

NAME

data – built-in functions for handling internal data

DESCRIPTION

spec stores data in up to 256 independent data arrays called groups. Each group is configured (see below) to have a particular number of data elements per point. For example, each point in a group could have elements for H, K, L, and detector counts. Alternatively, each point could have just one element and be used to hold data obtained from an MCA.

Groups are configured using the `data_grp()` function. A group can have up to 2048 elements per point. The maximum number of points in a group is determined by the product of the number of elements per point and the number of points. That product can be no more than 65,536, and may be slightly less depending on how the number of elements divides into 2048. The maximum number of points for all groups is 262,144. (These limits are arbitrary and are set to control the size of static data arrays and auxiliary files. If requested, CSS can make the limits larger.)

When starting spec for the first time or with the `-f` (fresh) flag, one data group (group 0) is configured for 4096 points, with each point consisting of two elements. The old-style functions for accessing the internal data (`pl_get()`, `pl_put()`, etc.) are described at the end of this section. Their use is discouraged, though, as they will not be available in future releases.

When leaving spec, the current data group configuration and data points are saved.

spec has several functions to manipulate the internal data. These functions allow unary and binary arithmetic operations, math functions and analysis operations to be performed on all the elements of a group or among elements in different groups.

In the functions described below, if an element number is negative, the element number is obtained by adding the number of elements per point in the group to the negative element number. For example, element `-1` is the last element, element `-2` is the second to last, etc.

All functions reset to command level if an invalid group, point or element is given as an argument. Functions that don't need to return anything in particular return zero.

BUILT-IN FUNCTIONS

`data_grp(g, n, w)` – Configures data group *g*. The group will have *n* points, each having *w* elements. If *n* and *w* match the previous values for the group, the data in the group is unchanged. Otherwise, the data values of the reconfigured group are set to zero. If *w* is zero, the

group is eliminated. If n is zero, as many points as possible are configured. If n is negative, as many points as possible, but not more than $-n$ are configured. If g is -1 , the current group configuration is displayed.

`data_info(g , s)` – Returns data group configuration information for group g , according to the string s . Values for s are:

- "npts" – the number of configured points.
- "elem" – the number of configured elements.
- "last" – the number of the last point added to the group.
- "precision" – the number of bytes per element, either 4 or 8.

If the group number is invalid, or if the string s is none of the above, returns -1 .

`data_get(g , n , e)` – Returns the value of element e of point n in group g .

`data_put(g , n , e , v)` – Assigns the value v to element e of point n in group g .

`data_nput(g , n , $v0$ [, $v1$...])` – Assigns values to point n of group g . Element 0 is assigned $v0$, element 1 is assigned $v1$, etc. Not all elements need be given, although elements are assigned successively, starting at element 0.

`data_uop(gs , es , gd , ed , uop [, arg])` – Performs the unary operation specified by the string uop on element es for all points in group gs . The results are put in element ed of the corresponding points in group gd . The source and destination groups and/or elements may be the same. If the number of points in the groups differ, the operation is carried out on up to the smallest number of points among the groups.

Values for uop are:

- "clr" – clear to zero.
- "fill" – each element is set to point number, starting at 0.
- "neg" – negative of source.
- "abs" – absolute value of source.
- "inv" – inverse of source.
- "sin" – sine of source.
- "cos" – cosine of source.
- "tan" – tangent of source.
- "asin" – arcsine of source.
- "acos" – arccosine of source.

"atan" – arctangent of source.
 "log" – natural logarithm of source.
 "exp" – exponential of source.
 "log10" – log base 10 of source.
 "pow" – the *arg* power of source.
 "copy" – value of source.
 "rev" – reversed copy of source.
 "sqrt" – square root of source.
 "set" – all elements set to the value of *arg*.
 "contract" – every *arg* points are averaged to make a new point.
 "add" – source plus *arg*.
 "sub" – source minus *arg*.
 "mul" – source times *arg*.
 "div" – source divided by *arg*.

