, which has type *data-type***, where *data-type* is the data type of the columns. The array is newly-allocated, and may be modified as needed by the caller. Note that for type SDDS_STRING, *data-type* is char *.

  On failure, returns NULL and records an error message.

- **see also:**
  - `SDDS_NumberOfErrors` (4.67)
  - `SDDS_PrintErrors` (4.69)
  - `SDDS_SetColumnsOfInterest` (4.80)
  - `SDDS_SetRowsOfInterest` (4.84)

4.50 SDDS_GetParameter

- **name:**
  SDDS_GetParameter
- **description:**
  Returns the value of a named parameter of the current data table of a data set.
- **synopsis:** #include "SDDS.h"
  void *SDDS_GetParameter(SDDS_TABLE *SDDS_table, char *parameter_name, void *memory)
- **arguments:**
  - **SDDS_table:** Address of the SDDS_TABLE structure for the data set.
  - **parameter_name:** A NULL-terminated character string giving the name of the parameter for which the value is desired.
  - **memory:** Pointer to memory in which to place the value of the parameter. If NULL, memory is allocated.
- **return value:**
  On success, returns a pointer to the location in which the data has been placed. The pointer should be cast to *data-type**, where *data-type* is the data type of the parameter. Note that for type SDDS_STRING, *data-type* is char *. This data is newly-allocated, and may be modified as needed by the caller. On failure, returns NULL and records an error message.
- **see also:**
  - `SDDS_NumberOfErrors` (4.67)
  - `SDDS_PrintErrors` (4.69)

4.51 SDDS_GetParameterDefinition

- **name:**
  SDDS_GetParameterDefinition
- **description:**
  Returns a pointer to a structure describing a named data parameter.
- **synopsis:** #include "SDDS.h"
  PARAMETER_DEFINITION *SDDS_GetParameterDefinition(SDDS_TABLE *SDDS_table, char *name);

```
typedef struct {
    char *name, *symbol, *units, *description, *format\_string, *fixed\_value;
    long type;
    /* internal data follows that should not be accessed by users */
} PARAMETER\_DEFINITION;
```

- **arguments:**
  - **SDDS_table:** Address of the SDDS_TABLE structure for the data set.
  - **name:** A NULL-terminated character string giving the name of the parameter for which the definition is desired.
- **return value:**
  On success, returns the address of the internally-stored structure containing the definition of the parameter. On failure, returns NULL and records an error message.
- **see also:**
  - `SDDS_GetParameterInformation` (4.53)
  - `SDDS_GetParameterNames` (4.54)
  - `SDDS_NumberOfErrors` (4.67)
  - `SDDS_PrintErrors` (4.69)

4.52 SDDS_GetParameterIndex

- **name:**
  SDDS_GetParameterIndex
- **description:**
  Returns the index of a named parameter in the data set. This is used with some routines for faster access to the data or to information about the data.
- **synopsis:** #include "SDDS.h"
  long SDDS_GetParameterIndex(SDDS_TABLE *SDDS_table, char *parameter_name)
- **arguments:**
  - **SDDS_table:** Address of the SDDS_TABLE structure for the data set.
  - **parameter_name:** A NULL-terminated character string giving the name of the parameter for which information is desired.
- **return value:**
  On success, returns a non-negative integer giving the index of the parameter. On failure, returns -1 and records an error message.
- **see also:**
  - `SDDS_GetParameterDefinition` (4.51)
  - `SDDS_GetParameterInformation` (4.53)
  - `SDDS_GetParameterNames` (4.54)
  - `SDDS_NumberOfErrors` (4.67)
  - `SDDS_PrintErrors` (4.69)

4.53 SDDS_GetParameterInformation

- **name:**
  SDDS_GetParameterInformation

- **description:**
  Returns information about a specified parameter. This routine is the preferred alternative to SDDS_GetParameterDefinition.

- **synopsis:** #include "SDDS.h"
  long SDDS_GetParameterInformation(SDDS_TABLE *SDDS_table, char *field_name, void *memory, long mode, name-or-index)

- **arguments:**
  - **SDDS_table:** Address of the SDDS_TABLE structure for the data set.
  - **field_name:** A NULL terminated string giving the name of the field that information is requested for. These field names are identical to the names used in the namelist commands.
  - **memory:** The address in which to place the information. Should have type *data-type** in the user's program, where *data-type* is the data type of the information. Note that for STRING information, *data-type* is char *. If memory is NULL, the existence and type of the information will be verified; the type will be returned as usual.
  - **mode:** One of the following constants (defined in SDDS.h):
    * SDDS_GET_BY_NAME - Indicates that the next argument is a NULL-terminated character string containing the name of the parameter for which information is desired.
    * SDDS_GET_BY_INDEX - Indicates that the next argument is a non-negative integer giving the index of the parameter for which information is desired. This index is obtained from SDDS_DefineParameter or SDDS_GetParameterIndex.

- **return value:**
  On success, returns the SDDS data type of the information. This is one of the following constants (defined in SDDS.h): SDDS_DOUBLE, SDDS_FLOAT, SDDS_LONG, SDDS_SHORT, SDDS_CHARACTER, or SDDS_STRING.

  On failure, returns zero and records an error message.

- **see also:**
  - `SDDS_GetParameterDefinition` (4.51)
  - `SDDS_GetParameterIndex` (4.52)
  - `SDDS_NumberOfErrors` (4.67)
  - `SDDS_PrintErrors` (4.69)
  - `SDDS_VerifyParameterExists` (4.90)

4.54 SDDS_GetParameterNames

- **name:**
SDDS_GetParameterNames

- **description:**
Returns an array of character strings giving the names of the parameters for a data set.

- **synopsis:** #include "SDDS.h"
char **SDDS_GetParameterNames(SDDS_TABLE *SDDS_table, long *number)

- **arguments:**
  - **SDDS_table:** Address of the SDDS_TABLE structure for the data set.
  - **number:** Address of a location in which to place the number of parameters.

- **return value:**
On success, returns a pointer to an array of NULL-terminated character strings giving the names of the parameters. On failure, returns NULL and records an error message.

- **see also:**
  - `SDDS_GetParameterDefinition` (4.51)
  - `SDDS_GetParameterInformation` (4.53)
  - `SDDS_NumberOfErrors` (4.67)
  - `SDDS_PrintErrors` (4.69)

4.55 SDDS_GetParameterType

- **name:**
SDDS_GetParameterType

- **description:**
Returns the SDDS data type of a named parameter in the data set.

- **synopsis:** #include "SDDS.h"
long SDDS_GetParameterType(SDDS_TABLE *SDDS_table, long parameter_index)

- **arguments:**
  - **SDDS_table:** Address of the SDDS_TABLE structure for the data set.
  - **parameter_index:** The integer index of the parameter in the data set. Returned by SDDS_DefineParameter or obtained from SDDS_GetParameterIndex.

- **return value:**
On success, returns a non-negative integer giving the SDDS data type; this will be one of the constants (defined in SDDS.h) SDDS_DOUBLE, SDDS_FLOAT, SDDS_LONG, SDDS_SHORT, SDDS_CHARACTER, or SDDS_STRING. On failure, returns -1 and records an error message.

