

Extended IDL Help

This page was created by the IDL library routine `mk_html_help`. For more information on this routine, refer to the IDL Online Help Navigator or type:

```
? mk_html_help
```

at the IDL command line prompt.

Last modified: Thu Sep 5 12:44:09 2013.

List of Routines

- [ATANH](#)
- [ATOMIC_WEIGHT](#)
- [BESELI_FRACT](#)
- [BESELK_FRACT](#)
- [CHEM2LBL](#)
- [CHISQR](#)
- [CIRCLE_FIT](#)
- [CLEAR](#)
- [COM_FIND](#)
- [CONT_IMAGE](#)
- [CONT_IMAGE2](#)
- [CURVE_LABEL](#)
- [CW_BGROUPO_RXO](#)
- [CW_CURVE_LABEL](#)
- [CW_DRAWSIZE](#)
- [CW_FIELD_RXO](#)
- [CW_FSLIDER_RXO](#)
- [CW_LEGEND_RXO](#)
- [CW_PLOTAXES](#)
- [CW_PLOTAXIS](#)
- [CW_PLOTLABEL](#)
- [CW_PLOTSTYLE](#)
- [CW_PLOTSTYLES](#)
- [CW_PLOTTITLE_CHAR](#)
- [CW_VECTOR](#)
- [DGTZ_IMAGE](#)

- [DGTZ_PLOT](#)
- [DIALOG](#)
- [DISPLAYED_TABLE_CELLS](#)
- [DISPLAY_FONT](#)
- [DLIB](#)
- [EDGE_FIND](#)
- [ELECTRON_MFP](#)
- [EPlot](#)
- [EROM](#)
- [ERRORF_FIT](#)
- [EXPO_FIT](#)
- [FILE_DATE](#)
- [FINDEX](#)
- [FLOYD_SAMPLING](#)
- [FRACTAL_FIT](#)
- [FWHM](#)
- [GAUSSEXPO_FIT](#)
- [GAUSS_FIT](#)
- [GET_PEAK](#)
- [GET_PT](#)
- [GET_ROI](#)
- [GHOSTVIEW](#)
- [GREEK](#)
- [KAISER_BESSEL](#)
- [LEGEND_RXO](#)
- [LPRINT](#)
- [LS](#)
- [MAKE_LATEX_TBL](#)
- [MK_BITARRAY](#)
- [MK_NEW_PTRS](#)
- [MORE](#)
- [MPFIT](#)
- [OEPlot](#)
- [PLOT_MOVIE](#)
- [PLOT_PRINT](#)
- [PLOT_TEXT](#)
- [PROFILE_NI](#)
- [PTRS_NEW](#)
- [PWD](#)
- [RECROI](#)
- [RECTANGLE](#)

- [REC_IMAGE](#)
- [ROI_WIDTH](#)
- [ROTATION](#)
- [ROT_MAT](#)
- [RXO_COLOR](#)
- [SECONDS2CLOCK](#)
- [SHIFT_PLOT](#)
- [SHOW_CT](#)
- [SINC](#)
- [SINCSQUARE FIT](#)
- [SMALL_WINDOW](#)
- [SP](#)
- [SQUARE_PLOT](#)
- [SYM](#)
- [SYMBOLS](#)
- [TEXT_WIDTH](#)
- [TRACK_PLOT](#)
- [TWOSCOMPLEMENT](#)
- [VALUE_TO_INDEX](#)
- [VECTOR](#)
- [WRITE_MPEG](#)
- [XDISPLAYFILE](#)
- [XWD2GIF](#)

Routine Descriptions

ATANH

[\[Next Routine\]](#) [\[List of Routines\]](#)

NAME :

ATANH

PURPOSE :

Inverse of TANH

$\operatorname{atanh} z = 1/2 \ln((1+z)/(1-z))$

CALLING SEQUENCE :

Result=ATANH(Input)

MODIFICATION HISTORY:

David L. Windt, RXO, April 2013
davidwindt@gmail.com

(See `./atanh.pro`)

ATOMIC_WEIGHT

[\[Previous Routine\]](#) [\[Next Routine\]](#) [\[List of Routines\]](#)

NAME:

ATOMIC_WEIGHT

PURPOSE:

Function to return the atomic weight of specified chemical elements.

CALLING SEQUENCE:

Result=ATOMIC_WEIGHT(SYMBOL)

INPUTS:

SYMBOL - A string or string array specifying the name or names of the chemical elements. Each element of SYMBOL must be a one or two character string, corresponding to the chemical symbol of the atom. Case is ignored.

KEYWORD PARAMETERS:

ALL - Set this to return all 92 atomic weights and symbols.

OUTPUTS:

Result - The atomic weight of the specified atom or atoms.

RESTRICTIONS:

Only the first 92 elements are available.

PROCEDURE:

The mass of the proton is first calculated using quantum field theory, and then...actually, it's just a lookup table.

EXAMPLE:

Print the atomic weight of carbon:

```
print,ATOMIC_WEIGHT('C')
```

MODIFICATION HISTORY:

David L Windt, Bell Labs, May 1997

1-Sep-13: Returns a value of -1 if the value for SYMBOL is invalid.

davidwindt@gmail.com

(See `./atomic_weight.pro`)

BESELI_FRACT

[\[Previous Routine\]](#) [\[Next Routine\]](#) [\[List of Routines\]](#)

NAME:

BESELI_FRACT

PURPOSE:

This function returns the Modified Bessel Function of the First Kind of Order N, for any N, i.e., including fractional and negative orders.

CALLING SEQUENCE:

Result = BESELI_FRACT(X, N)

INPUTS:

X - The value for which the I Bessel function is required. X must be greater than 0. The result will have the same dimensions as X.

N - The Bessel function order.

PROCEDURE:

The series expansion

$$I_n(x) = \text{SUM}_{(k=0 \rightarrow \text{inf})} [(x/2)^{(n+2k)} / k! \text{ Gamma}(n+k+1)]$$

is used, and is terminated when the k'th term is less than .001.

MODIFICATION HISTORY:

David L. Windt, Bell Laboratories, June 1993
windt@bell-labs.com

(See ./beseli_fract.pro)

BESELK_FRACT

[\[Previous Routine\]](#) [\[Next Routine\]](#) [\[List of Routines\]](#)

NAME :

BESELK_FRACT

PURPOSE :

This function returns the Modified Bessel Function of the Second Kind of order N, for any N, i.e., including fractional and negative orders.

CALLING SEQUENCE :

Result = BESELK_FRACT(X, N)

INPUTS :

X - The value for which the K Bessel function is required. X must be greater than 0. The result will have the same dimensions as X.

N - The Bessel function order.

PROCEDURE :

This function uses the BESELI_FRACT function:

Results=(BESELI_FRACT(X,-N)-BESELI_FRACT(X,B))*!PI/2./SIN(N*!PI)

MODIFICATION HISTORY:

David L. Windt, Bell Laboratories, June 1993
windt@bell-labs.com

(See ./beselk_fract.pro)

CHEM2LBL

[\[Previous Routine\]](#) [\[Next Routine\]](#) [\[List of Routines\]](#)

NAME :

CHEM2LBL

PURPOSE:

Convert a 'chemical name', i.e. a string containing characters and numbers - H2O, for example - into a string containing formatting commands so that the numbers become subscripts when using the result in IDL graphics.

For example: "H2O" would come back as "H!d2!nO".

CALLING SEQUENCE:

Result = CHEM2LBL(CHEMICAL)

INPUTS:

CHEMICAL - a string or string array specifying the chemical name(s).

KEYWORD PARAMETERS:

NOREFERENCE - if this keyword is set, text following an underscore character in CHEMICAL will be ignored. The default behavior is that any text following an underscore character will be surrounded by brackets (i.e. < >) and subscripted. For example, "SiO2_tetragonal" will be returned as "!nSiO!d2 <tetragonal>!n"

MODIFICATION HISTORY:

David L. Windt, March 1997
windt@bell-labs.com

June 2013: CHEMICAL can now be a string array.
davidwindt@gmail.com

(See ./chem2lbl.pro)

CHISQR

[\[Previous Routine\]](#) [\[Next Routine\]](#) [\[List of Routines\]](#)

NAME:

CHISQR

PURPOSE:

Compute the Chi Square statistic of a function and a fit to the function.

CALLING SEQUENCE:

```
Result=CHISQR(Y,SIGMA_Y,YFIT)
```

INPUTS:

Y - Input array.

SIGMA_Y - Uncertainty in Y.

YFIT - Fit to Y.

PROCEDURE:

```
CHISQR=TOTAL((Y-YFIT)^2/SIGMA_Y^2)
```

MODIFICATION HISTORY:

David L. Windt, Bell Labs, November 1989
windt@bell-labs.com

(See ./chisqr.pro)

CIRCLE_FIT

[\[Previous Routine\]](#) [\[Next Routine\]](#) [\[List of Routines\]](#)

NAME:

CIRCLE_FIT

PURPOSE:

Fit $y=f(x)$ where:
 $F(x) = y_c + \sqrt{r^2 - (x-x_c)^2}$
(x_c, y_c)=circle center, r =circle radius

CALLING SEQUENCE:

```
YFIT = CIRCLE_FIT(X,Y,A)
```

INPUTS:

X - independent variable, must be a vector.

Y - dependent variable, must have the same number of points ;
as x.

A - initial guess at fit coefficient [xc, yc, r]

OUTPUTS:

YFIT - fitted function.

OPTIONAL OUTPUT PARAMETERS:

A - Fit coefficients. a three element vector containing xc,
yc, and r.

MODIFICATION HISTORY:

Adapted from GAUSSFIT

D. L. Windt, davidwindt@gmail.com
Aug 2010

(See ./circle_fit.pro)

CLEAR

[\[Previous Routine\]](#) [\[Next Routine\]](#) [\[List of Routines\]](#)

NAME:

CLEAR

CATEGORY:

Stupid little convenience routines.

PURPOSE:

Clear the screen, just like the Unix command.

CALLING SEQUENCE:

CLEAR

RESTRICTIONS:

Only works on Unix platforms.

MODIFICATION HISTORY:

David L. Windt, Bell Labs, November 1989
windt@bell-labs.com

(See ./clear.pro)

COM_FIND

[\[Previous Routine\]](#) [\[Next Routine\]](#) [\[List of Routines\]](#)

NAME:

COM_FIND

PURPOSE:

Return the 'center-of-mass' of the supplied data array.

CALLING SEQUENCE:

Result = COM_FIND(X,Y)

INPUTS:

X, Y - 1D data arrays.

OUTPUTS:

Result = The X value corresponding to the center of mass, i.e., the X value that divides the integral of Y in half. Specifically, if X_com is the center-of-mass value, then

Integral(Y)_from_0_to_X_com = Integral(Y)_from_X_com_to_MAX(X)

KEYWORD PARAMETERS:

INTERPOLATE - Set this keyword to return a 'floating-point' index instead of an integer value.

X_com=COM_FIND(X,Y,/INTERPOLATE)

EXAMPLE:

Make some noisy data:

```
x=VECTOR(0.,5.,100)
y=SIN(x)+0.1*RANDOMN(seed,100)
```

Determine the center of mass:

```
x_com=COM_FIND(y)
```

MODIFICATION HISTORY:

David L. Windt, December 2003

windt@astro.columbia.edu

(See ./com_find.pro)

CONT_IMAGE

[\[Previous Routine\]](#) [\[Next Routine\]](#) [\[List of Routines\]](#)

NAME:

CONT_IMAGE

PURPOSE:

Overlay an image and a contour plot.

CALLING SEQUENCE:

CONT_IMAGE, IMAGE[,X,Y]

INPUTS:

IMAGE - 2 dimensional array to display.

OPTIONAL INPUTS:

X - 1 dimensional array of x-axis values.

Y - 1 dimensional array of y-axis values.

KEYWORD PARAMETERS:

WINDOW_SCALE - Set to scale the window size to the image size, otherwise the image size is scaled to the window size. Ignored when outputting to devices with scalable pixels.

ASPECT - Set to retain image's aspect ratio. Assumes square pixels. If ASPECT is set, the aspect ratio is retained.

INTERP - Set to bi-linear interpolate if image is resampled.

NOCONTOUR - Set to just display the image with plot axes.

INVERT - Set to invert the image scale, ie image=255-image

TOP - The maximum value of the scaled image. If not set, then it's set to (!d.n_colors < 255)-1.

MIN_VALUE - The minimum value of IMAGE to be displayed.

MAX_VALUE - The maximum value of IMAGE to be displayed.

COLORBAR - Set to display a color bar alongside the image.

BAR_TITLE - A text string to be used as the colorbar title if COLORBAR is set.

BAR_WIDTH - Width of the colorbar, in pixels. Default is 10 pixels for non-scalable pixel devices, or 2% of the plot width for scalable pixel devices.

BAR_OFFSET - Offset spacing between plot and colorbar. Default is 10.

NOAXIS - Set to inhibit drawing plot axes

NOSCALE - Set to inhibit scaling of input array.

NLEVELS - CONTOUR keyword.

MODIFICATION HISTORY:

Adapted (i.e. stolen) from IMAGE_CONT.

D. L. Windt, Bell Laboratories, Nov 1989.

April 1994:

Changed image scaling to go from 32 to !d.n_colors, so that
TEK_COLOR can be called to use first 32 colors for other plotting.

Added _EXTRA keyword.

March 1998 - Added TOP, MIN_VALUE, MAX_VALUE, COLORBAR, and
BAR_TITLE keywords. Also fixed quite a few bugs.
Note that setting the XSTYLE, XTYPE, YSTYLE, and
YTYPE keywords has no effect: these parameters
are always set to 0,1,0, and 1, respectively.

August 1998 - Plots are now drawn properly when !p.multi is
different from 0. Added BAR_OFFSET keyword.

windt@bell-labs.com

May 2011:

Changed scaling to go from 32 to !d.table_size-33
Added noaxis and noscale keywords

May 2013:

Added NLEVELS keyword, passed to CONTOUR

davidwindgt@gmail.com

(See ./cont_image.pro)

CONT_IMAGE2

[\[Previous Routine\]](#) [\[Next Routine\]](#) [\[List of Routines\]](#)

NAME:

CONT_IMAGE2

PURPOSE:

Display an image and overlay the contours from a second image.

CALLING SEQUENCE:

CONT_IMAGE2, IMAGE1, IMAGE2, X, Y

INPUTS:

IMAGE1 = Image to display.

IMAGE2 = Image from which contours are drawn.

KEYWORD PARAMETERS:

WINDOW_SCALE = set to scale the window size to the image size,
otherwise the image size is scaled to the window size.
Ignored when outputting to devices with scalable pixels.

ASPECT = set to retain image's aspect ratio. Assumes square
pixels. If /WINDOW_SCALE is set, the aspect ratio is
retained.

INTERP = set to bi-linear interpolate if image is resampled.

Plus IDL graphics keywords: XTITLE, YTITLE, SUBTITLE, TITLE

PROCEDURE:

If the device has scalable pixels then the image is written over the
plot window.