If any of the operations would result in an exception (divide by zero, log or square root of a negative number, etc), the operation is not performed and a count of the operations skipped is printed as an error message.

`data_bop(gs0, es0, gs1, es1, gd, ed, bop)` – Performs the binary operation specified by the string *bop* on elements *es0* and *es1* for all points in the groups *gs0* and *gs1*. The results are put in element *ed* for the corresponding points of group *gd*. The source and destination groups and/or elements may be the same. If the number of points in the groups differ, the operation is carried out on up to the smallest number of points among the groups.

Values for *bop* are:

"add" – The sum of the source elements.
 "sub" – Source 0 minus source 1.
 "mul" – The product of the source elements.
 "div" – Source 0 divided by source 1.

If the divide would result in an exception, the operation is not performed and a count of the operations skipped is printed as an error message.

`data_anal(g, s, n, e0, e1, op [, arg])` – Performs the operations indicated by *op* on *n* points in group *g*, starting at point *s*. The operations use the values in element *e0* (if applicable) and *e1*. If *n* is zero, the operations are performed on points from *s* to the last point added using `data_nput()` or `data_put()`.

- "min" – Returns the minimum value of $e1$. ($e0$ is unused.)
- "max" – Returns the maximum value of $e1$. ($e0$ is unused.)
- "i_at_min" – Returns the point number of the data point with the minimum value of $e1$. ($e0$ is unused.)
- "i_at_max" – Returns the point number of the data point with the maximum value of $e1$. ($e0$ is unused.)
- "i_<=_value" – Returns the point number of the nearest data point in $e1$ at or below arg , starting from the first point. ($e0$ is unused.)
- "i_>=_value" – Returns the point number of the nearest data point in $e1$ at or above arg , starting at the last point. ($e0$ is unused.)
- "uhmx" – Returns the value in $e0$ corresponding to half the maximum value in $e1$, and at a higher index.
- "lhmx" – Returns the value in $e0$ corresponding to half the maximum value in $e1$, and at a lower index.
- "sum" – Returns the sum of the values in $e1$. ($e0$ is unused.)
- "fwhm" – Returns the full-width in $e0$ at half the maximum value of $e1$.
- "cfwhm" – Returns the center of the full-width in $e0$ at half the maximum value of $e1$.
- "com" – Returns the center of mass in $e0$ with respect to $e1$. The value is the sum of the products of each $e0$ and $e1$ divided by the number of points.
- "x_at_min" – Returns the value of $e0$ at the minimum in $e1$.
- "x_at_max" – Returns the value of $e0$ at the maximum in $e1$.
- "sumsq" – Returns the sum of the squares in $e1$. ($e0$ is unused.)

The following operations treat a data group as a two dimensional data array with rows indexed by the point number and the columns indexed by the element number. The operations work on the portion of the group determined by the starting row s , the number of rows n , the starting column $e0$ and the end row $e1$. As usual, if n is zero, all points (rows) from s to the last are considered. A negative element (column) number is added to the group width to obtain the element (column) to use.

- "gmin" – Returns the minimum value.
- "gmax" – Returns the maximum value.
- "gsum" – Returns the sum of all values.
- "i_at_gmin" – Returns the index number of the minimum value.
The index number is the row number times the group width

plus the element number.

"i_at_gmax" – Returns the index number, as defined above, of the maximum value.