- **see also:**

- SDDS_DefineParameter (4.20)
- SDDS_GetParameterDefinition (4.51)
- SDDS_GetParameterIndex (4.52)
- SDDS_GetParameterInformation (4.53)
- SDDS_GetParameterNames (4.54)
- SDDS_NumberOfErrors (4.67)
- SDDS_PrintErrors (4.69)

4.56 SDDS_GetRow

- **name:**
  SDDS_GetRow
- **description:**
  Returns a pointer to a specified row of homogeneous data values from a table. The columns included are those declared to be "of interest" by calling, for example, SDDS_SetColumnsOfInterest. The rows available are those declared to be of interest by calling, for example, SDDS_SetRowsOfInterest.
- **synopsis:** #include "SDDS.h"
  void *SDDS_GetRow(SDDS_TABLE *SDDS_table, long srow_index, void *memory);
- **arguments:**
  - **SDDS_table:** Address of the SDDS_TABLE structure for the data set.
  - **srow_index:** Index of the row desired in the list of selected rows (or "rows of interest").
  - **memory:** Address of location into which to write the data values. If NULL, memory is allocated for this purpose.
- **return value:**
  On success, returns the address of the first item of the row. This pointer should be cast to *data-type**, where *data-type* is the data type of the columns selected. Note that for type SDDS_STRING, *data-type* is char *. The data is stored in newly-allocated memory (or in the memory passed by the caller), and may be modified as needed by the caller.

  On failure, returns NULL and records an error message.

4.57 SDDS_GetTypeSize

- **name:**
  SDDS_GetTypeSize
- **description:**
  Returns the size of the data type in bytes.
- **synopsis:** #include "SDDS.h"
  long SDDS_GetTypeSize(long sdds_type);
- **arguments:**

- **sdds_type:** The SDDS data type for which the size is wanted. Must be one of the constants (defined in SDDS.h) SDDS_DOUBLE, SDDS_FLOAT, SDDS_LONG, SDDS_SHORT, SDDS_CHARACTER, or SDDS_STRING

- **return value:**
  On success, returns a positive integer giving the size of the data type in bytes. On failure, returns -1 and records an error message.

4.58 SDDS_GetValue

- **name:**
  SDDS_GetValue

- **description:**
  Returns a pointer to the data in a specified column and row of the current data table of a data set. Only the rows that are "of interest" are returned.

- **synopsis:** #include "SDDS.h"
  void *SDDS_GetValue(SDDS_TABLE *SDDS_table, char *column_name, long srow_index, void *memory);

- **arguments:**
  - **SDDS_table:** Address of the SDDS_TABLE structure for the data set.
  - **column_name:** A NULL-terminated string giving the name of the column for which data is desired.
  - **srow_index:** Index of the row desired in the list of selected rows (or "rows of interest").
  - **memory:** Address of location into which to write the data. If NULL, memory is allocated for this purpose.

- **return value:**
  On success, returns the address into which the data was placed. This pointer should be cast to *data-type**, where *data-type* is the data type of the columns selected. Note that for type SDDS_STRING, *data-type* is char *.

  On failure, returns NULL and records an error message.

- **see also:**
  - `SDDS_GetMatrixOfRows` (4.49)
  - `SDDS_GetRow` (4.56)
  - `SDDS_NumberOfErrors` (4.67)
  - `SDDS_PrintErrors` (4.69)
  - `SDDS_SetRowsOfInterest` (4.84)

4.59 SDDS_InitializeAppend

- **name:**
  SDDS_InitializeAppend
- **description:**
  Appends to a file by adding a new page.
- **synopsis:** #include "SDDS.h"
  long SDDS_InitializeAppend(SDDS_TABLE *SDDS_table, char *filename);
- **arguments:**
  - **SDDS_table:** Address of the SDDS_TABLE structure for the data set.
  - **filename:** A NULL-terminated character string giving the name of the existing file.
- **return value:**
  Returns 1 on success. On failure, returns 0 and records an error message.
- **see also:**
  - `SDDS_InitializeAppendToPage` (4.60)

4.60 SDDS_InitializeAppendToPage

- **name:**
  SDDS_InitializeAppendToPage
- **description:**
  Appends to a file by adding to the last page.
- **synopsis:** #include "SDDS.h"
  long SDDS_InitializeAppendToPage(SDDS_TABLE *SDDS_table, char *filename, long expected_n_rows, long *rowsPresentReturn);
- **arguments:**
  - **SDDS_table:** Address of the SDDS_TABLE structure for the data set.
  - **filename:** A NULL-terminated character string giving the name of the existing file.
  - **expected_n_rows:** The expected number of rows in the data table, used to preallocate memory for storing data values.
  - **rowsPresentReturn:** Pointer that returns the number or rows in the file.
- **return value:**
  Returns 1 on success. On failure, returns 0 and records an error message.
- **see also:**
  - `SDDS_InitializeAppend` (4.59)

4.61 SDDS_InitializeCopy

- **name:**
  SDDS_InitializeCopy
- **description:**
  Initializes a SDDS_TABLE structure in preparation for copying a data table from another SDDS_TABLE structure.
- **synopsis:** #include "SDDS.h"
  long SDDS_InitializeCopy(SDDS_TABLE *SDDS_target, SDDS_TABLE *SDDS_source, char *filename, char *filemode);
- **arguments:**
  - **SDDS_target:** Address of SDDS_TABLE structure into which to copy data.
  - **SDDS_source:** Address of SDDS_TABLE structure from which to copy data.
  - **filename:** A NULL-terminated character string giving a filename to be associated with the new SDDS_TABLE. Typically, the name of a file to which the copied data will be written after modification. Ignored if NULL.
  - **filemode:** A NULL-terminated character string giving the fopen file mode to be used to open the file named by filename. Ignored if filename is NULL.
- **return value:**
  Returns 1 on success. On failure, returns 0 and records an error message.
- **see also:**
  - `SDDS_CopyTable` (4.15)
  - `SDDS_NumberOfErrors` (4.67)
  - `SDDS_PrintErrors` (4.69)

4.62 SDDS_InitializeHeaderlessInput

- **name:**
  SDDS_InitializeHeaderlessInput
- **description:**
  Initializes a SDDS_TABLE structure for use in reading data from a SDDS file. This involves opening the file but not reading a header. It is assumed that the file contains no header, and that the caller will set up the SDDS_TABLE structure using calls to the SDDS_Define X routines. This is way of using SDDS is *not* recommended, as it defeats the purpose of having a self-describing file by imbedding the description in a program.
- **synopsis:** #include "SDDS.h"
  long SDDS_InitializeHeaderlessInput(SDDS_TABLE *SDDS_table, char *filename);
- **arguments:**
  - **SDDS_table:** Address of the SDDS_TABLE structure for the data set.
  - **filename:** A NULL-terminated character string giving the name of the data file.