MODIFICATION HISTORY:

Adapted (i.e. stolen) from IMAGE_CONT

D. L. Windt, Bell Laboratories, June 1991.

April 1994:

Changed image scaling to go from 32 to !d.n_colors, so that
TEK_COLOR can be called to use first 32 colors for other plotting.
Added _EXTRA keyword.

windt@bell-labs.com

(See ./cont_image2.pro)

CURVE_LABEL

[\[Previous Routine\]](#) [\[Next Routine\]](#) [\[List of Routines\]](#)

NAME:

CURVE_LABEL

PURPOSE:

Draw labels close to one or more (up to 30) curves that have been previously plotted.

CALLING SEQUENCE:

```
CURVE_LABEL,X,Y1,[Y2,Y3,Y4,Y5],LABELS=LABELS, $  
          [COLOR=COLOR,XPOSITION=XPOSITION,YOFFSET=YOFFSET]
```

INPUTS:

X - xaxis vector (1D array)
Y1 - 1st y axis vector to be labelled.

OPTIONAL INPUTS:

Y2 - 2nd y axis vector to be labelled.
Y3 - 3rd y axis vector to be labelled.
....etc....

KEYWORD PARAMETERS:

LABELS - String array of labels. The size of the LABELS array must match the number of y variables passed. This keyword is required.

XPOSITION - A scalar variable specifying where along the x axis the labels are to be drawn, in normal coordinates. Default = 0.25. Unless the NO_REPOSITION keyword is set, this might get changed if the procedure determines that the labels are too close together when drawing multiple labels. Setting XPOSITION to -1 will inhibit drawing the curve label altogether.

YOFFSET - A scalar specifying the distance in Y between the labels and the curves, in normal coordinates. Default = 0.01

NO_REPOSITION - Set this to inhibit moving the label positions if the labels are too close together when drawing multiple labels.

COLOR - Integer array of color indices for the labels.

_EXTRA - The idl _EXTRA keyword, for additional graphics keywords to the XYOUTS procedure.

PROCEDURE:

All labels are lined up at one point along the xaxis. The procedure will try to find a position along the xaxis for which the labels are not too close to each other. If it fails at this, it will just stick the labels at x=0.25 (normal).

EXAMPLE:

To label a plot containing three curves, try something like this:

```
plot,x,y1,/nodata
oplot,x,y1,color=2
oplot,x,y2,color=3
oplot,x,y3,color=4
curve_label,x,y1,y2,y3,labels=['Y1','Y2','Y3'],color=[2,3,4]
```

MODIFICATION HISTORY:

David L. Windt, Bell Labs, March, 1997.
windt@bell-labs.com

February, 1998 - Added the ability to inhibit labelling
the curve by specifying a value of -1 for XPOSITION.

(See `./curve_label.pro`)

CW_BGROUP_RXO

[\[Previous Routine\]](#) [\[Next Routine\]](#) [\[List of Routines\]](#)

NAME:

CW_BGROUP_RXO

PURPOSE:

CW_BGROUP_RXO is a compound widget that simplifies creating a base of buttons. It handles the details of creating the proper base (standard, exclusive, or non-exclusive) and filling in the desired buttons. Events for the individual buttons are handled transparently, and a CW_BGROUP_RXO event returned. This event can return any one of the following:

- The Index of the button within the base.
- The widget ID of the button.
- The name of the button.
- An arbitrary value taken from an array of User values.

CATEGORY:

Compound widgets.

CALLING SEQUENCE:

Widget = CW_BGROUP_RXO(Parent, Names)

To get or set the value of a CW_BGROUP_RXO, use the GET_VALUE and SET_VALUE keywords to WIDGET_CONTROL. The value of a CW_BGROUP_RXO is:

Type	Value
normal	None
exclusive	Index of currently set button

non-exclusive Vector indicating the position
 of each button (1-set, 0-unset)

INPUTS:

Parent: The ID of the parent widget.
Names: A string array, containing one string per button,
 giving the name of each button.

KEYWORD PARAMETERS:

BUTTON_UVALUE: An array of user values to be associated with
 each button and returned in the event structure.
COLUMN: Buttons will be arranged in the number of columns
 specified by this keyword.
EVENT_FUNCT: The name of an optional user-supplied event function
 for buttons. This function is called with the return
 value structure whenever a button is pressed, and
 follows the conventions for user-written event
 functions.
EXCLUSIVE: Buttons will be placed in an exclusive base, with
 only one button allowed to be selected at a time.
FONT: The name of the font to be used for the button
 titles. If this keyword is not specified, the default
 font is used.
FRAME: Specifies the width of the frame to be drawn around
 the base.
GRID: Buttons will be arranged on a uniform grid.
IDS: A named variable into which the button IDs will be
 stored, as a longword vector.
LABEL_LEFT: Creates a text label to the left of the buttons.
LABEL_TOP: Creates a text label above the buttons.
MAP: If set, the base will be mapped when the widget
 is realized (the default).
NONEXCLUSIVE: Buttons will be placed in an non-exclusive base.
 The buttons will be independent.
NO_RELEASE: If set, button release events will not be returned.
RETURN_ID: If set, the VALUE field of returned events will be
 the widget ID of the button.
RETURN_INDEX: If set, the VALUE field of returned events will be
 the zero-based index of the button within the base.
 THIS IS THE DEFAULT.
RETURN_NAME: If set, the VALUE field of returned events will be
 the name of the button within the base.
ROW: Buttons will be arranged in the number of rows
 specified by this keyword.
SCROLL: If set, the base will include scroll bars to allow
 viewing a large base through a smaller viewport.
SET_VALUE: The initial value of the buttons. This is equivalent
 to the later statement:

WIDGET_CONTROL, widget, set_value=value

SPACE: The space, in pixels, to be left around the edges
 of a row or column major base. This keyword is
 ignored if EXCLUSIVE or NONEXCLUSIVE are specified.

UVALUE: The user value to be associated with the widget.
UNAME: The user name to be associated with the widget.
XOFFSET: The X offset of the widget relative to its parent.
XPAD: The horizontal space, in pixels, between children
 of a row or column major base. Ignored if EXCLUSIVE
 or NONEXCLUSIVE are specified.
XSIZE: The width of the base.
X_SCROLL_SIZE: The width of the viewport if SCROLL is specified.
YOFFSET: The Y offset of the widget relative to its parent.
YPAD: The vertical space, in pixels, between children of
 a row or column major base. Ignored if EXCLUSIVE
 or NONEXCLUSIVE are specified.
YSIZE: The height of the base.
Y_SCROLL_SIZE: The height of the viewport if SCROLL is specified.

OUTPUTS:

 The ID of the created widget is returned.

SIDE EFFECTS:

 This widget generates event structures with the following definition:

 event = { ID:0L, TOP:0L, HANDLER:0L, SELECT:0, VALUE:0 }

 The SELECT field is passed through from the button event. VALUE is
 either the INDEX, ID, NAME, or BUTTON_UVALUE of the button,
 depending on how the widget was created.

RESTRICTIONS:

 Only buttons with textual names are handled by this widget.
 Bitmaps are not understood.

MODIFICATION HISTORY:

 15 June 1992, AB
 7 April 1993, AB, Removed state caching.
 6 Oct. 1994, KDB, Font keyword is not applied to the label.
 10 FEB 1995, DJC fixed bad bug in event procedure, getting
 id of stash widget.
 11 April 1995, AB Removed Motif special cases.
 Feb 2004 DLW, Added GRID keyword

 May 2013 - Renamed CW_BGROUP_RXO, DLW, davidwindt@gmail.com

(See ./cw_bgroup_rxo.pro)

CW_CURVE_LABEL

[\[Previous Routine\]](#) [\[Next Routine\]](#) [\[List of Routines\]](#)

NAME:

 CW_CURVE_LABEL

PURPOSE:

A compound widget used to select the position for a curve label; this widget is intended to be used in conjunction with the CURVE_LABEL procedure in this directory, in that this widget lets the user select a value from a slider from 0 to one, corresponding to the XPOSITION keyword in CURVE_LABEL.

CATEGORY:

Compound widgets.

CALLING SEQUENCE:

Result = CW_CURVE_LABEL(PARENT)

INPUTS:

PARENT - The ID of the parent widget.

KEYWORD PARAMETERS:

UVALUE - Supplies the user value for the widget.

VALUE - Initial value for the widget: a floating point between 0 and 1, corresponding to the XPOSITION keyword in CURVE_LABEL.

TITLE - A title for the widget.

FRAME - Set to draw a frame around the widget; ignored if PARENT is present.

FORMAT - Format string for CW_FSLIDER (default is F5.3)

FONT - Fonts to use for labels and buttons.

DONE - Set this to add a Done button, in addition to the standard Apply button.

YPAD, SPACE - keywords to widget_base

OUTPUTS:

The ID of the created widget is returned.

PROCEDURE/EXAMPLE:

A slider widget is created in which the user can select a position value. By pressing the "Apply" button, an event is returned, allowing the calling procedure to redraw the curve label if desired.

This widget generates an event when the user presses the Apply button or the Done button, if present. The EVENT.TAG keyword will return either "APPLY" or "DONE" accordingly.

MODIFICATION HISTORY:

David L. Windt, Bell Labs, April 1997
windt@bell-labs.com

July 2003: Added YPAD, SPACE keywords

(See ./cw_curve_label.pro)

CW_DRAWSIZE

[\[Previous Routine\]](#) [\[Next Routine\]](#) [\[List of Routines\]](#)

NAME:

CW_DRAWSIZE

PURPOSE:

A compound widget used to change the size of an existing draw widget. The widget contains fields for the X and Y draw size (in pixels), an Apply button, and optionally a Done button.

CATEGORY:

Compound widgets.

CALLING SEQUENCE:

Result = CW_DRAWSIZE(PARENT,DRAW_WIDGET)

INPUTS:

PARENT - The ID of the parent widget.

DRAW_WIDGET - The id of the draw widget being resized.

KEYWORD PARAMETERS:

UVALUE - Supplies the user value for the widget.

FRAME - set to draw a frame around the widget; ignored if
PARENT is present.

ROW - set to place the two window size fields (x,y) in a row.

COLUMN - set to place the two window size fields (x,y) in a column.

FONT - fonts to use for labels and buttons.

DONE - set this to add a Done button, in addition to the standard
Apply button.

NO_RETURN - The default behavior is that the user must press <return> after entering new values. Set this keyword so that an event is returned even if the user just changes a value and then moves the cursor outside of the text entry area.

OUTPUTS:

The ID of the created widget is returned.

PROCEDURE/EXAMPLE:

A widget is created in which the user can specify the X and Y draw widget size in pixels. By pressing the "Apply" button, the draw widget is resized, and an event is returned, allowing the calling procedure to repaint the window if desired.

This widget generates an event when the user presses the Apply button or the Done button, if present. The EVENT.TAG keyword will return either "APPLY" or "DONE" accordingly.

MODIFICATION HISTORY:

David L. Windt, Bell Labs, March 1997
windt@bell-labs.com

DLW, June 1997, Added NO_RETURN keyword.

DLW, Sep 1997, Fixed bug that caused initial values of X and Y pixel sizes to be displayed as floating point values rather than integers.

(See ./cw_drawsize.pro)

CW_FIELD_RXO

[\[Previous Routine\]](#) [\[Next Routine\]](#) [\[List of Routines\]](#)

NAME:

CW_FIELD_RXO

PURPOSE:

This widget cluster function manages a data entry field widget. The field consists of a label and a text widget. CW_FIELD's can be string fields, integer fields or floating-point fields. The default is an editable string field.

CATEGORY:

Widget Clusters.

CALLING SEQUENCE:

Result = CW_FIELD_RXO(Parent)

INPUTS:

Parent: The widget ID of the widget to be the field's parent.

KEYWORD PARAMETERS:

TITLE: A string containing the text to be used as the label for the field. The default is "Input Field:".

VALUE: The initial value in the text widget. This value is automatically converted to the type set by the STRING, INTEGER, and FLOATING keywords described below.

UVALUE: A user value to assign to the field cluster. This value can be of any type.

UNAME: A user supplied string name to be stored in the widget's user name field.

FRAME: The width, in pixels, of a frame to be drawn around the entire field cluster. The default is no frame.

RETURN_EVENTS: Set this keyword to make cluster return an event when a <CR> is pressed in a text field. The default is not to return events. Note that the value of the text field is always returned when the WIDGET_CONTROL, field, GET_VALUE=X command is used.

ALL_EVENTS: Like RETURN_EVENTS but return an event whenever the contents of a text field have changed.

COLUMN: Set this keyword to center the label above the text field. The default is to position the label to the left of the text field.

ROW: Set this keyword to position the label to the left of the text field. This is the default.

XSIZE: An explicit horizontal size (in characters) for the text input area. The default is to let the window manager size the widget. Using the XSIZE keyword is not recommended.

YSIZE: An explicit vertical size (in lines) for the text input area. The default is 1.

STRING: Set this keyword to have the field accept only string values. Numbers entered in the field are converted to their string equivalents. This is the default.

FLOATING: Set this keyword to have the field accept only floating-point values. Any number or string entered is converted to its floating-point equivalent.

INTEGER: Set this keyword to have the field accept only integer values. Any number or string entered is converted to its integer equivalent (using FIX). For example, if 12.5 is entered in this type of field, it is converted to 12.

LONG: Set this keyword to have the field accept only long integer values. Any number or string entered is converted to its long integer equivalent (using LONG).

FONT: A string containing the name of the X Windows font to use for the TITLE of the field.

FIELDFONT: A string containing the name of the X Windows font to use for the TEXT part of the field.

NOEDIT: Normally, the value in the text field can be edited. Set this keyword to make the field non-editable.

NO_RETURN: The default behavior is that the user must press <return> after entering new values. Set this keyword so that an event is returned even if the user just changes a value and then moves the cursor outside of the text entry area.

TEXT_ID: The widget id of the text widget.

UNITS: A string to be placed after the text entry box.

RIGHT_ALIGN: Set this keyword for the field text to be right-aligned within the field.

RESOURCE_NAME: The X windows system RESOURCE_NAME keyword (as in the WIDGET_TEXT routine), which only applies to the text widget.

SPACE: Keyword to widget_base.

XPAD: Keyword to widget_base.

YPAD: Keyword to widget_base.

FORMAT: Valid format code for numerical field display. e.g., FORMAT='(F4.1)'

OUTPUTS:

This function returns the widget ID of the newly-created cluster.

COMMON BLOCKS:

None.

PROCEDURE:

Create the widgets, set up the appropriate event handlers, and return the widget ID of the newly-created cluster.

EXAMPLE:

The code below creates a main base with a field cluster attached to it. The cluster accepts string input, has the title "Name:", and has a frame around it:

```
base = WIDGET_BASE()
field = CW_FIELD_RXO(base, TITLE="Name:", /FRAME)
WIDGET_CONTROL, base, /REALIZE
```

MODIFICATION HISTORY:

Written by: Keith R. Crosley June 1992
KRC, January 1993 -- Added support for LONG
 integers.
AB, 7 April 1993, Removed state caching.
JWG, August 1993, Completely rewritten to make
 use of improved TEXT widget functionality
ACY, 25 March, 1994, fix usage of FRAME keyword
KDB, May 1994, Initial value =0 would result
 in a null text field. Fixed
 keyword check.
CT, RSI, March 2001: Pass keywords directly into WIDGET_BASE,
 without assigning default values, since the defaults are
 handled by WIDGET_BASE. Avoids assigning defaults if user passes
 in undefined variables.
CT, RSI, July 2001: Fix bug in previous mod. If user passes in a
 numeric VALUE but forgets to set the /FLOAT, we still need
 to convert to a string before passing onto WIDGET_TEXT.