`data_dump(g, s, n, e0 [, e1 ...] [, fmt1] [, fmt2])` – Efficiently writes elements from group *g* to turned on output devices. The starting point is *s* and the number of points is *n*. The elements specified by *e0*, *e1*, etc., are printed. If *e0* is the string "all" , all the elements for each point are printed. If *n* is zero, only the points from *s* to the last point added using `data_nput()` or `data_put()` are printed. The element arguments can be combined in a single space- or comma-delimited string. The optional argument *fmt1* is a string, having the format "%#", that specifies how many data points (specified by the number #) are to be printed on each line. If the number # is followed by the letter C, a backslash is added to each continued line, appropriate for saving MCA data in manageable length lines. New versions (since May 1, 1995) of the C-PLOT *scans.4* user function interpret the continued lines correctly for MCA data. The optional argument *fmt2* is a string that specifies an alternate *printf()*-style format for the values. Only e, g and f formats are recognized. For example, "%15.8f" uses fixed-point format with eight digits after the decimal point and a fifteen-character-wide field. The default output format is "%g" . See *printf()* in a C manual for more information. Note that in the default installation, the internal data arrays use single-precision floating values, which contain only about 8 decimal digits of significance.

`data_read(file_name, g, s, n)` – Reads data from the ASCII file *file_name*, and stuffs the data into group *g* starting at point *s*, reading up to *n* points. If *n* is zero, all the points in the file are read. The values on each line of the file are assigned into successive elements for each point in the group. If there are more elements on a line in the file than fit in the group, or if there are more points in the file than in the group, the extra values are ignored. Lines beginning with the # character are ignored. Returns -1 if the file can't be opened, otherwise returns the number of points read. At present, the maximum input line length is 2,048 characters.

`data_read(arr, g, s, n)` – Similar to the above usage, but transfers the data from the data array *arr* to the data group. Rows in the data array correspond to points in the data group. Columns in the data array correspond to elements in the data group. See the *arrays* help file for information on data arrays.

`data_plot(g, s, n, e0, e1 [, e2 ...])` – Plots the current data in group *g* starting at point *s* and plotting *n* points. Element *e0* is used for *x*. Elements given by the subsequent arguments (up to a maximum of 64) are plotted along the *y* axis. The element arguments can be combined in a single space- or comma-delimited string. If *n* is zero, only the points from *s* to the last point added using `data_nput()` or `data_put()` are plotted. If preceded by a call of `plot_cntl("add-point")` and the ranges have not changed, only point *s+n-1* is drawn. If preceded by a call of `plot_cntl("addline")` the current plot will not be erased, and the plot ranges will not be changed. The plotting area is not automatically erased by a call of `data_plot()`—use `plot_cntl("erase")` for that. The axis ranges are set using the `plot_range()` function. See `plot_cntl()` for other options that affect drawing the plot.

`data_plot(g, s, n, "all")` – As above, but uses element zero for *x* and the remaining elements (up to a maximum of 64) for *y* values. The number of elements is set with the `data_grp()` function.

`data_fit(pars, g, s, n, edata, epars [, ...])` – Performs a linear fit of the data in element *edata* to the terms in the elements specified by *epars*. The fitted parameters are returned in the array *pars* supplied by the user. The function returns the *chi-squared* value of the fit, if the fit was successful. A `-1` is returned if there are insufficient arguments or the covariance matrix is singular. The fit algorithm is along the same lines as the `lfit()` routine in *Numerical Recipes* (W.H. Press, et al., Cambridge University Press, 1986, page 512).

OLD-STYLE COMPATIBILITY FUNCTIONS

The following functions are currently supported, but will be eliminated in future releases. New macros should be written using the functions described above.

`pl_put(i, x, y)` – Adds *x* and *y* as elements 0 and 1 of group 0, point *i*.

`pl_xget(i)` – Returns the value of element 0 in group 0, point *i*.

`pl_yget(i)` – Returns the value of element 1 in group 0, point *i*.

`pl_dump(s, n, how)` – Efficiently writes pairs of elements from group 0 to turned on output devices. The starting point is *s* and the number of points is *n*. If *how*=0 element 0 values are printed, if *how*=1 element 1 values are printed and if *how*=2 both values are printed.

`plot_pts(x, s, n)` – Plots element 1 versus element 0 of the data values in group 0 starting at point *s* and plotting *n* points. If preceded by a

data

reference

data

call of `plot_cntl("addpoint")` and the ranges have not changed,
only point $s+n-1$ is drawn.

SEE ALSO

`data_pipe plot splot`