- **return value:**
  Returns 1 on success. On failure, returns 0 and records an error message.
- **see also:**
  - `SDDS_DefineArray` (4.18)
  - `SDDS_DefineColumn` (4.19)
  - `SDDS_DefineParameter` (4.20)
  - `SDDS_ReadTable` (4.71)
  - `SDDS_NumberOfErrors` (4.67)
  - `SDDS_PrintErrors` (4.69)
  - `SDDS_InitializeInput` (4.63)

4.63 SDDS_InitializeInput

- **name:**
  SDDS_InitializeInput
- **description:**
  Initializes a SDDS_TABLE structure for use in reading data from a SDDS file. This involves opening the file and reading the SDDS header.
- **synopsis:** #include "SDDS.h"
  long SDDS_InitializeInput(SDDS_TABLE *SDDS_table, char *filename);
- **arguments:**
  - **SDDS_table:** Address of the SDDS_TABLE structure for the data set.
  - **filename:** A NULL-terminated character string giving the name of the file to set up for input.
- **return value:**
  Returns 1 on success. On failure, returns 0 and records an error message.
- **see also:**
  - `SDDS_InitializeHeaderlessInput` (4.62)
  - `SDDS_NumberOfErrors` (4.67)
  - `SDDS_PrintErrors` (4.69)
  - `SDDS_ReadTable` (4.71)

4.64 SDDS_InitializeOutput

- **name:**
  SDDS_InitializeOutput
- **description:**
  Initializes a SDDS_TABLE structure for use writing data to a SDDS file. This involves opening zeroing all elements of the structure and opening the file.

- **synopsis:** #include "SDDS.h"
  long SDDS_InitializeOutput(SDDS_TABLE *SDDS_table, long data_mode, long lines_per_row, char *description, char *contents, char *filename)
- **arguments:**
  - **SDDS_table:** Address of the SDDS_TABLE structure for the data set.
  - **data_mode:** On of the constants SDDS_ASCII or SDDS_BINARY, defined in SDDS.h.
  - **lines_per_row:** A positive integer giving the number of lines to use for each row of data in ASCII output mode.
  - **description:** A NULL-terminated character string informally describing the data set.
  - **contents:** A NULL-terminated character string formally specifying the data set. Users are advised to set and/or examine this value in order to uniquely identify different types of data sets, using a set of description keywords chosen by the applications programmer.
  - **filename:** A NULL-terminated character string giving the name of the file to set up for output.
- **return value:**
  Returns 1 on success. On failure, returns 0 and records an error message.
- **see also:**
  - `SDDS_DefineParameter` (4.20)
  - `SDDS_DefineColumn` (4.19)
  - `SDDS_WriteLayout` (4.91)
  - `SDDS_WriteTable` (4.92)
  - `SDDS_NumberOfErrors` (4.67)
  - `SDDS_PrintErrors` (4.69)

4.65 SDDS_LengthenTable

- **name:**
  SDDS_LengthenTable
- **description:**
  Increasing the number of allocated rows.
- **synopsis:** #include "SDDS.h"
  long SDDS_LengthenTable(SDDS_TABLE *SDDS_table, long n_additional_rows);
- **arguments:**
  - **SDDS_table:** Address of the SDDS_TABLE structure for the data set.
  - **n_additional_rows:** Number of additional rows.
- **return value:**
  Returns 1 on success. On failure, returns 0 and records an error message.
- **see also:**
  - `SDDS_InitializeOutput` (4.64)

4.66 SDDS_Logic

- **name:**
  SDDS_Logic
- **description:**
  This manual page describes both the SDDS_Logic function (which is rarely used by applications) and, more importantly, the method by which selection logic is applied for SDDS_SetColumnsOfInterest and SDDS_SetRowsOfInterest.
- **synopsis:** #include "SDDS.h"
  long SDDS_Logic(long previous, long match, unsigned long logic);
- **arguments:**
  - **previous:** A logical value resulting from a previous logical operation or from an initial condition. As usual in C, non-zero is true and zero is false.
  - **match:** A logical value resulting from a new matching or selection operation, e.g., matching to a wildcard string or comparing to a numerical selection window.
  - **logic:** A bitwise combination of the following constants, defined in SDDS.h:
    * SDDS_NEGATE_MATCH: Indicates that the logical value of match should be negated (or inverted). This would correspond, for example, to asking that a string *not* match a wildcard sequence or that a value be *outside* a numerical selection window.
    * SDDS_NEGATE_PREVIOUS: Indicates that the logical value of previous should be negated (or inverted).
    * SDDS_AND, SDDS_OR: One or fewer of these may be specified. If neither is specified, the value of previous is ignored. Otherwise, the indicated logical operation is performed with the previous and match values, subject to prior negation as per the last two items.
    * SDDS_NEGATE_EXPRESSION: Indicates that the resultant logical value of the previous three steps should be negated.
- **return value:**
  Returns 1 one if the result is true, 0 if false.
- **see also:**
  - `SDDS_SetColumnFlags` (4.77)
  - `SDDS_SetColumnsOfInterest` (4.80)
  - `SDDS_SetRowFlags` (4.83)
  - `SDDS_SetRowsOfInterest` (4.84)

4.67 SDDS_NumberOfErrors

- **name:**
  SDDS_NumberOfErrors
- **description:**
  Returns the number of errors recorded by SDDS library routines.

- **synopsis:** #include "SDDS.h"
  long SDDS_NumberOfErrors()
- **arguments:**
  None
- **return value:**
  The number of errors recorded by SDDS library routines since the last call to SDDS_PrintErrors.
- **see also:**
  - `SDDS_ClearErrors` (4.6)
  - `SDDS_PrintErrors` (4.69)

4.68 SDDS_ParameterCount

- **name:**
  SDDS_ParameterCount
- **description:**
  Used to retrieve the number of parameters.
- **synopsis:** #include "SDDS.h"
  long SDDS_ParameterCount(SDDS_TABLE *SDDS_table)
- **arguments:**
  - **SDDS_table:** Address of the SDDS_TABLE structure for the data set.
- **return value:**
  Returns the number of parameters.
- **see also:**
  - `SDDS_ArrayCount` (4.1)
  - `SDDS_ColumnCount` (4.7)
  - `SDDS_RowCount` (4.73)

4.69 SDDS_PrintErrors

- **name:**
  SDDS_PrintErrors
- **description:**
  Prints the error history to a specified FILE stream. When an error is encountered by SDDS library routines, most record an error message that may be retrieved with this facility.
- **synopsis:** #include "SDDS.h"
  void SDDS_PrintErrors(FILE *fp, long mode)
- **arguments:**

  - **fp:** The stream to which to print the error history.
  - **mode:** A flag word specifying the actions of SDDS_PrintErrors. It is composed by or'ing together any of the following constants (defined in SDDS.h):
    * SDDS_VERBOSE_PrintErrors - Specifies that all errors will be printed out, rather than just the first error.
    * SDDS_EXIT_PrintErrors - Specifies that, if there are errors, the program should exit to the shell after printing the errors.