David L. Windt, Columbia Univ., April 2003

Modified CW_FIELD.PRO (IDL 5.6 version) to include NO_RETURN,
TEXT_ID, UNITS, RESOURCE_NAME, RIGHT_ALIGN, SPACE, XPAD and YPAD
keywords, and implemented workaround to deal with widget bug when
using the NO_RETURN keyword on some platforms.

May 2013 - Renamed CWFIELD_RXO, DLW, davidwindt@gmail.com

(See `./cw_field_rxo.pro`)

CW_FSLIDER_RXO

[\[Previous Routine\]](#) [\[Next Routine\]](#) [\[List of Routines\]](#)

NAME:

CW_FSLIDER_RXO

PURPOSE:

The standard slider provided by the WIDGET_SLIDER() function is integer only. This compound widget provides a floating point slider.

CATEGORY:

Compound widgets.

CALLING SEQUENCE:

widget = CW_FSLIDER_RXO(Parent)

INPUTS:

Parent: The ID of the parent widget.

KEYWORD PARAMETERS:

DRAW: Set this keyword to zero if events should only

be generated when the mouse is released. If it is non-zero, events will be generated continuously when the slider is adjusted. Note: On slow systems, /DRAG performance can be inadequate. The default is DRAG=0.

EDIT: Set this keyword to make the slider label be editable. The default is EDIT=0.

EVENT_FUNC: The name of an optional user-supplied event function for events. This function is called with the return value structure whenever the slider value is changed, and follows the conventions for user-written event functions.

FORMAT: Provides the format in which the slider value is displayed. This should be a format as accepted by the STRING procedure. The default is FORMAT='(G13.6)'

FRAME: Set this keyword to have a frame drawn around the widget. The default is FRAME=0.

MAXIMUM: The maximum value of the slider. The default is MAXIMUM=100.

MINIMUM: The minimum value of the slider. The default is MINIMUM=0.

SCROLL Sets the SCROLL keyword to the WIDGET_SLIDER underlying this compound widget. Unlike WIDGET_SLIDER, the value given to SCROLL is taken in the floating units established by MAXIMUM and MINIMUM, and not in pixels.

SUPPRESS_VALUE: If true, the current slider value is not displayed. The default is SUPPRESS_VALUE=0.

TITLE: The title of slider. (The default is no title.)

UVALUE: The user value for the widget.

UNAME: The user name for the widget.

VALUE: The initial value of the slider

VERTICAL: If set, the slider will be oriented vertically. The default is horizontal.

XSIZE: For horizontal sliders, sets the length.

YSIZE: For vertical sliders, sets the height.

COMPACT: For horizontal sliders, the label is placed inline with the value

OUTPUTS:

The ID of the created widget is returned.

SIDE EFFECTS:

This widget generates event structures containing a field named value when its selection thumb is moved. This is a floating point value.

PROCEDURE:

WIDGET_CONTROL, id, SET_VALUE=value can be used to change the current value displayed by the widget. Optionally, the value supplied to the SET_VALUE keyword can be a three element vector consisting of [value, minimum, maximum] in order to change the minimum and maximum values as well as the slider value itself.

WIDGET_CONTROL, id, GET_VALUE=var can be used to obtain the current value displayed by the widget. The maximum and minimum values of the slider can also be obtained by calling the

FSLIDER_GET_VALUE function directly (rather than the standard usage through the WIDGET_CONTROL interface) with the optional keyword MINMAX:

```
sliderVals = FSLIDER_GET_VALUE(id, /MINMAX)
```

When called directly with the MINMAX keyword, the return value of FSLIDER_GET_VALUE is a three element vector containing [value, minimum, maximum].

MODIFICATION HISTORY:

April 2, 1992, SMR and AB

Based on the RGB code from XPALETTE.PRO, but extended to support color systems other than RGB.

5 January 1993, Mark Rivers, Brookhaven National Labs

Added EDIT keyword.

7 April 1993, AB, Removed state caching.

28 July 1993, ACY, set_value: check labelid before setting text.

3 October 1995, AB, Added SCROLL keyword.

15 July 1998, ACY, Added ability to set and get minimum and maximum.

24 July 2000, KDB, Fixed scroll keyword modification.

March 2001, CT, RSI: Add double precision. Store value internally, separate from either scrollbar value or text label value.

May 2011, D. Windt

Changed title layout

May 2013 - Renamed CW_FSLIDER_RXO, DLW, davidwindt@gmail.com

(See ./cw_fslider_rxo.pro)

CW_LEGEND_RXO

[\[Previous Routine\]](#) [\[Next Routine\]](#) [\[List of Routines\]](#)

NAME:

CW_LEGEND_RXO

PURPOSE:

A compound widget used to set values for the POSITION, NOBOX, and BOXFILL keywords to the LEGEND_RXO procedure.

CATEGORY:

Compound widgets.

CALLING SEQUENCE:

Result = CW_LEGEND_RXO(PARENT,LABEL)

INPUTS:

PARENT - The ID of the parent widget.

LABEL - a label to be drawn to the left (or top, for /column) of the widget.

OPTIONAL KEYWORD PARAMETERS:

ADD_LABEL_CURVE_OPTION - set to add "Label Curves" as an additional, last option in the list of legend positions. (Return value for Label Curves is 13.)

UVALUE - Supplies the user value for the widget.

VALUE - an array of initial values:
[POSITION, NOBOX, BOXFILL]

FRAME - set to draw a frame around the widget.

FONT - font keyword for labels etc.

XPAD, YPAD, SPACE - keywords to widget_base

OUTPUTS:

The ID of the created widget is returned.

COMMON BLOCKS:

CW_PLOTSTYLE: private common block containing color bitmaps for 'buttons' and menus, and arrays of valid values for thick, psym and symsize.

RESTRICTIONS:

Uses the cgImage command in place of the tv command to display images correctly on all devices. cgImage is part of the Coyote Graphics System at www.idlcoyote.com.

PROCEDURE/EXAMPLE:

MODIFICATION HISTORY:

David L. Windt, Reflective X-ray Optics, May 2013
davidwindt@gmail.com.

-

(See ./cw_legend_rxo.pro)

CW_PLOTAXES

[\[Previous Routine\]](#) [\[Next Routine\]](#) [\[List of Routines\]](#)

NAME:

CW_PLOTAXES

PURPOSE:

A compound widget used to change the type, range and style (bit 0) values of one or more plot axis structure variable. A CW_PLOTAXIS (single axis) widget is created for each element of the LABELS input parameter. The widget also includes an Apply button, and (optionally) a Done button.

CATEGORY:

Compound widgets.

CALLING SEQUENCE:

Result = CW_PLOTAXES(PARENT,LABELS)

INPUTS:

PARENT - The ID of the parent widget.

LABELS - A string array of labels to be drawn to the left (or top) of each of the CW_PLOTAXIS widgets.

OPTIONAL KEYWORD PARAMETERS:

UVALUE - Supplies the user value for the widget.

FRAME - Set to draw a frame around the widget.

VALUE - An (n,4) array of initial values, where
n = n_elements(LABELS), and each row has
the form VALUE(i,*)=[type,min,max,style]

FONT - Font keyword for labels etc.

ROW - Set to create a row of column-oriented CW_PLOTAXIS widgets.

COLUMN - Set to create a column of row-oriented CW_PLOTAXIS
widgets. (default)

DONE - Set this to add a Done button, in addition to the standard
Apply button.

AXIS_IDS - An array of widget id's for the individual
CW_PLOTAXIS widgets.

X_SCROLL_SIZE, Y_SCROLL_SIZE - if these values are non-zero,
then the base widget which
holds the CW_PLOTAXIS widgets

will include scroll bars.

NO_RETURN - The default behavior is that the user must press <return> after entering new values. Set this keyword so that new values are accepted even if the user just changes a value and then moves the cursor outside of the text entry area.

XPAD, YPAD, SPACE - keywords to widget_base

OUTPUTS:

The ID of the created widget is returned.

PROCEDURE/EXAMPLE:

The idea is that this cw would be used in a widget intended to allow the user to interactively adjust the settings for a plot. For instance, you might have a menu item such as Plot Options->Scaling, which would create a popup widget containing a CW_PLOTAXES subwidget for the X and Y plot variables. When the user makes changes to the Type, Range, and Style values, and then presses the Apply button, the popup widget event handler would re-draw the plot accordingly.

This widget generates an event when the user presses the Apply button or the Done button, if present. The EVENT.TAG keyword will return either "APPLY" or "DONE" accordingly.

Example:

```
axes=CW_PLOTAXES(BASE,['X','Y'],/DONE, $  
                VALUE=TRANPOSE([[FLTARR(4)],[FLTARR(4)]])
```

MODIFICATION HISTORY:

David L. Windt, Bell Labs, March 1997
windt@bell-labs.com

DLW, June 1997, Added NO_RETURN keyword.

July 2003: Added YPAD, SPACE keywords

January 2004: Added GRID keywords

(See ./cw_plotaxes.pro)

CW_PLOTAXIS

[\[Previous Routine\]](#) [\[Next Routine\]](#) [\[List of Routines\]](#)

NAME:

CW_PLOTAXIS

PURPOSE:

A compound widget used to change the type, range and style (bit 0 only) values of plot axis structure variable.

CATEGORY:

Compound widgets.

CALLING SEQUENCE:

Result = CW_PLOTAXIS(PARENT,LABEL)

INPUTS:

PARENT - The ID of the parent widget.

LABEL - a label to be drawn to the left of the widget.

OPTIONAL KEYWORD PARAMETERS:

UVALUE - Supplies the user value for the widget.

FRAME - set to draw a frame around the widget.

VALUE - a 4-element array of the form [type,min,max,style].
type and min,max correspond to the !axis.type and
!axis.range variables, and style is bit 0 of
!axis.style.

FONT - font keyword for labels etc.

ROW - set to orient the subwidgets in a row (default.)

COLUMN - set to orient the subwidgets in a column.

NO_RETURN - The default behavior is that the user must press
<return> after entering new values. Set this
keyword so that new values are accepted even if
the user just changes a value and then moves the
cursor outside of the text entry area.

XPAD, YPAD, SPACE - keywords to widget_base

GRID - Set this to also add widgets to set the grid (i.e.,
ticklen) and gridstyle values.

NOGRID - Set this to draw the grid widgets but never map
them. Useful when aligning multiple cw_plotaxis
widgets.

OUTPUTS:

The ID of the created widget is returned.

COMMON BLOCKS:

CW_PLOTSTYLE: private common block containing color bitmaps for 'buttons' and menus, and arrays of valid values for thick, psym and symsize.

RESTRICTIONS:

Uses the cgImage command in place of the tv command to display images correctly on all devices. cgImage is part of the Coyote Graphics System at www.idlcoyote.com.

PROCEDURE/EXAMPLE:

The idea is that one or more instances of this cw would be used in a widget intended to allow the user to interactively adjust the settings for a plot. For instance, you might have a menu item such as Plot Options->Scaling, which would create a popup widget containing CW_PLOTAXIS subwidgets for the X and Y plot variables. When the user makes changes to the Type, Range, and Style values, the popup widget event handler would re-draw the plot accordingly.

The widget returns events when any of it's children generate events. The returned event has the form
{CW_PLOTAXIS_EVENT, ID:id, TOP:top, HANDLER:handler, TAG:tag}
where TAG indicates which child widget generated the event: possible values for EVENT.TAG are TYPE, MIN, MAX, and STYLE. If GRID is set, then GRID and GRIDSTYLE tags are possible as well.

MODIFICATION HISTORY:

David L. Windt, Bell Labs, March 1997
windt@bell-labs.com

DLW, June 1997, Added NO_RETURN keyword.

DLW, November 1997, Text fields for Range values are now updated when the user makes a change; the specified values are converted to floating point.

January 2004: Added XPAD, YPAD, SPACE keywords; Changed widget types. Added GRID keyword. Changed to double precision.

May 2013: Added color bitmaps for gridstyle options.

(See ./cw_plotaxis.pro)

CW_PLOTLABEL

[\[Previous Routine\]](#) [\[Next Routine\]](#) [\[List of Routines\]](#)

NAME:

CW_PLOTLABEL

PURPOSE:

A compound widget used to select the position for a plot label or legend. This widget is intended to be used in conjunction with the PLOT_TEXT or LEGEND procedures in this directory, in that this widget lets the user select one of 13 pre-defined positions corresponding to the POSITION keyword in PLOT_TEXT and LEGEND.

CATEGORY:

Compound widgets.

CALLING SEQUENCE:

Result = CW_PLOTLABEL(PARENT)

INPUTS:

PARENT - The ID of the parent widget.

KEYWORD PARAMETERS:

UVALUE - Supplies the user value for the widget.

VALUE - initial value for the widget: an integer between 0 and 12, corresponding to the POSITION keyword in plot_text or legend.

TITLE - a title for the widget.

FRAME - set to draw a frame around the widget; ignored if PARENT is present.

FONT - fonts to use for labels and buttons.

DONE - set this to add a Done button, in addition to the standard Apply button.

NO_BELOW - set this to inhibit drawing the three buttons that correspond to label positions below the plot, i.e., position values of 1, 2 and 3.

YPAD, SPACE - keywords to widget_base

OUTPUTS:

The ID of the created widget is returned.

PROCEDURE/EXAMPLE:

A widget is created in which the user can select one of 13 position values. By pressing the "Apply" button, an event is returned, allowing the calling procedure to redraw the plot label or legend if desired.

This widget generates an event when the user presses the Apply button or the Done button, if present. The EVENT.TAG keyword will return either "APPLY" or "DONE" accordingly.

MODIFICATION HISTORY:

David L. Windt, Bell Labs, April 1997
windt@bell-labs.com

July 2003: Added YPAD, SPACE keywords

(See `./cw_plotlabel.pro`)

CW_PLOTSTYLE

[\[Previous Routine\]](#) [\[Next Routine\]](#) [\[List of Routines\]](#)

NAME:

CW_PLOTSTYLE

PURPOSE:

A compound widget used to set values for the graphics keywords COLOR, LIFESTYLE, THICK, PSYM, and SYMSIZE.

CATEGORY:

Compound widgets.

CALLING SEQUENCE:

Result = CW_PLOTSTYLE(PARENT,LABEL)

INPUTS:

PARENT - The ID of the parent widget.

LABEL - a label to be drawn to the left (or top, for /column) of the widget.

OPTIONAL KEYWORD PARAMETERS:

UVALUE - Supplies the user value for the widget.

VALUE - an array of initial values:
[COLOR,LINestyle,THICK,PSYM,SYMSIZE]

FRAME - set to draw a frame around the widget.

FONT - font keyword for labels etc.

ROW - set to orient the subwidgets in a row (default.)

COLUMN - set to orient the subwidgets in a column.

NO_PSYM - set to omit the PSYM and SYMSIZE widgets. If NO_SYM is set, then PSYM=0 and SYMSIZE=0 will be returned when using WIDGET_CONTROL,GET_VALUE; PSYM and SYMSIZE are ignored when using WIDGET_CONTROL,SET_VALUE

XPAD, YPAD, SPACE - keywords to widget_base

INIT_ONLY - set to create the 24-bit bitmaps and value arrays that are stored in the private CW_PLOTSTYLE common block listed below.