- **return value:**
  None

- **see also:**
  - `SDDS_ClearErrors` (4.6)
  - `SDDS_NumberOfErrors` (4.67)

4.70 SDDS_PrintTypedValue

- **name:**
  SDDS_PrintTypedValue

- **description:**
  Prints a data value of a specified type using an optional printf format string.

- **synopsis:** #include "SDDS.h"
  long SDDS_PrintTypedValue(void *data, long index, long type, char *format, FILE *fp)

- **arguments:**
  - **data:** The reference address of the data to be printed.
  - **index:** The offset of the address of the item to be printed from the reference address, in units of the size of the declared type.
  - **type:** The type of the data, specified by one of the constants (defined in SDDS.h) SDDS_DOUBLE, SDDS_FLOAT, SDDS_LONG, SDDS_SHORT, or SDDS_CHARACTER.
  - **format:** A NULL-terminated character string giving a printf format specification for printing the data. If NULL, a reasonable default is chosen.
  - **fp:** The FILE stream to which to print the data.

- **return value:**
  Returns 1 on success. On failure, returns 0 and records an error message.

- **see also:**
  - `SDDS_CastValue` (4.2)
  - `SDDS_ConvertToDouble` (4.8)
  - `SDDS_GetColumnInDoubles` (4.42)
  - `SDDS_NumberOfErrors` (4.67)
  - `SDDS_PrintErrors` (4.69)

4.71 SDDS_ReadTable

- **name:**
  SDDS_ReadTable
- **description:**
  Reads a new data table from a data set. The data set must have previously been initialized using SDDS_InitializeInput.
- **synopsis:** #include "SDDS.h"
  long SDDS_ReadTable(SDDS_TABLE *SDDS_table);
- **arguments:**
  - **SDDS_table:** Address of the SDDS_TABLE structure for the data set.
- **return value:**
  On success, returns a positive integer giving the "table number" being returned. This is a value that is 1 for the first table and increments by 1 for each subsequent table. -1 is returned if there are no tables remaining in the data set.

  On failure, returns 0 and records an error message.
- **see also:**
  - `SDDS_InitializeInput` (4.63)
  - `SDDS_NumberOfErrors` (4.67)
  - `SDDS_PrintErrors` (4.69)

4.72 SDDS_ReconnectFile

- **name:**
  SDDS_ReconnectFile
- **description:**
  Opens a file closed by SDDS_Disconnect to the previous position.
- **synopsis:** #include "SDDS.h"
  long SDDS_ReconnectFile(SDDS_TABLE *SDDS_table)
- **arguments:**
  - **SDDS_table:** Address of the SDDS_TABLE structure for the data set.
- **return value:**
  Returns 1 on success. On failure, returns 0 and records an error message.
- **see also:**
  - `SDDS_DisconnectFile` (4.17)

4.73 SDDS_RowCount

- **name:**
  SDDS_RowCount
- **description:**
  Used to retrieve the number of rows.
- **synopsis:** #include "SDDS.h"
  long SDDS_RowCount(SDDS_TABLE *SDDS_table)
- **arguments:**
  - **SDDS_table:** Address of the SDDS_TABLE structure for the data set.
- **return value:**
  Returns the number of rows.
- **see also:**
  - `SDDS_ArrayCount` (4.1)
  - `SDDS_ColumnCount` (4.7)
  - `SDDS_ParameterCount` (4.68)

4.74 SDDS_SaveLayout

- **name:**
  SDDS_SaveLayout
- **description:**
  Saves the SDDS header describing the layout of the data tables that will follow. Usually called before SDDS_WriteLayout.
- **synopsis:** #include "SDDS.h"
  long SDDS_SaveLayout(SDDS_TABLE *SDDS_table);
- **arguments:**
  - **SDDS_table:** Address of the SDDS_TABLE structure for the data set.
- **return value:**
  Returns 1 on success. On failure, returns 0 and records an error message.
- **see also:**
  - `SDDS_InitializeOutput` (4.64)
  - `SDDS_InitializeCopy` (4.61)
  - `SDDS_WriteLayout` (4.91)

4.75 SDDS_SetArray

- **name:**
  SDDS_SetArray
- **description:**
  Accepts a pointer to a structure containing the data and other information to be used for an array in the current data table of a data set.
- **synopsis:** #include "SDDS.h"
  long SDDS_SetArray(SDDS_TABLE *SDDS_table, char *array_name, int32_t mode, void *pointer, int32_t dimension-list);
- **arguments:**
  - **SDDS_table:** Address of the SDDS_TABLE structure for the data set.
  - **array_name:** A NULL-terminated character string giving the name of the array to return.
  - **mode:** SDDS_POINTER_ARRAY or SDDS_CONTIGUOUS_DATA.
  - **pointer:** Address of multidimensional pointer array indexing the data. The type of this pointer in the caller's program is *data-type**n, where *data-type* is the data type of the array and *n represents a sequence of n asterisks, n begin the number of dimensions of the array. (Note that for type SDDS_STRING, *data-type* is char *.) This corresponds to the pointer field of the SDDS_ARRAY structure (see the manual page for SDDS_GetArray). The data is copied so that the caller may change it after this call without affecting the behavior of SDDS routines.
  - **dimension-list:** n arguments of type long, where n is the number of dimensions. Successive arguments give the size in each dimension.
- **return value:**
  On success, returns 1. On failure, returns 0 and records an error message.
- **see also:**
  - `SDDS_NumberOfErrors` (4.67)
  - `SDDS_PrintErrors` (4.69)
  - `SDDS_GetArray` (4.34)

4.76 SDDS_SetColumn

- **name:**
  SDDS_SetColumn
- **description:**
  Sets the values for one data column.
- **synopsis:** #include "SDDS.h"
  long SDDS_SetColumn(SDDS_TABLE *SDDS_table, long mode, void *data, long rows, ...)
- **arguments:**

- **SDDS_table:** Address of the SDDS_TABLE structure for the data set.
- **mode:** Valid modes are SDDS_SET_BY_INDEX and SDDS_SET_BY_NAME.
- **data:** Pointer to an array of data. The elements of the array must be of the same type as the column type.
- **rows:** Number of rows in the column.
- **...:** If the mode is SDDS_SET_BY_INDEX it expects an integer argument for the index. If the mode is SDDS_SET_BY_NAME it expects a NULL-terminated character string giving the name of the column.

- **return value:**
  On success, returns 1. On failure, returns 0 and records an error message.

- **see also:**
  - `SDDS_SetColumnFromDoubles` (4.78)
  - `SDDS_SetColumnFromLongs` (4.79)

4.77 SDDS_SetColumnFlags

- **name:**
  SDDS_SetColumnFlags

- **description:**
  Sets initial values of accept/reject flags for the columns of the current data table of a data set. A non-zero flag indicates that a column is "of interest".

- **synopsis:** #include "SDDS.h"
  long SDDS_SetColumnFlags(SDDS_TABLE *SDDS_table, long column_flag_value);

- **arguments:**
  - **SDDS_table:** Address of the SDDS_TABLE structure for the data set.
  - **column_flag_value:** An integer value indicating the status to assign to all columns. A non-zero value indicates acceptance, while a zero value indicates rejection.