OUTPUTS:

The ID of the created widget is returned.

COMMON BLOCKS:

CW_PLOTSTYLE: private common block containing color bitmaps for 'buttons' and menus, and arrays of valid values for thick, psym and symsize.

RESTRICTIONS:

Uses the cgImage command in place of the tv command to display images correctly on all devices. cgImage is part of the Coyote Graphics System at www.idlcoyote.com.

PROCEDURE/EXAMPLE:

The idea is that one or more instances of this cw would be used in a widget intended to allow the user to interactively adjust the settings for a plot. For instance, you might have a menu item such as Plot Options->Styles, which would create a popup widget containing CW_PLOTSTYLE subwidgets for each of the variables being plotted. When the user makes changes to the Color, Linestyle, Thick, Psym, and Symsize values, the popup widget event handler would re-draw the plot accordingly.

The user is presented with pulldown menus displaying:

- 32 color choices corresponding to the 1st 32 color table entries.
- 6 linestyle choices (linestyle=0 to 5)

- 9 thickness choices (thick=1 to 9)
- 35 psym choices, corresponding to the 18 symbols obtained using the SYM function with and without lines. Note that the value of psym returned by this widget is intended therefore to be used with the SYM function.
- 8 symsize choices (symsize=0.25 to 2.00, in 0.25 increments)

The widget returns events when any of it's children generate events. The returned event has the form
 {CW_PLOTSTYLE_EVENT, ID:id, TOP:top, HANDLER:handler, TAG:tag}
 where TAG indicates which child widget generated the event:
 Possible values for EVENT.TAG are COLOR, LINESYLE, THICK
 PSYM, and SYMSIZE.

MODIFICATION HISTORY:

David L. Windt, Bell Labs, March 1997
 windt@bell-labs.com

January 2004: Added XPAD, YPAD, SPACE keywords, and added draw widgets to display results.

May 2013: Re-written to use 24-bit color bitmap buttons and pull-down menus in place of draw and droplist widgets. No longer reliant on using TEK_COLOR; load whatever colors you like in the first 32 color indices of the color table.

DLW, RXO, davidwindt@gmail.com.

-

(See ./cw_plotstyle.pro)

CW_PLOTSTYLES

[\[Previous Routine\]](#) [\[Next Routine\]](#) [\[List of Routines\]](#)

NAME:

CW_PLOTSTYLES

PURPOSE:

A compound widget used to set values for the graphics keywords COLOR, LINESYLE, THICK, PSYM, and SYMSIZE, for several plot variables. A CW_PLOTSTYLE (single variable) widget is created for each element of the LABELS input parameter. The widget also includes an Apply button, and (optionally) a Done button.

CATEGORY:

Compound widgets.

CALLING SEQUENCE:

```
Result = CW_PLOTSTYLES(PARENT,LABEL)
```

INPUTS:

PARENT - The ID of the parent widget.

LABELS - a string array of labels to be drawn to the left
(or top) of each of the CW_PLOTSTYLE widgets.

OPTIONAL KEYWORD PARAMETERS:

UVALUE - Supplies the user value for the widget.

VALUE - an (n,5) array of initial values, where
n = n_elements(LABELS), and each row has
the form VALUE(i,*)=[color,linestyle,thick,psym,symsize]

FRAME - set to draw a frame around the widget.

FONT - font keyword for labels etc.

ROW - set to create a row of column-oriented CW_PLOTSTYLE widgets.

COLUMN - set to create a column of row-oriented CW_PLOTSTYLE
widgets. (default)

DONE - set this to add a Done button, in addition to the standard
Apply button.

STYLE_IDS - an array of widget id's for the individual
CW_PLOTSTYLE widgets.

X_SCROLL_SIZE, Y_SCROLL_SIZE - if these values are non-zero,
then the base widget which
holds the CW_PLOTSTYLE widgets
will include scroll bars.

XPAD, YPAD, SPACE - keywords to widget_base

OUTPUTS:

The ID of the created widget is returned.

PROCEDURE/EXAMPLE:

The idea is that this compound widget would be used in a
widget intended to allow the user to interactively adjust
the style settings for several variables contained in a
plot. For instance, you might have a menu item such as Plot
Options->Styles, which would create a popup widget

containing a CW_PLOTSTYLES subwidget, allowing the user to affect each of the variables in the plot. When the user makes changes to the Color, Linestyle, Thick, Psym, and Symsize values, the popup widget event handler would re-draw the plot accordingly.

This widget generates an event when the user presses the Apply button or the Done button, if present. The EVENT.TAG keyword will return either "APPLY" or "DONE" accordingly.

Example:

```
style=CW_PLOTSTYLE(BASE,['A','B'],/DONE, $
                    VALUE=TRANPOSE([[FLTARR(5)],[FLTARR(5)]])
```

MODIFICATION HISTORY:

David L. Windt, Bell Labs, March 1997
windt@bell-labs.com

July 2003: Added XPAD, YPAD, SPACE keywords

(See ./cw_plotstyles.pro)

CW_PLOTTITLE_CHAR

[\[Previous Routine\]](#) [\[Next Routine\]](#) [\[List of Routines\]](#)

NAME:

CW_PLOTTITLE_CHAR

PURPOSE:

A compound widget used to set values for the graphics keywords CHARSIZE, SUBTITLE, and TITLE. The widget contains fields for these parameters, an Apply button, and (optionally) a Done button.

CATEGORY:

Compound widgets.

CALLING SEQUENCE:

Result = CW_PLOTTITLE_CHAR(PARENT)

INPUTS:

PARENT - The ID of the parent widget.

OPTIONAL KEYWORD PARAMETERS:

UVALUE - Supplies the user value for the widget.

FRAME - set to draw a frame around the widget.

VALUE - a structure, containing initial values for the charsize, subtitle and title fields, of the form {charsize:_float_, subtitle: _string_, title:_string_}

FONT - font keyword for labels etc.

DONE - set this to add a Done button, in addition to the standard Apply button.

IDS - widget ids of the title, subtitle, and charsize cw_field_rxo widgets, and the apply button widget.

NO_RETURN - The default behavior is that the user must press <return> after entering new values. Set this keyword so that new values are accepted even if the user just changes a value and then moves the cursor outside of the text entry area.

YPAD, SPACE - keywords to widget_base

OUTPUTS:

The ID of the created widget is returned.

PROCEDURE/EXAMPLE:

The idea is that this cw would be used in a widget intended to allow the user to interactively adjust the settings for a plot. For instance, you might have a menu item such as Plot Options->Titles/Charsize, which would create a popup widget containing a CW_PLOTTITLE_CHAR subwidget. When the user makes changes to the CW_PLOTTITLE_CHAR fields, and then presses the Apply button, the popup widget event handler would re-draw the plot accordingly.

This widget generates an event when the user presses the Apply button or the Done button, if present. The EVENT.TAG keyword will return either "APPLY" or "DONE" accordingly.

Example:

```
title_char=CW_PLOTTITLE_CHAR(BASE,/DONE, $
                           VALUE={CHARSIZE:!P.CHARSIZE,SUBTITLE:!P.SUBTITLE,
                                   TITLE:!P.TITLE})
```

MODIFICATION HISTORY:

David L. Windt, Bell Labs, March 1997
windt@bell-labs.com

DLW, June 1997, Added NO_RETURN keyword.

July 2003: Added YPAD, SPACE keywords

(See ./cw_plottitle_char.pro)

CW_VECTOR

[\[Previous Routine\]](#) [\[Next Routine\]](#) [\[List of Routines\]](#)

NAME:

CW_VECTOR

PURPOSE:

A compound widget used to get input necessary to create a "vector", in the spirit of the VECTOR function in this directory, i.e, get input for the MIN, MAX, and PTS values. The widget also lets the user specify the increment between points, and whether the point spacing is linear or logarithmic.

CATEGORY:

compound widgets.

CALLING SEQUENCE:

Result = CW_VECTOR(PARENT)

INPUTS:

PARENT - the id of the parent widget.

OPTIONAL KEYWORD PARAMETERS:

UVALUE - Supplies the user value for the widget.

FRAME - Set to draw a frame around the widget.

FONT - Font keyword for labels etc.

TITLE - A string used to label the widget

XSIZE - An explicit horizontal size (in characters) for the min, max and increment input areas. The default is to let the window manager size the widget.

NXSIZE - An explicit horizontal size (in characters) for the pts field area. The default is to let the window manager size the widget.

YSIZE - An explicit vertical size (in lines) for the text input

areas. The default is 1.

VALUE - A structure used to set the initial value of the widget, containing the following tags:

min, max, n and log - the parameters used to specify a vector (see vector.pro)

format - a valid format command string used to format the min, max, and increment values. a null string will result in default floating-point formatting.

nformat - a valid format command string to format the pts field. a null string will result in default integer formatting.

units - a string used to label the vector units. for example, if the CW_VECTOR widget is being used to get input to create a vector of lengths in feet, then set value.units='feet'

uunits - a flag to indicate whether or not to actually update the units label.

The same value structure is used with WIDGET_CONTROL to set the value of a CW_VECTOR, as in
WIDGET_CONTROL,WIDGET,SET_VALUE=VALUE

When using the GET_VALUE keyword with WIDGET_CONTROL, however, the returned value is a structure with only four tags: {min,max,pts,log}

MINRANGE, MAXRANGE - These keywords define the range of acceptable values for the min and max fields. If not set, any values for min and max are allowed; otherwise, (min > MINRANGE) < MAXRANGE, and (max > MINRANGE) < MAXRANGE. None, one or both of these keywords can be specified.

MINN - The minimum allowable value for n. default is 1.

NO_RETURN - The default behavior is that the user must press <return> after entering new values. Set this keyword so that an event is returned even if the user just changes a value and then moves the cursor outside of the text entry area.

SPACE - Keyword to all widget_base's used to create this compound widget.

XPAD, YPAD - keyword to widget_base

OUTPUTS:

The id of the created widget is returned.

PROCEDURE:

Entering a value in the pts, min or max fields will set the increment field. Entering a value in the increment field will set the points field, and possibly the max field if the increment doesn't divide evenly into the range specified by min and max.

EXAMPLE:

Create a CW_VECTOR to get input to create a vector of lengths in [feet]:

```
base=WIDGET_BASE()
length_widget=CW_VECTOR(BASE,VALUE={MIN:0.,MAX:10.,N:11,LOG:0, $
                                UNITS:'feet', $
                                FORMAT:'(F10.2)',
                                NFORMAT:'(I4)',
                                UUNITS:1}
                                TITLE='LENGTHS')
```

Later, get the widget values and create the length vector:

```
WIDGET_CONTROL,length_widget,GET_VALUE=value
lengths=VECTOR(value.min,value.max,value.n,log=value.log)
```

MODIFICATION HISTORY:

David L. Windt, Bell Labs, March 1997
windt@bell-labs.com

DLW, June 1997, Added NO_RETURN keyword.

DLW, November 1997, Removed TRACKING keyword; corrected bug that caused improper updates when NO_RETURN was set and the user toggled between linear and logarithmic step sizes.

DLW, June 2003,
Implemented workaround to deal with widget bug when using the NO_RETURN keyword on some platforms. Added SPACE keyword.

windt@astro.columbia.edu

February 2004: Added XPAD, YPAD, SPACE keywords

(See ./cw_vector.pro)

DGTZ_IMAGE

[\[Previous Routine\]](#) [\[Next Routine\]](#) [\[List of Routines\]](#)

NAME:

DGTZ_IMAGE

PURPOSE:

A widget application to interactively measure distances in an image, either between two points, two horizontal lines, or two vertical lines.

CATEGORY:

Image analysis

CALLING SEQUENCE:

DGTZ_IMAGE, IMAGE

INPUTS:

IMAGE = 2-D array containing image.

OUTPUTS:

The measured distances are listed on the widget, and can also be saved to a text file (using MORE.)

KEYWORD PARAMETERS:

UNITS - String specifying units. Default is 'units'.

COMMON BLOCKS:

dgtz_image, internal to this program.

MODIFICATION HISTORY:

David L. Windt, Bell Laboratories, May 1997
windt@bell-labs.com

Jul 1997: Corrected problem with widget labels

(See ./dgtz_image.pro)

DGTZ_PLOT

[\[Previous Routine\]](#) [\[Next Routine\]](#) [\[List of Routines\]](#)

NAME:

DGTZ_PLOT

PURPOSE:

A widget application used to extract (X,Y) values from an image of plot. For example, you can use this program to extract data from a published plot that you've scanned and converted to an image array.

CALLING SEQUENCE:

DGTZ_PLOT, IMAGE, X RANGE, Y RANGE

INPUTS:

IMAGE - 2D array containing the plot image.

X RANGE - 2-element array specifying data range of X axis on plot image.

Y RANGE - 2-element array specifying data range of Y axis on plot image.

KEYWORD PARAMETERS:

XTYPE - set if plot image has log x axis.

YTYPE - set if plot image has log x axis.

SXMAX - Visible size of draw widget along x direction, in pixels.
Default=512.

SYMAX - Visible size of draw widget along y direction, in pixels.
Default=512.

OUTPUTS:

The digitized X,Y pairs are listed on a widget. You can also save these data to a file (using MORE.)

COMMON BLOCKS:

DGTZ_PLOT internal to this procedure.

PROCEDURE:

The image of the plot is displayed on a widget, and the user can digitize points which are converted to X,Y values. The first step, however, is generally to calibrate the X and Y axes; the endpoints of the specified axis are digitized, after pressing the Calibrate X Axis or Calibrate Y Axis button.

MODIFICATION HISTORY:

David L. Windt, Bell Laboratories, May, 1997

September, 1998 - Addex SXMAX and SYMAX keywords.

windt@bell-labs.com

(See `./dgtz_plot.pro`)

DIALOG

[\[Previous Routine\]](#) [\[Next Routine\]](#) [\[List of Routines\]](#)

NAME:

DIALOG

PURPOSE:

A popup widget dialog box to get user input. Like WIDGET_MESSAGE, which is better in some cases, but this widget also includes fields and lists.

CATEGORY:

Widgets.

CALLING SEQUENCE:

Result = DIALOG([TEXT])

OPTIONAL INPUTS:

TEXT - The label seen by the user.

KEYWORD PARAMETERS:

There are 6 types of dialogs, each with unique behavior. With each default dialog type are associated buttons; these buttons can be overridden with the BUTTONS keyword, except in the case of the LIST and FIELD dialogs.

One of the following six keywords MUST be set:

ERROR - Display an error message; default BUTTONS =
['Abort','Continue']

WARNING - Display a warning message. default BUTTONS = ['OK']

INFO - Display an informational message;
default BUTTONS = ['Cancel','OK']

QUESTION - Ask a question. default BUTTONS =
['Cancel','No','Yes']

LIST - Get a selection from a list of choices. default
BUTTONS = ['Cancel','OK'] Must specify CHOICES = string

array of list choices.

Set the RETURN_INDEX keyword to cause the returned value to be the zero-based index of the selected list item.

FILTER can be set to allow the user to filter the list of choices. When FILTER is set, the CHOICE_TYPES keyword must also be supplied: CHOICE_TYPES = string array of names, same length as the CHOICE array, that indicates the type for each element of choice.

FILTER example:

```
FILTER=1
```

```
CHOICES=[ 'Red', 'Blue', 'Green', 'One', 'Two', 'Three' ]
```

```
CHOICE_TYPES=[ 'Colors', 'Colors', 'Colors', 'Numbers', 'Numbers', 'Numbers' ].
```

When the user click the "Filter" button that will be displayed, ; only Colors or Numbers will be listed, depending on ; which type is selected from the displayed droplist of choice types.

FIELD - Get user input, using CW_FIELD. default BUTTONS = ['Cancel','OK']. FLOAT, INTEGER, LONG, and STRING keywords apply here, as does the VALUE keyword to set an initial value. Furthermore, the TEXT input variable can be specified as a two-element array, in which case the 2nd element will appear AFTER the FIELD.

XSIZE - X-Size of FIELD

GROUP - Group leader keyword.

TITLE - title of popup widget.

OUTPUTS:

In the case of LIST or FIELD dialogs, this function returns the selected list element or the user input, respectively. Otherwise, this function returns the name of the pressed button.

EXAMPLE:

1. Create a QUESTION DIALOG widget.

```
D = DIALOG(/QUESTION,'Do you want to continue?')
```

2. Get the user to enter a number.

```
D = DIALOG(/FLOAT,VALUE=3.14159,'Enter a new value for pi.')
```

3. Get the user to choose from a list of options.

```
D = DIALOG(/LIST,CHOICES=['Snoop','Doggy','Dog'])
```

MODIFICATION HISTORY:

David L. Windt, Bell Labs, March 1997

May 1997 - Added GROUP keyword, and modified use of MODAL keyword to work with changes in IDL V5.0

Feb 2013 - Added 2-element TEXT parameter and XSIZE keyword for FIELD dialogs

May 2013 - Added FILTER and CHOICE_TYPES keywords; removed common block.

davidwindt@gmail.com

(See `./dialog.pro`)

DISPLAYED_TABLE_CELLS

[\[Previous Routine\]](#) [\[Next Routine\]](#) [\[List of Routines\]](#)

NAME:

DISPLAYED_TABLE_CELLS

PURPOSE:

This function returns a four-element vector

CALLING SEQUENCE:

```
Result = DISPLAYED_TABLE_CELLS(Table)
```

INPUTS:

Table - Widget id of the widget_table.

OUTPUTS:

EXAMPLE:

MODIFICATION HISTORY:

Daryl Atencio, Research Systems, Oct 2003

(See `./displayed_table_cells.pro`)

DISPLAY_FONT

[\[Previous Routine\]](#) [\[Next Routine\]](#) [\[List of Routines\]](#)

NAME:

DISPLAY_FONT

PURPOSE:

Display the font sets listed in the IDL User's Guide.

CALLING SEQUENCE:

DISPLAY_FONT[,FONT_NUMBER,HARDWARE=HARDWARE]

OPTIONAL INPUT PARAMETERS:

FONT_NUMBER - The font index. If not supplied, the user is prompted for input.

KEYWORD PARAMETERS:

HARDWARE - set to use the hardware fonts (i.e. PostScript for !d.name='PS') set; otherwise Hershey sets are used.

MODIFICATION HISTORY:

D. L. Windt, Bell Laboratories, Sept. 1991
windt@bell-labs.com

(See `./display_font.pro`)

DLIB

[\[Previous Routine\]](#) [\[Next Routine\]](#) [\[List of Routines\]](#)

NAME:

DLIB

PURPOSE:

A cheesy alias to DOC_LIBRARY, with a name that's easier to type.

CATEGORY:

Cheesy aliases.

MODIFICATION HISTORY:

D. L. Windt, Bell Labs, April 1990.
windt@bell-labs.com

(See ./dlib.pro)

EDGE_FIND

[\[Previous Routine\]](#) [\[Next Routine\]](#) [\[List of Routines\]](#)

NAME:

EDGE_FIND

PURPOSE:

Return the center of the rising or falling edge of the
supplied data array.

CALLING SEQUENCE:

Result = EDGE_FIND(X,Y[, /RISING][, /FALLING])

INPUTS:

X, Y - 1D data arrays.

OUTPUTS:

Result = The X value corresponding to the center of the rising
or falling edge of the Y data.

KEYWORD PARAMETERS:

RISING - Set this keyword to find the rising edge. This is the
default.

FALLING - Set this keyword to find the falling edge.

EXAMPLE:

Make some noisy data:

```
x=VECTOR(-8.,8.,100)
y=ATAN(x)+.1*RANDOMN(seed,100)
```

Determine the rising edge:

```
x_edge=EDGE_FIND(y)
```

PROCEDURE:

Pretty cheesy: locate the first maximum (rising edge) or minimum (falling edge) of the derivative of Y. There's almost certainly a better way...

MODIFICATION HISTORY:

David L. Windt, December 2003

windt@astro.columbia.edu

(See ./edge_find.pro)

ELECTRON_MFP

[\[Previous Routine\]](#) [\[Next Routine\]](#) [\[List of Routines\]](#)

NAME:

ELECTRON_MFP

PURPOSE:

This function returns the elastic mean-free-path for electrons of energy E, in a material having density N, and atomic number Z.

CALLING SEQUENCE:

Result = ELECTRON_MFP(Z,A,RHO,E)

INPUTS:

Z = Atomic number

A = Atomic weight (g/mole)

RHO = Density (g/cm³)

E = electron energy in keV

OUTPUTS:

This function returns the elastic mean-free-path, in angstroms.

EXAMPLE:

The elastic mean-free-path of tungsten (Z=74, Rho=19.35) at an electron energy of 100 keV = ELECTRON_MFP(74,183.85,19.35,100.)

MODIFICATION HISTORY:

Written by D. L. Windt, Bell Labs, June 1994
windt@bell-labs.com

(See `./electron_mfp.pro`)

EPlot

[\[Previous Routine\]](#) [\[Next Routine\]](#) [\[List of Routines\]](#)

NAME:

EPlot

PURPOSE:

Plot x vs y , with vertical error bars on y .

CALLING SEQUENCE:

```
EPlot, Y, SIGY
EPlot, X, Y, SIGY
EPlot, Y, SIGY_UP, SIGY_DOWN
EPlot, X, Y, SIGY_UP, SIGY_DOWN
```

INPUTS:

X , Y - 1-D arrays

$SIGY$ - Uncertainty in Y , i.e. $Y \pm SIGY$

$SIGY_UP$, $SIGY_DOWN$ - \pm uncertainties in Y , i.e.,
 $Y + SIGY_UP$ $- SIGY_DOWN$

KEYWORD PARAMETERS:

`BARLINESTYLE` = Linestyle for error bars.

plus all valid IDL plot keywords. Only the `COLOR`,
`THICK`, `NOCLIP`, and `T3D` keywords apply to the error
bars.

MODIFICATION HISTORY:

D. L. Windt, Bell Laboratories, November 1989
Replaced specific plot/oplot keywords with `_EXTRA`,
April, 1997

windt@bell-labs.com

(See `./eplot.pro`)

EROM

[\[Previous Routine\]](#) [\[Next Routine\]](#) [\[List of Routines\]](#)

NAME:

EROM

PURPOSE:

Read columns of data from a text file.

This program can be used to read data written by the MORE program.

The file to be read must be such that if the data are space-separated, then all variables are numeric; String variables are allowed only if the data are separated by tabs, colons, etc.

The file may contain any number of comment lines - which MUST begin with a semicolon, and MUST be positioned before all data lines.

CALLING SEQUENCE:

EROM,V0[,V1,V2,...V9]

or

EROM,V=V

KEYWORD PARAMETERS:

V - Set this keyword to a named variable that will be returned as an array of structures holding the data and the variable names specified in the last comment line. See RESTRICTIONS below for more details.

FILE - String specifying the name of a file; if not supplied, the user is queried.

SKIP - The number of lines at the beginning of the file that should be skipped.

TAB - Specify /TAB for tab-separated data. (The default is space-separated data.) It is only necessary to specify this keyword if the file contains any string data columns.

SEPARATOR - A string specifying the character separating the data columns.

COMMENT - Set this keyword to a named variable that will be returned as a string array holding the comment lines included in the file.

GROUP - GROUP_LEADER keyword passed to DIALOG_PICKFILE if FILE is not specified.

CANCEL - Set this keyword to a named variable that will be returned to indicate if the user pressed the CANCEL button when prompted for a file to read, if the FILE keyword is not set.

OUTPUTS:

If the V keyword is not used, then the user must specify the correct number of Vi (V0, V1, etc.) output parameters. ; There must be as many Vi's specified in the call to EROM as there are columns of data. The V's are double-precision arrays, unless either the TAB or SEPARATOR keyword is specified in which case they are all string arrays.

RESTRICTIONS:

If EROM is called with the V keyword, then the columns of data contained in the file are returned as double-precision fields in the returned V structure variable. Use of the V keyword requires that the data file contain at least one comment line, and the last comment line MUST include the names of the data variables separated by the "|" character.

For example, to read a file using the V keyword containing three columns of 10 rows of data, then the last comment line in the file must look like this:

```
; First Variable Name | Second One | Another Variable Name
```

Thus the V structure returned by EROM will have the following tag names:

```
HELP,/STR,V
```

```
V[0].VALUE      DOUBLE  Array[10]
V[0].NAME       STRING  'First Variable Name'
V[1].VALUE      DOUBLE  Array[10]
V[1].NAME       STRING  'Second One'
V[2].VALUE      DOUBLE  Array[10]
V[2].NAME       STRING  'Another Variable Name'
```

MODIFICATION HISTORY:

David L. Windt, Bell Labs, March 1990

January, 1997 - DLW

Modified to ignore lines beginning with semicolons, and to accept data separated by tabs, etc.; Removed the notitle and comment keyword; included pickfile to prompt for filenames when not specified.

June, 1997 - DLW

Returned numeric variables are now double-precision instead of floating-point.

windt@bell-labs.com

DLW, May 2003

Added V, COMMENTS, GROUP and CANCEL keywords.
Replaced call to PICKFILE with call to DIALOG_PICKFILE

davidwindt@gmail.com

(See **./erom.pro**)

ERRORF_FIT

[\[Previous Routine\]](#) [\[Next Routine\]](#) [\[List of Routines\]](#)

NAME:

ERRORF_FIT

PURPOSE:

fit $y=f(x)$ where:
 $f(x) = a_0 * \text{errorf}((x-a_1)/a_2) + a_3 + x * a_4$

CALLING SEQUENCE:

YFIT = ERRORF_FIT(X,Y,A)

INPUTS:

X - independent variable, must be a vector.

Y - dependent variable, must have the same number of points ;
as x.

A - initial values of adjustable parameters.

OUTPUTS:

YFIT = fitted function.

MODIFICATION HISTORY:

Adapted from GAUSSFIT

D. L. Windt, Bell Laboratories, June 1990
windt@bell-labs.com

(See **./errorf_fit.pro**)

EXPO_FIT

[\[Previous Routine\]](#) [\[Next Routine\]](#) [\[List of Routines\]](#)

NAME:

EXPO_FIT

PURPOSE:

Fit $y=f(x)$ where:
 $F(x) = a_0 \cdot \exp(-\text{abs}(x-a_1)/a_2) + a_3$
 a_0 = height of exp, a_1 = center of peak, a_2 = 1/e width,
Estimate the parameters a_0, a_1, a_2, a_3 and then call curvefit.

CALLING SEQUENCE:

YFIT = EXPO_FIT(X,Y,A)

INPUTS:

X - independent variable, must be a vector.

Y - dependent variable, must have the same number of points ;
as x.

OUTPUTS:

YFIT - fitted function.

OPTIONAL OUTPUT PARAMETERS:

A - Fit coefficients. a four element vector as described
above.

MODIFICATION HISTORY:

Adapted from GAUSSFIT

D. L. Windt, Bell Laboratories, March, 1990
windt@bell-labs.com

27-Feb-2003: Initial value for a may now be specified.

(See ./expo_fit.pro)

FILE_DATE

[\[Previous Routine\]](#) [\[Next Routine\]](#) [\[List of Routines\]](#)

NAME:

FILE_DATE

PURPOSE:

Determine Unix file creation date.

CALLING SEQUENCE:

Result=FILE_DATE(FILE_NAME)

INPUTS:

FILE_NAME - A string specifying the name of the file

OUTPUTS:

Result - a string specifying the file creation date.

RESTRICTIONS:

Probably won't work the way you want. So sue me.

MODIFICATION HISTORY:

David L. Windt, Bell Labs, May 1997
windt@bell-labs.com

(See ./file_date.pro)

FINDEX

[\[Previous Routine\]](#) [\[Next Routine\]](#) [\[List of Routines\]](#)

NAME:

FINDEX

PURPOSE: Compute "floating point index" into a table using binary search. The resulting output may be used with INTERPOLATE.

USEAGE: result = findex(u,v)

INPUT:

u a monitically increasing or decreasing 1-D grid
v a scalar, or array of values

OUTPUT:

result Floating point index. Integer part of RESULT(i) gives the index into to U such that V(i) is between

U(RESULT(i)) and U(RESULT(i)+1). The fractional part is the weighting factor

$$\frac{V(i)-U(RESULT(i))}{U(RESULT(i)+1)-U(RESULT(i))}$$

DISCUSSION:

This routine is used to expedite one dimensional interpolation on irregular 1-d grids. Using this routine with INTERPOLATE is much faster than IDL's INTERPOL procedure because it uses a binary instead of linear search algorithm. The speedup is even more dramatic when the same independent variable (V) and grid (U) are used for several dependent variable interpolations.

EXAMPLE:

; In this example I found the FINDEX + INTERPOLATE combination
; to be about 60 times faster than INTERPOL.

```
u=randomu(iseed,200000) & u=u(sort(u))
v=randomu(iseed,10)      & v=v(sort(v))
y=randomu(iseed,200000) & y=y(sort(y))

t=systime(1) & y1=interpolate(y,findex(u,v)) & print,systime(1)-t
t=systime(1) & y2=interpol(y,u,v)           & print,systime(1)-t
print,f='(3(a,10f7.4/))','findex:    ',y1,'interpol: ',y2,'diff:    ',y1-y2
```

AUTHOR: Paul Ricchiazzi 21 Feb 97
Institute for Computational Earth System Science
University of California, Santa Barbara
paul@icess.ucsb.edu

REVISIONS:

(See ./findex.pro)

FLOYD_SAMPLING

[\[Previous Routine\]](#) [\[Next Routine\]](#) [\[List of Routines\]](#)

NAME:

FLOYD_SAMPLING

PURPOSE:

Randomly choose a unique set of M integers out of a set of N integers ranging in value from 0 to N-1.

This program uses a sampling algorithm invented by Robert Floyd.

CALLING SEQUENCE:

```
Result=FLOYD_SAMPLING(SEED,M,N)
```

INPUTS:

SEED = A variable or constant used by the call to RANDOMU to initialize the random sequence on input, and in which the state of the random number generator is saved on output. Keep in mind that the number sequences this function returns will not be random if called repeatedly SEED with undefined.

M = number of integers to select randomly. M must be greater than or equal to 1.

N = number of integers from which to select (i.e., ranging from 0 to N-1). N must be greater than or equal to 2, and must be greater than M.

OUTPUTS:

Result = M-element array of randomly selected integers.