- **return value:**
  Returns 1 on success. On failure, returns 0 and records an error message.

- **see also:**
  - `SDDS_SetColumnsOfInterest` (4.80)
  - `SDDS_NumberOfErrors` (4.67)
  - `SDDS_PrintErrors` (4.69)

4.78 SDDS_SetColumnFromDoubles

- **name:**
SDDS_SetColumnFromDoubles
- **description:**
Sets the values for one data column.
- **synopsis:** #include "SDDS.h"
long SDDS_SetColumnFromDoubles(SDDS_TABLE *SDDS_table, long mode, double *data, long rows, ...)
- **arguments:**
  - **SDDS_table:** Address of the SDDS_TABLE structure for the data set.
  - **mode:** Valid modes are SDDS_SET_BY_INDEX and SDDS_SET_BY_NAME.
  - **data:** Pointer to an array of data.
  - **rows:** Number of rows in the column.
  - **...:** If the mode is SDDS_SET_BY_INDEX it expects an integer argument for the index. If the mode is SDDS_SET_BY_NAME it expects a NULL-terminated character string giving the name of the column.
- **return value:**
On success, returns 1. On failure, returns 0 and records an error message.
- **see also:**
  - `SDDS_SetColumn` (4.76)
  - `SDDS_SetColumnFromLongs` (4.79)

4.79 SDDS_SetColumnFromLongs

- **name:**
SDDS_SetColumnFromLongs
- **description:**
Sets the values for one data column.
- **synopsis:** #include "SDDS.h"
long SDDS_SetColumnFromLongs(SDDS_TABLE *SDDS_table, long mode, long *data, long rows, ...)
- **arguments:**
  - **SDDS_table:** Address of the SDDS_TABLE structure for the data set.
  - **mode:** Valid modes are SDDS_SET_BY_INDEX and SDDS_SET_BY_NAME.
  - **data:** Pointer to an array of data.
  - **rows:** Number of rows in the column.
  - **...:** If the mode is SDDS_SET_BY_INDEX it expects an integer argument for the index. If the mode is SDDS_SET_BY_NAME it expects a NULL-terminated character string giving the name of the column.

- **return value:**
  On success, returns 1. On failure, returns 0 and records an error message.

- **see also:**

  - `SDDS_SetColumn` (4.76)
  - `SDDS_SetColumnFromDoubles` (4.78)

4.80 SDDS_SetColumnsOfInterest

- **name:**
  SDDS_SetColumnsOfInterest

- **description:**
  Modifies the acceptance status of the columns of the current data table of a data set using the column names and information supplied by the caller.

- **synopsis:** #include "SDDS.h"
  long SDDS_SetColumnsOfInterest(SDDS_TABLE *SDDS_table, long mode, ...);

- **arguments:**

  - **SDDS_table:** Address of the SDDS_TABLE structure for the data set.
  - **mode:** One of the constants SDDS_NAME_ARRAY, SDDS_NAMES_STRING, SDDS_NAME_STRINGS, or SDDS_MATCH_STRING, defined in SDDS.h. The calling syntax for each of these modes is:
    * SDDS_NAME_ARRAY - SDDS_SetColumnsOfInterest(SDDS_TABLE *SDDS_table, SDDS_NAME_ARRAY, long n_entries, char **name), where name is an array of pointers to n_entries NULL-terminated character strings. The columns named in these strings are added to the list of columns deemed to be "of interest" by the caller. The order in which the caller gives the column names is recorded, and calls to SDDS_GetRow and SDDS_GetMatrixOfRows return the columns in this order.
    * SDDS_NAMES_STRING - SDDS_SetColumnsOfInterest(SDDS_TABLE *SDDS_table, SDDS_NAMES_STRING, char *string), where string is a NULL-terminated character string giving the names of the columns of interest separated by white-space. The columns named in these strings are added to the list of columns deemed to be "of interest" by the caller. The order in which the caller gives the column names is recorded, and calls to SDDS_GetRow and SDDS_GetMatrixOfRows return the columns in this order.
    * SDDS_NAME_STRINGS - SDDS_SetColumnsOfInterest(SDDS_TABLE *SDDS_table, SDDS_NAME_STRINGS, char *string, ..., NULL), where string and all following arguments are NULL-terminated character strings, each giving the name of a column. The list is terminated by the value NULL. The columns named in these strings are added to the list of columns deemed to be "of interest" by the caller. The order in which the caller gives the column names is recorded, and calls to SDDS_GetRow and SDDS_GetMatrixOfRows return the columns in this order.

* SDDS_MATCH_STRING - SDDS_SetColumnsOfInterest(SDDS_TABLE *SDDS_table, SDDS_MATCH_STRING, char *string, long logic_mode), where string is a wildcard-containing, NULL-terminated character string to which column names will be matched. The columns so matched are added to the list of columns deemed to be "of interest" by the caller. The order in which the caller gives the column names is recorded, and calls to SDDS_GetRow and SDDS_GetMatrixOfRows return the columns in this order. See the manual page for SDDS_Logic for a discussion of the logic_mode parameter.

- **return value:**
  Returns 1 on success. On failure, returns 0 and records an error message.

- **see also:**
  - `SDDS_DeleteUnsetColumns` (4.28)
  - `SDDS_Logic` (4.66)
  - `SDDS_NumberOfErrors` (4.67)
  - `SDDS_PrintErrors` (4.69)
  - `SDDS_SetColumnFlags` (4.77)

4.81 SDDS_SetParameters

- **name:**
  SDDS_SetParameters

- **description:**
  Sets the value of one or more parameters for the current data table of a data set. Must be preceeded by a call to SDDS_StartTable to initialize the table.

- **synopsis:** #include "SDDS.h"
  long SDDS_SetParameters(SDDS_TABLE *SDDS_table, long mode, name-or-index, value-or-pointer, ..., terminator)

- **arguments:**
  - **SDDS_table:** Address of the SDDS_TABLE structure for the data set.
  - **mode:** A bit-wise combination of the constants SDDS_SET_BY_INDEX, SDDS_SET_BY_NAME, SDDS_PASS_BY_VALUE, and SDDS_PASS_BY_REFERENCE, which are defined in SDDS.h. One and only one of SDDS_SET_BY_INDEX and SDDS_SET_BY_NAME must be set, indicating that the parameters to be set are indicated by index number or by name, respectively. One and only one of SDDS_PASS_BY_VALUE and SDDS_PASS_BY_REFERENCE must be set, indicating that the values for the parameters are passed by value or by reference, respectively. The syntax for the four possible combinations is:
    * mode = SDDS_SET_BY_INDEX — SDDS_PASS_BY_VALUE: long SDDS_SetParameters(SDDS_TABLE *SDDS_table, long mode, long index1, value1, long index2, value2, ..., -1)

* mode = SDDS_SET_BY_INDEX — SDDS_PASS_BY_REFERENCE: long SDDS_SetParameters(SDDS_TABLE *SDDS_table, long mode, long index1, void *data1, long index2, void *data2, ..., -1)
* mode = SDDS_SET_BY_NAME — SDDS_PASS_BY_VALUE: long SDDS_SetParameters(SDDS_TABLE *SDDS_table, long mode, char *name1, value1, char *name2, value2, ..., NULL)
* mode = SDDS_SET_BY_NAME — SDDS_PASS_BY_REFERENCE: long SDDS_SetParameters(SDDS_TABLE *SDDS_table, long mode, char *name1, void *data1, char *name2, void *data2, ..., NULL)

Note that for data of type SDDS_STRING, pass-by-value means passing an item of type char *, while pass by reference means passing an item of type char **.

- **return value:**
Returns 1 on success. On failure, returns 0 and records an error message.

- **see also:**
  - `SDDS_SetParametersFromDoubles` (4.82)
  - `SDDS_StartTable` (4.86)
  - `SDDS_NumberOfErrors` (4.67)
  - `SDDS_PrintErrors` (4.69)

4.82 SDDS_SetParametersFromDoubles

- **name:**
SDDS_SetParametersFromDoubles

- **description:**
Sets the value of one or more parameters for the current data table of a data set. Must be preceeded by a call to SDDS_StartTable to initialize the table.

- **synopsis:** #include "SDDS.h"
long SDDS_SetParametersFromDoubles(SDDS_TABLE *SDDS_table, long mode, name-or-index, value-or-pointer, ..., terminator)

- **arguments:**
  - **SDDS_table:** Address of the SDDS_TABLE structure for the data set.
  - **mode:** A bit-wise combination of the constants SDDS_SET_BY_INDEX, SDDS_SET_BY_NAME, SDDS_PASS_BY_VALUE, and SDDS_PASS_BY_REFERENCE, which are defined in SDDS.h. One and only one of SDDS_SET_BY_INDEX and SDDS_SET_BY_NAME must be set, indicating that the parameters to be set are indicated by index number or by name, respectively. One and only one of SDDS_PASS_BY_VALUE and SDDS_PASS_BY_REFERENCE must be set, indicating that the values for the parameters are passed by value or by reference, respectively. The syntax for the four possible combinations is:
    * mode = SDDS_SET_BY_INDEX — SDDS_PASS_BY_VALUE: long SDDS_SetParameters(SDDS_TABLE *SDDS_table, long mode, long index1, value1, long index2, value2, ..., -1)

* mode = SDDS_SET_BY_INDEX — SDDS_PASS_BY_REFERENCE: long SDDS_SetParameters(SDDS_TABLE *SDDS_table, long mode, long index1, double *data1, long index2, double *data2, ..., -1)
* mode = SDDS_SET_BY_NAME — SDDS_PASS_BY_VALUE: long SDDS_SetParameters(SDDS_TABLE *SDDS_table, long mode, char *name1, value1, char *name2, value2, ..., NULL)
* mode = SDDS_SET_BY_NAME — SDDS_PASS_BY_REFERENCE: long SDDS_SetParameters(SDDS_TABLE *SDDS_table, long mode, char *name1, double *data1, char *name2, double *data2, ..., NULL)

- **return value:**
  Returns 1 on success. On failure, returns 0 and records an error message.

- **see also:**
  - `SDDS_SetParameters` (4.81)
  - `SDDS_StartTable` (4.86)
  - `SDDS_NumberOfErrors` (4.67)
  - `SDDS_PrintErrors` (4.69)

4.83 SDDS_SetRowFlags

- **name:**
  SDDS_SetRowFlags

- **description:**
  Sets initial values of acceptance flags for the rows of the current data table of a data set. A non-zero flag indicates that a row is "of interest".

- **synopsis:** #include "SDDS.h"
  long SDDS_SetRowFlags(SDDS_TABLE *SDDS_table, long row_flag_value);

- **arguments:**
  - **SDDS_table:** Address of the SDDS_TABLE structure for the data set.
  - **row_flag_value:** An integer value indicating the status to assign to all rows. A non-zero value indicates acceptance, while a zero value indicates rejection.

- **return value:**
  Returns 1 on success. On failure, returns 0 and records an error message.

- **see also:**
  - `SDDS_SetRowsOfInterest` (4.84)
  - `SDDS_NumberOfErrors` (4.67)
  - `SDDS_PrintErrors` (4.69)

4.84 SDDS_SetRowsOfInterest

- **name:**
  SDDS_SetRowsOfInterest
- **description:**
  Modifies the acceptance status of the rows of the current data table of a data set by wildcard matching of a string to the entries in a specified column.
- **synopsis:** #include "SDDS.h"
  long SDDS_SetRowsOfInterest(SDDS_TABLE *SDDS_table, char *selection_column, char *matching_string, long logic);
- **arguments:**
  - **SDDS_table:** Address of the SDDS_TABLE structure for the data set.
  - **selection_column:** A NULL-terminated character string giving the name of the column the values in which will be used for modifying the acceptance status of each row. The column must be of type SDDS_STRING.
  - **matching_string:** A NULL-terminated, optionally wildcard-containing character string to which the entries in the selection column are matched.
  - **logic:** A word of bit flags indicating how to combine the previous status of each row with the acceptance status based on matching to the matching_string. See the manual entry for SDDS_Logic for details.
- **return value:**
  On success, returns the total number of rows that are of interest. On failure, returns -1 and records an error message.
- **see also:**
  - `SDDS_DeleteUnsetRows` (4.29)
  - `SDDS_FilterRowsOfInterest` (4.30)
  - `SDDS_GetMatrixOfRows` (4.49)
  - `SDDS_GetRow` (4.56)
  - `SDDS_Logic` (4.66)
  - `SDDS_NumberOfErrors` (4.67)
  - `SDDS_PrintErrors` (4.69)
  - `SDDS_SetRowFlags` (4.83)

4.85 SDDS_SetRowValues

- **name:**
  SDDS_SetRowValues
- **description:**
  Allows setting the values in a specified row of the current data table of a data set. Must be preceeded by a call to SDDS_StartTable to initialize the table.