EXAMPLE:

```
Result=FLOYD_SAMPLING(5,100)
```

Result is a 5-element integer array containing possible values from 0 to 99, with no duplicates. It might look like this:

```
Result=[81,3,24,71,60]
```

MODIFICATION HISTORY:

David L. Windt, RXO, April 2013
davidwindt@gmail.com

(See ./floyd_sampling.pro)

FRACTAL_FIT

[\[Previous Routine\]](#) [\[Next Routine\]](#) [\[List of Routines\]](#)

NAME:

FRACTAL_FIT

PURPOSE:

Fit $y=f(x)$ where:
 $F(x) = a_0/(x^{a_1}) [+a_2]$

Estimate the parameters a_0, a_1, a_2 and then call curvefit.

CALLING SEQUENCE:

YFIT = FRACTAL_FIT(X,Y,A,BACKGROUND=BACKGROUND)

INPUTS:

X = independent variable, must be a vector and MUST BE POSITIVE!

Y = dependent variable, must have the same number of points as x.

BACKGROUND = set to add a background term (a_2).

OUTPUTS:

YFIT = fitted function.

OPTIONAL OUTPUT PARAMETERS:

A = coefficients. a two [three] element vector as described above.

RESTRICTIONS:

X must be positive.

MODIFICATION HISTORY:

D. L. Windt, Bell Laboratories, March, 1990
windt@bell-labs.com

(See ./fractal_fit.pro)

FWHM

[\[Previous Routine\]](#) [\[Next Routine\]](#) [\[List of Routines\]](#)

NAME:

FWHM

PURPOSE:

Interactively measure the full-width-half-max of a region of a curve that has been previously plotted.

CALLING SEQUENCE:

RESULT=FWHM(XAXIS,YAXIS)

INPUTS:

XAXIS - The x axis variable which has been plotted.

YAXIS - The y axis variable which has been plotted.

OPTIONAL INPUT PARAMETERS:

RANGE - Vector of subscripts, which refers to the range of X,Y values over which the FWHM is to be determined. If not supplied, then GET_ROI is used to interactively define the range. To use FWHM with a non-interactive graphics device, range MUST be supplied.

KEYWORD PARAMETERS:

CWHM - The center point of the peak, defined as the mid-point of the FWHM region.

YZERO - The zero point level. If not specified, the zero point level is determined from the endpoints of the region of interest of the curve.

YHM - The value at which the full-width is computed. Allowable range is 0. to 1. If not specified, .5 is used.

INVERT - Set to get width of 'absorption line' rather than 'emission line'.

NOHIGHLIGHT - Set to inhibit highlighting the region of interest.

H_COLOR - The color index for highlighting the region of interest. Default is 7 (Yellow.)

H_THICK - The thickness for highlighting the region of interest.

NOLABEL - Set to inhibit labelling the fwhm.

L_HEADER - String specifying the label header. Default=' '.

L_COLOR - Color index for the label.

L_FORMAT - Format string for label (eg. '(F4.2)').

UNITS - String specifying units along x axis, used in label.

CHARSIZE - Size of label text.

PSYM - PSYM

L_CWHM - Set to include CWHM value in label.

OUTPUTS:

Result - The full-width-half-max of the region of interest of the curve, in x-axis data units.

OPTIONAL OUTPUT PARAMETERS:

ROI - The subscripts of the digitized region of interest.

FWHM_ROI - The subscripts of the region between the fwhm points and the max (min) of the function.

LINE_PTS - A 4-element array containing the coordinates of the line drawn on the plot: [x0,x1,y0,y1]

LABEL - The label for the plot.

L_POS - A two element array containing the x,y coordinates of the label, in data coords.

RESTRICTIONS:

The data must be plotted prior to calling FWHM.

PROCEDURE:

The user is asked to digitize the endpoints of the region of interest with the mouse. The region is highlighted, and the fwhm is labelled.

MODIFICATION HISTORY:

D. L. Windt, Bell Laboratories, November 1989

March 1998 - Removed MANUAL keyword.

January 2004 - Now using local slopes to interpolate fwhm points, for greater precision.

- Added CWHM keyword.

windt@astro.columbia.edu

(See ./fwhm.pro)

GAUSSEXPO_FIT

[\[Previous Routine\]](#) [\[Next Routine\]](#) [\[List of Routines\]](#)

NAME:

GAUSSEXPO_FIT

PURPOSE:

Fit $y=f(x)$ where:

$f(x) = a_0 \cdot \exp(-z^2/2) + a_3 \cdot \exp(-\text{abs}(x-a_4)/a_5) + a_6$ and $z=(x-a_1)/a_2$

a0 = height of gaussian, a1 = center of gaussian, a2 = 1/e width of ; gaussian, a3 = height of exponential, a4 = center of exponential, ; a5 = 1/e width of exponential, a6=background.

Estimate the parameters a0,a1,a2,a3,a4,a5,a6 and then call curvefit.

CALLING SEQUENCE:

YFIT = GAUSSEXPO_FIT(X,Y,A)

INPUTS:

X = independent variable, must be a vector.

Y = dependent variable, must have the same number of points as x.

OUTPUTS:

YFIT = fitted function.

OPTIONAL OUTPUT PARAMETERS:

A = Fit coefficients. A six element vector as described above.

MODIFICATION HISTORY:

Adapted from GAUSSFIT

D. L. Windt, Bell Laboratories, March, 1990
windt@bell-labs.com

(See ./gaussexpo_fit.pro)

GAUSS_FIT

[\[Previous Routine\]](#) [\[Next Routine\]](#) [\[List of Routines\]](#)

NAME:

GAUSS_FIT

PURPOSE:

Fit $y=f(x)$ where:

$f(x) = a0 \cdot \exp(-z^2/2) + a3$

and $z=(x-a1)/a2$

a0 = height of gaussian, a1 = center of gaussian, a2 = 1/e width,
a3 = background.

Estimate the parameters a0,a1,a2,a3 and then call CURFIT.

CALLING SEQUENCE:

YFIT = GAUSS_FIT(X,Y,A)

INPUTS:

X - independent variable, must be a vector.

Y - dependent variable, must have the same number of points ;
as x.

OUTPUTS

YFIT - fitted function.

OPTIONAL OUTPUT PARAMETERS:

A - Fit coefficients. a three element vector as described
above.

MODIFICATION HISTORY:

Adapted from GAUSSFIT

D. L. Windt, Bell Laboratories, March, 1990
windt@bell-labs.com

(See ./gauss_fit.pro)

GET_PEAK

[\[Previous Routine\]](#) [\[Next Routine\]](#) [\[List of Routines\]](#)

NAME:

GET_PEAK

PURPOSE:

Interactively find the local maximum of a previously plotted
curve, and indicate it on the plot.

CALLING SEQUENCE:

Result=GET_PEAK(XAXIS,YAXIS)

INPUTS:

XAXIS = the x axis variable which has been plotted.

YAXIS = the y axis variable which has been plotted.

KEYWORD PARAMETERS:

COLOR - the color index for marking the local maximum.

NOMARK - set to disable marking the location of the peak.

NOHIGHLIGHT - set to disable highlighting the region of interest.

H_COLOR - the color index for highlighting the region of interest. Default is 7 (Yellow).

H_THICK- the thickness for highlighting the region ; of interest.

PRINT - set to print the x,y values of the peak.

OUTPUTS:

Result = the array subscript of the local max.

SIDE EFFECTS:

TEK_COLOR is used to load in the tektronix colors.
The region of interest of the curve is highlighted.
A vertical line is drawn through the local maximum.

PROCEDURE:

The user is asked to digitize the endpoints of the region of interest with the mouse using GET_ROI.

MODIFICATION HISTORY:

D. L. Windt, Bell Laboratories, February 1990.

windt@bell-labs.com

(See ./get_peak.pro)

GET_PT

[\[Previous Routine\]](#) [\[Next Routine\]](#) [\[List of Routines\]](#)

NAME:

GET_PT

PURPOSE:

Digitize a point on a previously plotted curve, and return the corresponding array element.

CALLING SEQUENCE:

Result = GET_PT(XAXIS,YAXIS,XPOINT,YPOINT)

INPUTS:

XAXIS - the x axis vector which was used to make the plot.

YAXIS - the y axis vector which was used to make the plot.

KEYWORD PARAMETERS:

NOHIGHLIGHT - set to inhibit putting a red mark on the curve
at the digitized point.

MESSAGE - a string to print as the message to the user.
Default = 'Digitize a point: '

NOINIT - set to inhibit placing the cursor in the center of
the plot window.

OUTPUTS:

Result - The array subscript of the digitized point.

OPTIONAL OUTPUT PARAMETERS:

XPOINT, YPOINT - the digitized points.

SIDE EFFECTS:

A mark is drawn on the plot at the digitized point.

PROCEDURE:

The user is asked to digitize a point on the curve using the
mouse. The VALUE_TO_INDEX function is used to find the
closest array element.

MODIFICATION HISTORY:

D. L. Windt, Bell Laboratories, November 1989
Feb. 1991, Removed call to TEK_COLOR
Mar. 1997, replaced index search code with call to
VALUE_TO_INDEX function.

windt@bell-labs.com

(See ./get_pt.pro)

GET_ROI

[\[Previous Routine\]](#) [\[Next Routine\]](#) [\[List of Routines\]](#)

NAME:

GET_ROI

PURPOSE:

Get a region-of-interest of a previously plotted curve.

CALLING SEQUENCE:

Result=GET_ROI(XAXIS,YAXIS)

INPUTS:

XAXIS = the x axis variable which has been plotted.

YAXIS = the y axis variable which has been plotted.

KEYWORD PARAMETERS:

NOHIGHLIGHT - set to disable highlighting the region of interest.

H_COLOR - the color index for highlighting the region of interest. Default is 7 (Yellow).

H_THICK - the thickness for highlighting the region ; of interest.

PSYM - PSYM.

OUTPUTS:

Result = the array of subscripts of the roi.

SIDE EFFECTS:

TEK_COLOR is used to load in the tektronix colors.
The region of interest of the curve is highlighted.

PROCEDURE:

The user is asked to digitize the endpoints of the region of interest with the mouse using GET_PT. The region is highlighted (unless nohighlight is set.)

MODIFICATION HISTORY:

D. L. Windt, Bell Laboratories, November 1989

windt@bell-labs.com

(See ./get_roi.pro)

GHOSTVIEW

[\[Previous Routine\]](#) [\[Next Routine\]](#) [\[List of Routines\]](#)

NAME:

GHOSTVIEW

PURPOSE:

Use the Unix ghostview program to view an IDL postscript file

CALLING SEQUENCE:

GHOSTVIEW [,FILE=FILE]

KEYWORD PARAMETERS:

FILE - the name of the file to view. Default is idl.ps

RESTRICTIONS:

Since the procedure spawns a "ghostview" process,
such an executable must exist or it ain't goin' nowhere.

PROCEDURE:

If the current device is PS, the program will issue
a DEVICE,/CLOSE command.

It will then SPAWN,'ghostview file_name&'

MODIFICATION HISTORY:

David L. Windt, Bell Labs, March 1997
windt@bell-labs.com

(See ./ghostview.pro)

GREEK

[\[Previous Routine\]](#) [\[Next Routine\]](#) [\[List of Routines\]](#)

NAME:

GREEK

PURPOSE:

This function returns the string needed to draw the specified
greek character using either the vector graphics font no. 4,
or PostScript font 9.

If (!d.name eq 'PS') and (!p.font eq 0), then the PostScript font will be used. Otherwise, the vector font will be used.

CALLING SEQUENCE:

Result = GREEK(Name)

INPUTS:

Name - String specifying the greek character name. Valid inputs are:

alpha, beta, gamma, delta, epsilon, zeta, eta, theta
iota, kappa, lambda, mu, nu, xi, omicron, pi, rho,
sigma, tau, upsilon, phi, chi, psi, omega

Alpha, Beta, Gamma, Delta, Epsilon, Zeta, Eta, Theta
Iota, Kappa, Lambda, Mu, Nu, Xi, Omicron, Pi, Rho,
Sigma, Tau, Upsilon, Phi, Chi, Psi, Omega

Although not greek, the following characters are also valid (but will only work with the 'default' font !3):

angstrom, Angstrom, degrees, plus_minus

KEYWORDS:

FORCE_PS - Set to use PostScript font, regardless of the value of !d.name and !p.font.

PLAIN - Set to just return Name in plain text.

APPEND_FONT - Set to append the characters specifying a 'default' font: !3. That is, if this keyword is set, then the command

Result=GREEK(theta,/APPEND_FONT)

will return the string

'!9q!3' for PostScript and '!4h!3' for vector fonts.

OUTPUTS:

Result - The string containing the specified greek character.

EXAMPLE:

Result=GREEK(theta)

In this case, Result='!9q' if !d.name is 'PS' and !p.font is 0; otherwise, Result='!4h'

MODIFICATION HISTORY:

David L. Windt, Bell Labs, September 1998.
windt@bell-labs.com

(See ./greek.pro)

KAISER_BESSEL

[\[Previous Routine\]](#) [\[Next Routine\]](#) [\[List of Routines\]](#)

NAME:

KAISER_BESSEL

PURPOSE:

Window function for Fourier Transform filtering.

CATEGORY:

Signal, image processing.

CALLING SEQUENCE:

Result = KAISER_BESSEL(N1) (for 1D)

Result = KAISER_BESSEL(N1,N2) (for 2D)

INPUTS:

N1 - The number of columns of the result.

N2 - The number of rows of the result.

KEYWORD PARAMETERS:

ALPHA - The value of $\pi \cdot \alpha$ is half of the time-bandwidth
product. Default = 3.0

OUTPUTS:

Result(i) = BESELI(!pi*alpha*sqrt(1-((findgen(N)-N/2) / (N/2))^2),0) / \$
BESELI(!pi*alpha,0)

MODIFICATION HISTORY:

David L Windt, Bell Labs, August 1996
May, 1997 - Added 2D option.
windt@bell-labs.com

(See ./kaiser_bessel.pro)

LEGEND_RXO

[\[Previous Routine\]](#) [\[Next Routine\]](#) [\[List of Routines\]](#)

NAME:

LEGEND_RXO

PURPOSE:

Add to a plot a legend containing lines and plotting symbols, optionally enclosed in a box.

CALLING SEQUENCE:

LEGEND_RXO, LABELS

INPUTS:

LABELS - n-element string array of labels.

KEYWORD PARAMETERS:

POSITION - an integer, specifying the location of the legend box:

0: no legend is drawn.
1: below plot, left
2: below plot, center
3: below plot, right
4: lower left
5: lower center
6: lower right
7: middle left
8: middle center
9: middle right
10: upper left
11: upper center
12: upper right

if not specified, default position=10

COLOR - n-element array of colors. default is !p.color

LINESTYLE - n-element array of linestyles. if omitted, only symbols are plotted.

THICK - n-element array of thicknesses. default is !p.thick

PSYM - n-element array of psym values. if positive, only symbols are plotted.

SYMSIZE - n-element array of symsize values. default is !p.symsize

SYMBOLS - array of 'symbol' specifiers: each element of psym which is equal to 8 (user-defined symbol) must have a corresponding value for 'symbol' to be

used by the procedure SYMBOLS.