- **synopsis:** #include "SDDS.h"
  long SDDS_SetRowValues(SDDS_TABLE *SDDS_table, long mode, long row, ...);
- **arguments:**
  - **SDDS_table:** Address of the SDDS_TABLE structure for the data set.
  - **mode:** A bit-wise combination of the constants SDDS_SET_BY_INDEX, SDDS_SET_BY_NAME, SDDS_PASS_BY_VALUE, and SDDS_PASS_BY_REFERENCE, which are defined in SDDS.h. One and only one of SDDS_SET_BY_INDEX and SDDS_SET_BY_NAME must be set, indicating that the values to be set are indicated by index number or by name, respectively. One and only one of SDDS_PASS_BY_VALUE and SDDS_PASS_BY_REFERENCE must be set, indicating that the values for the values are passed by value or by reference, respectively. The syntax for the four possible combinations is:
    * mode = SDDS_SET_BY_INDEX — SDDS_PASS_BY_VALUE: long SDDS_SetRowValues(SDDS_TABLE *SDDS_table, long mode, long index1, value1, long index2, value2, ..., -1)
    * mode = SDDS_SET_BY_INDEX — SDDS_PASS_BY_REFERENCE: long SDDS_SetRowValues(SDDS_TABLE *SDDS_table, long mode, long index1, void *data1, long index2, void *data2, ..., -1)
    * mode = SDDS_SET_BY_NAME — SDDS_PASS_BY_VALUE: long SDDS_SetRowValues(SDDS_TABLE *SDDS_table, long mode, char *name1, value1, char *name2, value2, ..., NULL)
    * mode = SDDS_SET_BY_NAME — SDDS_PASS_BY_REFERENCE: long SDDS_SetRowValues(SDDS_TABLE *SDDS_table, long mode, char *name1, void *data1, char *name2, void *data2, ..., NULL) Note that for data of type SDDS_STRING, pass-by-value means passing an item of type char *, while pass by reference means passing an item of type char **.
  - **rows:** The row of the data table into which the data is to be placed.
- **return value:**
- **see also:**
  - `SDDS_StartTable` (4.86)
  - `SDDS_NumberOfErrors` (4.67)
  - `SDDS_PrintErrors` (4.69)

4.86 SDDS_StartTable

- **name:**
  SDDS_StartTable
- **description:**
  Initializes a SDDS_TABLE structure in preparation for placing data into the table. Must be preceeded by a call to SDDS_InitializeOutput. May be called repeatedly to proceed to the next table in a data set, though the caller is expected to use SDDS_WriteTable to write each table to the disk

- **synopsis:** #include "SDDS.h"
  long SDDS_StartTable(SDDS_TABLE *SDDS_table, long expected_n_rows);
- **arguments:**
  - **SDDS_table:** Address of the SDDS_TABLE structure for the data set.
  - **expected_n_rows:** The expected number of rows in the data table, used to preallocate memory for storing data values.
- **return value:**
  Returns 1 on success. On failure, returns 0 and records an error message.
- **see also:**
  - `SDDS_InitializeOutput` (4.64)
  - `SDDS_NumberOfErrors` (4.67)
  - `SDDS_PrintErrors` (4.69)
  - `SDDS_WriteTable` (4.92)

4.87 SDDS_Terminate

- **name:**
  SDDS_Terminate
- **description:**
  Erases the data set description and frees all memory being used for a data set. If a file is defined and open for the data set, the file is closed.
- **synopsis:** #include "SDDS.h"
  long SDDS_Terminate(SDDS_TABLE *SDDS_table)
- **arguments:**
  - **SDDS_table:** Address of the SDDS_TABLE structure for the data set.
- **return value:**
  Returns 1 on success. On failure, returns 0 and records an error message.
- **see also:**
  - `SDDS_NumberOfErrors` (4.67)
  - `SDDS_PrintErrors` (4.69)

4.88 SDDS_VerifyArrayExists

- **name:**
  SDDS_VerifyArrayExists
- **description:**
  Returns the index of a named array if it exists as the specified data type.

- **synopsis:** #include "SDDS.h"
  long SDDS_VerifyArrayExists(SDDS_TABLE *SDDS_table, long mode, char *name) long SDDS_VerifyArrayExists(SDDS_TABLE *SDDS_table, long mode, long type, char *name)
- **arguments:**
  - **SDDS_table:** Address of the SDDS_TABLE structure for the data set.
  - **mode:** Valid modes are FIND_SPECIFIED_TYPE, FIND_ANY_TYPE, FIND_NUMERIC_TYPE, FIND_FLOATING_TYPE, FIND_INTEGER_TYPE
  - **type:** If mode is FIND_SPECIFIED_TYPE it expects an SDDS data type.
  - **name:** Name of the array in question.
- **return value:**
  If the desired array exists it returns the index of the SDDS array. Otherwise it returns -1.

  On failure, returns zero and records an error message.
- **see also:**
  - `SDDS_GetArrayDefinition` (4.35)
  - `SDDS_GetArrayIndex` (4.36)
  - `SDDS_GetArrayInformation` (4.37)

4.89 SDDS_VerifyColumnExists

- **name:**
  SDDS_VerifyColumnExists
- **description:**
  Returns the index of a named column if it exists as the specified data type.
- **synopsis:** #include "SDDS.h"
  long SDDS_VerifyColumnExists(SDDS_TABLE *SDDS_table, long mode, char *name) long SDDS_VerifyColumnExists(SDDS_TABLE *SDDS_table, long mode, long type, char *name)
- **arguments:**
  - **SDDS_table:** Address of the SDDS_TABLE structure for the data set.
  - **mode:** Valid modes are FIND_SPECIFIED_TYPE, FIND_ANY_TYPE, FIND_NUMERIC_TYPE, FIND_FLOATING_TYPE, FIND_INTEGER_TYPE
  - **type:** If mode is FIND_SPECIFIED_TYPE it expects an SDDS data type.
  - **name:** Name of the column in question.
- **return value:**
  If the desired column exists it returns the index of the SDDS column. Otherwise it returns -1.

  On failure, returns zero and records an error message.
- **see also:**

- SDDS_GetColumnDefinition (4.41)
- SDDS_GetColumnIndex (4.44)
- SDDS_GetColumnInformation (4.45)

4.90 SDDS_VerifyParameterExists

- **name:**
SDDS_VerifyParameterExists
- **description:**
Returns the index of a named parameter if it exists as the specified data type.
- **synopsis:** #include "SDDS.h"
long SDDS_VerifyParameterExists(SDDS_TABLE *SDDS_table, long mode, char *name)
long SDDS_VerifyParameterExists(SDDS_TABLE *SDDS_table, long mode, long type, char *name)
- **arguments:**
  - **SDDS_table:** Address of the SDDS_TABLE structure for the data set.
  - **mode:** Valid modes are FIND_SPECIFIED_TYPE, FIND_ANY_TYPE, FIND_NUMERIC_TYPE, FIND_FLOATING_TYPE, FIND_INTEGER_TYPE
  - **type:** If mode is FIND_SPECIFIED_TYPE it expects an SDDS data type.
  - **name:** Name of the parameter in question.
- **return value:**
If the desired parameter exists it returns the index of the SDDS parameter. Otherwise it returns -1.

  On failure, returns zero and records an error message.
- **see also:**
  - SDDS_GetParameterDefinition (4.51)
  - SDDS_GetParameterIndex (4.52)
  - SDDS_GetParameterInformation (4.53)

4.91 SDDS_WriteLayout

- **name:**
SDDS_WriteLayout
- **description:**
Writes the SDDS header describing the layout of the data tables that will follow.
- **synopsis:** #include "SDDS.h"
long SDDS_WriteLayout(SDDS_TABLE *SDDS_table);
- **arguments:**
  - **SDDS_table:** Address of the SDDS_TABLE structure for the data set.

- **return value:**
  Returns 1 on success. On failure, returns 0 and records an error message.

- **see also:**

  - `SDDS_InitializeOutput` (4.64)
  - `SDDS_InitializeCopy` (4.61)
  - `SDDS_SaveLayout` (4.74)

4.92 SDDS_WriteTable

- **name:**
  SDDS_WriteTable

- **description:**
  Writes out the current data table (i.e., the parameters and the tabular data). Must be preceeded by a call to SDDS_WriteLayout at some point.

- **synopsis:** #include "SDDS.h"
  long SDDS_WriteTable(SDDS_TABLE *SDDS_table);

- **arguments:**

  - **SDDS_table:** Address of the SDDS_TABLE structure for the data set.

- **return value:**
  Returns 1 on success. On failure, returns 0 and records an error message.

- **see also:**

  - `SDDS_WriteLayout` (4.91)
  - `SDDS_NumberOfErrors` (4.67)
  - `SDDS_PrintErrors` (4.69)

Contents

1 Definition of SDDS Protocol **1**
1.1 Structure of the SDDS Header 1
1.1.1 Data Set Description 2
1.1.2 Tabular-Data Column Definition 2
1.1.3 Parameter Definition 3
1.1.4 Array Data Definition 4
1.1.5 Header File Include Specification 5
1.1.6 Data Mode and Arrangement Defintion 5
1.2 Structure of SDDS ASCII Data Pages 6
1.3 Structure of SDDS Binary Data Pages 7

2 Overview of Library Routines **7**
2.1 Input Routines . . . 7
2.1.1 Setup . . . 7
2.1.2 Determining File Contents . . . 8
2.1.3 Column Selection . . . 9
2.1.4 Row Selection . . . 9
2.1.5 Access to Tabular Data . . . 9
2.1.6 Access to Array Data . . . 10
2.1.7 Access to Parameter Data . . . 10
2.2 Output Routines . . . 10
2.2.1 Setup . . . 10
2.2.2 Defining Data Set Elements . . . 10
2.2.3 Defining Data Table Elements . . . 10
2.2.4 Output . . . 11
2.3 Error Handling and Utilities . . . 11

3 Two Templates for SDDS Application Organization **11**
3.1 Accessing Data Stored in an SDDS File . . . 11
3.2 SDDS Output of Internally-Generated Data . . . 12

4 Manual Pages **13**
4.1 SDDS_ArrayCount . . . 13
4.2 SDDS_CastValue . . . 14
4.3 SDDS_CheckArray . . . 15
4.4 SDDS_CheckColumn . . . 15
4.5 SDDS_CheckParameter . . . 16
4.6 SDDS_ClearErrors . . . 16
4.7 SDDS_ColumnCount . . . 17
4.8 SDDS_ConvertToDouble . . . 17
4.9 SDDS_CopyArrays . . . 18
4.10 SDDS_CopyColumns . . . 18
4.11 SDDS_CopyLayout . . . 19
4.12 SDDS_CopyParameters . . . 19
4.13 SDDS_CopyRowDirect . . . 20
4.14 SDDS_CopyRowsOfInterest . . . 20
4.15 SDDS_CopyTable . . . 21
4.16 SDDS_CountRowsOfInterest . . . 21
4.17 SDDS_DisconnectFile . . . 22
4.18 SDDS_DefineArray . . . 22
4.19 SDDS_DefineColumn . . . 23
4.20 SDDS_DefineParameter . . . 24
4.21 SDDS_DefineParameter1 . . . 25
4.22 SDDS_DefineSimpleColumn . . . 26
4.23 SDDS_DefineSimpleColumns . . . 27
4.24 SDDS_DefineSimpleParameter . . . 28
4.25 SDDS_DefineSimpleParameters . . . 29
4.26 SDDS_DeleteColumn . . . 30
4.27 SDDS_DeleteParameter . . . 30

4.28 SDDS_DeleteUnsetColumns . . . 31
4.29 SDDS_DeleteUnsetRows . . . 31
4.30 SDDS_FilterRowsOfInterest . . . 32
4.31 SDDS_FindArray . . . 32
4.32 SDDS_FindColumn . . . 33
4.33 SDDS_FindParameter . . . 33
4.34 SDDS_GetArray . . . 34
4.35 SDDS_GetArrayDefinition . . . 35
4.36 SDDS_GetArrayIndex . . . 36
4.37 SDDS_GetArrayInformation . . . 36
4.38 SDDS_GetArrayNames . . . 37
4.39 SDDS_GetArrayType . . . 38
4.40 SDDS_GetColumn . . . 38
4.41 SDDS_GetColumnDefinition . . . 39
4.42 SDDS_GetColumnInDoubles . . . 39
4.43 SDDS_GetColumnInLong . . . 40
4.44 SDDS_GetColumnIndex . . . 41
4.45 SDDS_GetColumnInformation . . . 41
4.46 SDDS_GetColumnNames . . . 42
4.47 SDDS_GetColumnType . . . 43
4.48 SDDS_GetMatrixFromColumn . . . 43
4.49 SDDS_GetMatrixOfRows . . . 44
4.50 SDDS_GetParameter . . . 45
4.51 SDDS_GetParameterDefinition . . . 45
4.52 SDDS_GetParameterIndex . . . 46
4.53 SDDS_GetParameterInformation . . . 47
4.54 SDDS_GetParameterNames . . . 48
4.55 SDDS_GetParameterType . . . 48
4.56 SDDS_GetRow . . . 49
4.57 SDDS_GetTypeSize . . . 49
4.58 SDDS_GetValue . . . 50
4.59 SDDS_InitializeAppend . . . 51
4.60 SDDS_InitializeAppendToPage . . . 51
4.61 SDDS_InitializeCopy . . . 52
4.62 SDDS_InitializeHeaderlessInput . . . 52
4.63 SDDS_InitializeInput . . . 53
4.64 SDDS_InitializeOutput . . . 53
4.65 SDDS_LengthenTable . . . 54
4.66 SDDS_Logic . . . 55
4.67 SDDS_NumberOfErrors . . . 55
4.68 SDDS_ParameterCount . . . 56
4.69 SDDS_PrintErrors . . . 56
4.70 SDDS_PrintTypedValue . . . 57
4.71 SDDS_ReadTable . . . 58
4.72 SDDS_ReconnectFile . . . 58
4.73 SDDS_RowCount . . . 59
4.74 SDDS_SaveLayout . . . 59
4.75 SDDS_SetArray . . . 60

4.76 SDDS_SetColumn 60
4.77 SDDS_SetColumnFlags 61
4.78 SDDS_SetColumnFromDoubles 62
4.79 SDDS_SetColumnFromLongs 62
4.80 SDDS_SetColumnsOfInterest 63
4.81 SDDS_SetParameters 64
4.82 SDDS_SetParametersFromDoubles 65
4.83 SDDS_SetRowFlags 66
4.84 SDDS_SetRowsOfInterest 67
4.85 SDDS_SetRowValues 67
4.86 SDDS_StartTable 68
4.87 SDDS_Terminate 69
4.88 SDDS_VerifyArrayExists 69
4.89 SDDS_VerifyColumnExists 70
4.90 SDDS_VerifyParameterExists 71
4.91 SDDS_WriteLayout 71
4.92 SDDS_WriteTable 72