Examples: psym=[8,8,8,8],symbols=[1,2,20,30]
 psym=[1,2,8,8],symbols=[1,2]

USE_SYM - Set this keyword to use the SYM function to generate plotting symbols. In this case the SYMBOLS keyword is not needed; just specify PSYM values to be passed to the SYM function. i.e., PSYM=14,/USE_SYM will produce an filled rightfacing triangle

CHARSIZE - scalar specifying the size of the text.

TITLE - scalar string specifying legend title

T_COLOR - scalar specifying the color index of the title.

NOLINES - set to inhibit drawing lines and symbols; just draw labels in color.

SYM_ONLY - set to inhibit drawing lines; just draw symbols.

NOBOX - set to inhibit drawing a box around the legend

LINEWIDTH - width in character units. default = 4.

BOXPADX - padding in character units, between text and box in x. default=2.0

BOXPADY - padding in character units, between text and box in y. default=0.5

FONT - Set to an integer from 3 to 20, corresponding to the Hershey vector font sets, referring to the font used to display the text. If a font other than !3 is used in the text string, then FONT should be set accordingly. (Any font commands embedded in the text string are ignored.)

BOXFUDGE - A scaling factor, used to fudge the width of the box surrounding the text. Default=1.0.

BOXCOLOR - set to the color index used to draw the box. Default is !P.COLOR.

BOXFILL - set to the color index used to fill the box. Omit, or set to -1 for no fill. No effect if NOBOX=1.

Plus all valid graphics keywords for xyouts and plots

RESTRICTIONS:

When specifying a position of 1,2 or 3, you'll need to (a) use the same charsize value for the plot and for the legend, and (b) draw the plot with an extra ymargin(0). i.e., set ymargin(0)=7+n_elements(text_array)

MODIFICATION HISTORY:

David L. Windt, Bell Labs, March 1997
windt@bell-labs.com

May 2011, dlw:

Now using WIDTH keyword from XYOUTS to do an even better job of drawing the box.

October, 1997, dlw:

Now using the TEXT_WIDTH function, in order to do a somewhat better job of drawing the box around the text.

NONPRINTER_SCALE keyword parameter is now obsolete.

BOXFUDGEX keyword parameter added.

January 2004 - Added USE_SYM keyword

May 2013 - Added BOXCOLOR and BOXFILL, and renamed LEGEND_RXO, DLW, davidwindt@gmail.com

(See ./legend_rxo.pro)

LPRINT

[\[Previous Routine\]](#) [\[Next Routine\]](#) [\[List of Routines\]](#)

NAME:

LPRINT

PURPOSE:

Close an IDL graphics file and print it.

CALLING SEQUENCE:

LPRINT

KEYWORD PARAMETERS:

NORETURN - set this keyword to inhibit executing
set_plot,getenv('IDL_DEVICE') followed by
!p.font=-1

FILE - the name of the file to print. Default is device
dependent: idl.ps for PS, idl.hp for HP, and idl.pcl
for PCL devices.

PRINTER - set to the name of the printer to use. Default = lp

COMMAND - set to the name of the printer command to use. Default = lpr.

Note: the COMMAND, PRINTER, and FILE keywords are combined as follows:

if COMMAND='lpr', then the program spawns the unix command
"lpr -Pprinter file"

if COMMAND='lp', then the program spawns the unix command
"lp dprinter file"

if COMMAND is anything else, the program simply ignores the printer and file keywords, and spawns the command as is.

MODIFICATION HISTORY:

D. L. Windt, Bell Laboratories, November 1989
Added PRINTER keyword, June 1993.

Added COMMAND keywr, replaced RETURN with NORETURN keyword,
and added code to execute !p.font=-1 unless
NORETURN keyword is set. March, 1997.

windt@bell-labs.com

(See ./print.pro)

LS

[\[Previous Routine\]](#) [\[Next Routine\]](#) [\[List of Routines\]](#)

NAME:

LS

CATEGORY:

Stupid little convenience routines.

PURPOSE:

List the contents of the current directory, like the Unix 'ls' command.

CALLING SEQUENCE:

LS[,NAME]

NAME - An optional string specifying the names of the files to be listed. Wild cards are allowed. For example, `ls, '*.pro'` will list all files ending in `.pro`.

MODIFICATION HISTORY:

David L. Windt, Bell Labs, November 1989
windt@bell-labs.com

February, 1998 - Now works under Windows and MacOS, making use of `FINDFILE`. But the old `DIR` keyword is gone.

(See `./ls.pro`)

MAKE_LATEX_TBL

[\[Previous Routine\]](#) [\[Next Routine\]](#) [\[List of Routines\]](#)

NAME:

MAKE_LATEX_TBL

PURPOSE:

Create a LaTeX format table.

CALLING SEQUENCE:

MAKE_LATEX_TBL, ARRAY, TFILE

INPUTS:

ARRAY - (n,m) array of data.

TFILE - string specifying the name of the `'.tex'` file to create.

KEYWORD PARAMETERS:

COLUMNS - An n-element string specifying the LaTeX column format. For example, if `array = (3,m)`, then an acceptable value for columns would be `['|l|', '|c|', '|c|']`, which would left-justify the first column of data, and center the remaining two.

FORMAT - an n-element string specifying the FORMAT used to `PRINTF` the data. This must conform to IDL FORMAT standards. If not set, the default the data are printed using the IDL free format.

TITLE - a string specifying the title of the table.

HEADINGS - an n-element string array containing the table

headings.

NOHLINES - set to inhibit printing \hline between rows of data.

SIDE EFFECTS:

The '.tex' file is created.

RESTRICTIONS:

The TITLE is printed with vertical lines on either ; side, regardless of how the COLUMNS parameter may be ; set. It is thus necessary to edit the file to remove the vertical line commands if desired.

PROCEDURE:

The data contained in ARRAY are printed to a file ; with the appropriate '&' and '\\\'' symbols necessary ; for use as in the LaTeX tabular environment. If ; COLUMNS is not set, the default is '|c|' which centers the data in columns, with vertical line separators.

MODIFICATION HISTORY:

David L. Windt, Bell Laboratories, December 1989.
windt@bell-labs.com

(See ./make_latex_tbl.pro)

MK_BITARRAY

[\[Previous Routine\]](#) [\[Next Routine\]](#) [\[List of Routines\]](#)

NAME:

mk_bitarray

PURPOSE:

Create an array of 1's & 0's corresponding to input bits set
(works for negative numbers, too, unlike similar routines)

CATEGORY:

Bits

CALLING SEQUENCE:

IDL> array= mk_bitarray(input)
IDL> array= mk_bitarray(input, nbits=5)

INPUTS:

input = whatever; might be something like !X.STYLE

KEYWORD PARAMETERS:

NBITS=nbits - length of returned array (default to input type)
PRINT - if set, will print bits in groups of fours.

OUTPUTS:

Byte array containing 1's and 0's out

COMMON BLOCKS:

NONE

EXAMPLE:

```
IDL> print,mk_bitarray(3, nbits=8)
   1   1   0   0   0   0   0   0
IDL> dum = mk_bitarray( 1025, /print )'
  1 0 0 0   0 0 0 0   0 0 1 0   0 0 0 0
```

LIMITATION:

Only works on a scalar.

MODIFICATION HISTORY:

05-Jun-00 default nbits to input type. add print keyword.
30-Mar-99 Written by Bill Davis, PPPL

(See ./mk_bitarray.pro)

MK_NEW_PTRS

[\[Previous Routine\]](#) [\[Next Routine\]](#) [\[List of Routines\]](#)

NAME:

MK_NEW_PTRS

PURPOSE:

Make a copy of a pointer variable, or a structure variable containing pointer variables as tags, such that the de-referenced pointer values are copied, but new pointer heap variables are created in the copy of the original variable.

CALLING SEQUENCE:

Results=MK_NEW_PTRS(SOURCE)

INPUTS:

SOURCE = The source structure variable.

EXAMPLE:

```
IDL> a={n:ptr_new(/allocate_heap)}
IDL> b=a
IDL> help,a.n,b.n
<Expression>    POINTER    = <PtrHeapVar1>
<Expression>    POINTER    = <PtrHeapVar1>
IDL> b=mk_new_ptrs(a)
IDL> help,a.n,b.n
<Expression>    POINTER    = <PtrHeapVar1>
<Expression>    POINTER    = <PtrHeapVar2>
```

PROCEDURE:

SAVE and RESTORE to and from a temporary file are used to generate the new pointers. Ideally, there would be keyword to the STRUCT_ASSIGN procedure that would accomplish the same

thing without having to use this workaround...but so far (IDL 6.0) such a keyword does not exist.

MODIFICATION HISTORY:

David L. Windt, Columbia University, Oct-2003

windt@astro.columbia.edu

(See `./mk_new_ptrs.pro`)

MORE

[\[Previous Routine\]](#) [\[Next Routine\]](#) [\[List of Routines\]](#)

NAME:

MORE

PURPOSE:

Print one or more variables on the screen or to a file, using the MORE keyword to printf.

CALLING SEQUENCE:

MORE,v0[,v1,v2,...v19]

INPUTS:

V0,V1,...V19 - Any type of array variables; they must all be the same length.

KEYWORD PARAMETERS:

FILE - string specifying the name of an output file.

INDEX - set to print the array indices in the first column.

TITLE - string array of variable names.

COMMENT - string array of comments

TAB - set this keyword to create tab-separated data; this is useful when writing to a file if any of the variables are strings, in which case the data can be read using EROM,/TAB

COMMA - set this keyword to create comma-separated data

APPEND - If FILE is specified, the APPEND keyword caused data to be appended to the end of the file.

SEPARATE_TITLES - Set this keyword to insert a "|" character

between each variable name specified by the
TITLE keyword.

MODIFICATION HISTORY:

David L. Windt, Bell Labs, March 1990

Added comment keyword, August 1992

March 1997 - Title and comment lines are now written with
preceding semicolons. Fixed bug to correctly
deal with string arrays. Added TAB keyword.
Removed NOINDEX keyword. Added INDEX keyword.

November 1997 - Removed Unix-specific stuff, so that it now
works (somewhat) under Windows and MacOS.

May 2003 - Added APPEND and SEPARATE_TITLES keyword.

July 2007 - Changed loop counters to longword type.

November 2008 - Added COMMA keyword.

davidwindt@gmail.com

(See ./more.pro)

MPFIT

[\[Previous Routine\]](#) [\[Next Routine\]](#) [\[List of Routines\]](#)

NAME:

MPFIT

AUTHOR:

Craig B. Markwardt, NASA/GSFC Code 662, Greenbelt, MD 20770
craigm@lheamail.gsfc.nasa.gov

UPDATED VERSIONS can be found on my WEB PAGE:

<http://cow.physics.wisc.edu/~craigm/idl/idl.html>

PURPOSE:

Perform Levenberg-Marquardt least-squares minimization (MINPACK-1)

MAJOR TOPICS:

Curve and Surface Fitting

CALLING SEQUENCE:

```
parms = MPFIT(MYFUNCT, start_parms, FUNCTARGS=fcnargs, NFEV=nfev,  
              MAXITER=maxiter, ERRMSG=errmsg, NPRINT=nprint, QUIET=quiet,  
              FTOL=ftol, XTOL=xtol, GTOL=gtol, NITER=niter,  
              STATUS=status, ITERPROC=iterproc, ITERARGS=iterargs,  
              COVAR=covar, PERROR=perror, BESTNORM=bestnorm,  
              PARINFO=parinfo)
```

DESCRIPTION:

MPFIT uses the Levenberg-Marquardt technique to solve the least-squares problem. In its typical use, MPFIT will be used to fit a user-supplied function (the "model") to user-supplied data points (the "data") by adjusting a set of parameters. MPFIT is based upon MINPACK-1 (LMDIF.F) by More' and collaborators.

For example, a researcher may think that a set of observed data points is best modelled with a Gaussian curve. A Gaussian curve is parameterized by its mean, standard deviation and normalization. MPFIT will, within certain constraints, find the set of parameters which best fits the data. The fit is "best" in the least-squares sense; that is, the sum of the weighted squared differences between the model and data is minimized.

The Levenberg-Marquardt technique is a particular strategy for iteratively searching for the best fit. This particular implementation is drawn from MINPACK-1 (see NETLIB), and seems to be more robust than routines provided with IDL. This version allows upper and lower bounding constraints to be placed on each parameter, or the parameter can be held fixed.

The IDL user-supplied function should return an array of weighted deviations between model and data. In a typical scientific problem the residuals should be weighted so that each deviate has a gaussian sigma of 1.0. If X represents values of the independent variable, Y represents a measurement for each value of X, and ERR represents the error in the measurements, then the deviates could be calculated as follows:

$$\text{DEVIATES} = (Y - F(X)) / \text{ERR}$$

where F is the function representing the model. You are recommended to use the convenience functions MPFITFUN and MPFITEXPR, which are driver functions that calculate the deviates for you. If ERR are the 1-sigma uncertainties in Y, then

$$\text{TOTAL}(\text{DEVIATES}^2)$$

will be the total chi-squared value. MPFIT will minimize the chi-square value. The values of X, Y and ERR are passed through MPFIT to the user-supplied function via the FUNCTARGS keyword.

Simple constraints can be placed on parameter values by using the PARINFO keyword to MPFIT. See below for a description of this keyword.

MPFIT does not perform more general optimization tasks. See TNMIN instead. MPFIT is customized, based on MINPACK-1, to the least-squares minimization problem.

USER FUNCTION

The user must define a function which returns the appropriate values as specified above. The function should return the weighted

deviations between the model and the data. For applications which use finite-difference derivatives -- the default -- the user function should be declared in the following way:

```
FUNCTION MYFUNCT, p, X=x, Y=y, ERR=err
; Parameter values are passed in "p"
model = F(x, p)
return, (y-model)/err
END
```

See below for applications with explicit derivatives.

The keyword parameters X, Y, and ERR in the example above are suggestive but not required. Any parameters can be passed to MYFUNCT by using the FUNCTARGS keyword to MPFIT. Use MPFITFUN and MPFITEXPR if you need ideas on how to do that. The function *must* accept a parameter list, P.

In general there are no restrictions on the number of dimensions in X, Y or ERR. However the deviates *must* be returned in a one-dimensional array, and must have the same type (float or double) as the input arrays.

See below for error reporting mechanisms.

CHECKING STATUS AND HANDLING ERRORS

Upon return, MPFIT will report the status of the fitting operation in the STATUS and ERRMSG keywords. The STATUS keyword will contain a numerical code which indicates the success or failure status. Generally speaking, any value 1 or greater indicates success, while a value of 0 or less indicates a possible failure. The ERRMSG keyword will contain a text string which should describe the error condition more fully.

By default, MPFIT will trap fatal errors and report them to the caller gracefully. However, during the debugging process, it is often useful to halt execution where the error occurred. When you set the NOCATCH keyword, MPFIT will not do any special error trapping, and execution will stop wherever the error occurred.

MPFIT does not explicitly change the !ERROR_STATE variable (although it may be changed implicitly if MPFIT calls MESSAGE). It is the caller's responsibility to call MESSAGE, /RESET to ensure that the error state is initialized before calling MPFIT.

User functions may also indicate non-fatal error conditions using the ERROR_CODE common block variable, as described below under the MPFIT_ERROR common block definition (by setting ERROR_CODE to a number between -15 and -1). When the user function sets an error condition via ERROR_CODE, MPFIT will gracefully exit immediately and report this condition to the caller. The ERROR_CODE is returned in the STATUS keyword in that case.

EXPLICIT DERIVATIVES

In the search for the best-fit solution, MPFIT by default calculates derivatives numerically via a finite difference approximation. The user-supplied function need not calculate the derivatives explicitly. However, the user function *may* calculate the derivatives if desired, but only if the model function is declared with an additional position parameter, DP, as described below. If the user function does not accept this additional parameter, MPFIT will report an error. As a practical matter, it is often sufficient and even faster to allow MPFIT to calculate the derivatives numerically, but this option is available for users who wish more control over the fitting process.

There are two ways to enable explicit derivatives. First, the user can set the keyword AUTODERIVATIVE=0, which is a global switch for all parameters. In this case, MPFIT will request explicit derivatives for every free parameter.

Second, the user may request explicit derivatives for specifically selected parameters using the PARINFO.MPSIDE=3 (see "CONSTRAINING PARAMETER VALUES WITH THE PARINFO KEYWORD" below). In this strategy, the user picks and chooses which parameter derivatives are computed explicitly versus numerically. When PARINFO[i].MPSIDE EQ 3, then the ith parameter derivative is computed explicitly.

The keyword setting AUTODERIVATIVE=0 always globally overrides the individual values of PARINFO.MPSIDE. Setting AUTODERIVATIVE=0 is equivalent to resetting PARINFO.MPSIDE=3 for all parameters.

Even if the user requests explicit derivatives for some or all parameters, MPFIT will not always request explicit derivatives on every user function call.

EXPLICIT DERIVATIVES - CALLING INTERFACE

When AUTODERIVATIVE=0, the user function is responsible for calculating the derivatives of the **residuals** with respect to each parameter. The user function should be declared as follows:

```

;
; MYFUNCT - example user function
;   P - input parameter values (N-element array)
;   DP - upon input, an N-vector indicating which parameters
;         to compute derivatives for;
;         upon output, the user function must return
;         an ARRAY(M,N) of derivatives in this keyword
;   (keywords) - any other keywords specified by FUNCTARGS
; RETURNS - residual values
;
FUNCTION MYFUNCT, p, dp, X=x, Y=y, ERR=err
  model = F(x, p)          ;; Model function
  resid = (y - model)/err ;; Residual calculation (for example)

  if n_params() GT 1 then begin
    ; Create derivative and compute derivative array
    requested = dp ; Save original value of DP
    dp = make_array(n_elements(x), n_elements(p), value=x[0]*0)
  
```

```

      ; Compute derivative if requested by caller
      for i = 0, n_elements(p)-1 do if requested(i) NE 0 then $
        dp(*,i) = FGRAD(x, p, i) / err
      endif

      return, resid
    END

```

where FGRAD(x, p, i) is a model function which computes the derivative of the model F(x,p) with respect to parameter P(i) at X.

A quirk in the implementation leaves a stray negative sign in the definition of DP. The derivative of the **residual** should be *"-FGRAD(x,p,i) / err"* because of how the residual is defined (*"resid = (data - model) / err"*). ****HOWEVER**** because of the implementation quirk, MPFIT expects FGRAD(x,p,i)/err instead, i.e. the opposite sign of the gradient of RESID.

Derivatives should be returned in the DP array. DP should be an ARRAY(m,n) array, where m is the number of data points and n is the number of parameters. -DP[i,j] is the derivative of the ith residual with respect to the jth parameter (note the minus sign due to the quirk described above).

As noted above, MPFIT may not always request derivatives from the user function. In those cases, the parameter DP is not passed. Therefore functions can use N_PARAMS() to indicate whether they must compute the derivatives or not.

The derivatives with respect to fixed parameters are ignored; zero is an appropriate value to insert for those derivatives. Upon input to the user function, DP is set to a vector with the same length as P, with a value of 1 for a parameter which is free, and a value of zero for a parameter which is fixed (and hence no derivative needs to be calculated). This input vector may be overwritten as needed. In the example above, the original DP vector is saved to a variable called REQUESTED, and used as a mask to calculate only those derivatives that are required.

If the data is higher than one dimensional, then the **last** dimension should be the parameter dimension. Example: fitting a 50x50 image, "dp" should be 50x50xNPAR.

EXPLICIT DERIVATIVES - TESTING and DEBUGGING

For reasonably complicated user functions, the calculation of explicit derivatives of the correct sign and magnitude can be difficult to get right. A simple sign error can cause MPFIT to be confused. MPFIT has a derivative debugging mode which will compute the derivatives **both** numerically and explicitly, and compare the results.

It is expected that during production usage, derivative debugging should be disabled for all parameters.

In order to enable derivative debugging mode, set the following

PARINFO members for the ith parameter.

```
PARINFO[i].MPSIDE = 3      ; Enable explicit derivatives
PARINFO[i].MPDERIV_DEBUG = 1 ; Enable derivative debugging mode
PARINFO[i].MPDERIV_RELTOL = ?? ; Relative tolerance for comparison
PARINFO[i].MPDERIV_ABSTOL = ?? ; Absolute tolerance for comparison
```

Note that these settings are maintained on a parameter-by-parameter basis using PARINFO, so the user can choose which parameters derivatives will be tested.

When .MPDERIV_DEBUG is set, then MPFIT first computes the derivative explicitly by requesting them from the user function. Then, it computes the derivatives numerically via finite differencing, and compares the two values. If the difference exceeds a tolerance threshold, then the values are printed out to alert the user. The tolerance level threshold contains both a relative and an absolute component, and is expressed as,

$$\text{ABS}(\text{DERIV_U} - \text{DERIV_N}) \geq (\text{ABSTOL} + \text{RELTOL} * \text{ABS}(\text{DERIV_U}))$$

where DERIV_U and DERIV_N are the derivatives computed explicitly and numerically, respectively. Appropriate values for most users will be:

```
PARINFO[i].MPDERIV_RELTOL = 1d-3 ;; Suggested relative tolerance
PARINFO[i].MPDERIV_ABSTOL = 1d-7 ;; Suggested absolute tolerance
```

although these thresholds may have to be adjusted for a particular problem. When the threshold is exceeded, users can expect to see a tabular report like this one:

```
FJAC DEBUG BEGIN
#      IPNT      FUNC      DERIV_U      DERIV_N      DIFF_ABS      DIFF_REL
FJAC PARM 2
      80      -0.7308      0.04233      0.04233 -5.543E-07 -1.309E-05
      99       1.370      0.01417      0.01417 -5.518E-07 -3.895E-05
     118      0.07187     -0.01400     -0.01400 -5.566E-07  3.977E-05
     137       1.844     -0.04216     -0.04216 -5.589E-07  1.326E-05
FJAC DEBUG END
```

The report will be bracketed by FJAC DEBUG BEGIN/END statements. Each parameter will be delimited by the statement FJAC PARM n, where n is the parameter number. The columns are,

```
IPNT - data point number (0 ... M-1)
FUNC - function value at that point
DERIV_U - explicit derivative value at that point
DERIV_N - numerical derivative estimate at that point
DIFF_ABS - absolute difference = (DERIV_U - DERIV_N)
DIFF_REL - relative difference = (DIFF_ABS)/(DERIV_U)
```

When prints appear in this report, it is most important to check that the derivatives computed in two different ways have the same numerical sign and the same order of magnitude, since these are the most common programming mistakes.

A line of this form may also appear

```
# FJAC_MASK = 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

This line indicates for which parameters explicit derivatives are expected. A list of all-1s indicates all explicit derivatives for all parameters are requested from the user function.

CONSTRAINING PARAMETER VALUES WITH THE PARINFO KEYWORD

The behavior of MPFIT can be modified with respect to each parameter to be fitted. A parameter value can be fixed; simple boundary constraints can be imposed; limitations on the parameter changes can be imposed; properties of the automatic derivative can be modified; and parameters can be tied to one another.

These properties are governed by the PARINFO structure, which is passed as a keyword parameter to MPFIT.

PARINFO should be an array of structures, one for each parameter. Each parameter is associated with one element of the array, in numerical order. The structure can have the following entries (none are required):

- .VALUE - the starting parameter value (but see the START_PARAMS parameter for more information).
- .FIXED - a boolean value, whether the parameter is to be held fixed or not. Fixed parameters are not varied by MPFIT, but are passed on to MYFUNCT for evaluation.
- .LIMITED - a two-element boolean array. If the first/second element is set, then the parameter is bounded on the lower/upper side. A parameter can be bounded on both sides. Both LIMITED and LIMITS must be given together.
- .LIMITS - a two-element float or double array. Gives the parameter limits on the lower and upper sides, respectively. Zero, one or two of these values can be set, depending on the values of LIMITED. Both LIMITED and LIMITS must be given together.
- .PARNAME - a string, giving the name of the parameter. The fitting code of MPFIT does not use this tag in any way. However, the default ITERPROC will print the parameter name if available.
- .STEP - the step size to be used in calculating the numerical derivatives. If set to zero, then the step size is computed automatically. Ignored when AUTODERIVATIVE=0. This value is superceded by the RELSTEP value.
- .RELSTEP - the *relative* step size to be used in calculating the numerical derivatives. This number is the fractional size of the step, compared to the parameter value. This value supercedes the STEP setting. If the parameter is zero, then a default

step size is chosen.

.MPSIDE - selector for type of derivative calculation. This field can take one of five possible values:

- 0 - one-sided derivative computed automatically
- 1 - one-sided derivative $(f(x+h) - f(x))/h$
- 1 - one-sided derivative $(f(x) - f(x-h))/h$
- 2 - two-sided derivative $(f(x+h) - f(x-h))/(2*h)$
- 3 - explicit derivative used for this parameter

In the first four cases, the derivative is approximated numerically by finite difference, with step size $H=STEP$, where the STEP parameter is defined above. The last case, MPSIDE=3, indicates to allow the user function to compute the derivative explicitly (see section on "EXPLICIT DERIVATIVES"). AUTODERIVATIVE=0 overrides this setting for all parameters, and is equivalent to MPSIDE=3 for all parameters. For MPSIDE=0, the "automatic" one-sided derivative method will choose a direction for the finite difference which does not violate any constraints. The other methods (MPSIDE=-1 or MPSIDE=1) do not perform this check. The two-sided method is in principle more precise, but requires twice as many function evaluations. Default: 0.

.MPDERIV_DEBUG - set this value to 1 to enable debugging of user-supplied explicit derivatives (see "TESTING and DEBUGGING" section above). In addition, the user must enable calculation of explicit derivatives by either setting AUTODERIVATIVE=0, or MPSIDE=3 for the desired parameters. When this option is enabled, a report may be printed to the console, depending on the MPDERIV_ABSTOL and MPDERIV_RELTOL settings. Default: 0 (no debugging)

.MPDERIV_ABSTOL, .MPDERIV_RELTOL - tolerance settings for print-out of debugging information, for each parameter where debugging is enabled. See "TESTING and DEBUGGING" section above for the meanings of these two fields.

.MPMAXSTEP - the maximum change to be made in the parameter value. During the fitting process, the parameter will never be changed by more than this value in one iteration.

A value of 0 indicates no maximum. Default: 0.

.TIED - a string expression which "ties" the parameter to other free or fixed parameters as an equality constraint. Any expression involving constants and the parameter array P are permitted.
Example: if parameter 2 is always to be twice parameter

1 then use the following: `parinfo[2].tied = '2 * P[1]'`. Since they are totally constrained, tied parameters are considered to be fixed; no errors are computed for them, and any LIMITS are not obeyed.

[NOTE: the PARNAME can't be used in a TIED expression.]

`.MPPRINT` - if set to 1, then the default ITERPROC will print the parameter value. If set to 0, the parameter value will not be printed. This tag can be used to selectively print only a few parameter values out of many. Default: 1 (all parameters printed)

`.MPFORMAT` - IDL format string to print the parameter within ITERPROC. Default: '(G20.6)' (An empty string will also use the default.)

Future modifications to the PARINFO structure, if any, will involve adding structure tags beginning with the two letters "MP". Therefore programmers are urged to avoid using tags starting with "MP", but otherwise they are free to include their own fields within the PARINFO structure, which will be ignored by MPFIT.

PARINFO Example:

```
parinfo = replicate({value:0.D, fixed:0, limited:[0,0], $
                    limits:[0.D,0]}, 5)
parinfo[0].fixed = 1
parinfo[4].limited[0] = 1
parinfo[4].limits[0] = 50.D
parinfo[*].value = [5.7D, 2.2, 500., 1.5, 2000.]
```

A total of 5 parameters, with starting values of 5.7, 2.2, 500, 1.5, and 2000 are given. The first parameter is fixed at a value of 5.7, and the last parameter is constrained to be above 50.

RECURSION

Generally, recursion is not allowed. As of version 1.77, MPFIT has recursion protection which does not allow a model function to itself call MPFIT. Users who wish to perform multi-level optimization should investigate the 'EXTERNAL' function evaluation methods described below for hard-to-evaluate functions. That method places more control in the user's hands. The user can design a "recursive" application by taking care.

In most cases the recursion protection should be well-behaved. However, if the user is doing debugging, it is possible for the protection system to get "stuck." In order to reset it, run the procedure:

`MPFIT_RESET_RECURSION`

and the protection system should get "unstuck." It is safe to call this procedure at any time.

COMPATIBILITY

This function is designed to work with IDL 5.0 or greater.

Because TIED parameters and the "(EXTERNAL)" user-model feature use the EXECUTE() function, they cannot be used with the free version of the IDL Virtual Machine.

DETERMINING THE VERSION OF MPFIT

MPFIT is a changing library. Users of MPFIT may also depend on a specific version of the library being present. As of version 1.70 of MPFIT, a VERSION keyword has been added which allows the user to query which version is present. The keyword works like this:

```
RESULT = MPFIT(/query, VERSION=version)
```

This call uses the /QUERY keyword to query the version number without performing any computations. Users of MPFIT can call this method to determine which version is in the IDL path before actually using MPFIT to do any numerical work. Upon return, the VERSION keyword contains the version number of MPFIT, expressed as a string of the form 'X.Y' where X and Y are integers.

Users can perform their own version checking, or use the built-in error checking of MPFIT. The MIN_VERSION keyword enforces the requested minimum version number. For example,

```
RESULT = MPFIT(/query, VERSION=version, MIN_VERSION='1.70')
```

will check whether the accessed version is 1.70 or greater, without performing any numerical processing.

The VERSION and MIN_VERSION keywords were added in MPFIT version 1.70 and later. If the caller attempts to use the VERSION or MIN_VERSION keywords, and an *older* version of the code is present in the caller's path, then IDL will throw an 'unknown keyword' error. Therefore, in order to be robust, the caller, must use exception handling. Here is an example demanding at least version 1.70.

```
MPFIT_OK = 0 & VERSION = '<unknown>'
CATCH, CATCHERR
IF CATCHERR EQ 0 THEN MPFIT_OK = MPFIT(/query, VERSION=version, $
                                     MIN_VERSION='1.70')
CATCH, /CANCEL

IF NOT MPFIT_OK THEN $
  MESSAGE, 'ERROR: you must have MPFIT version 1.70 or higher in '+$
    'your path (found version '+version+')'
```

Of course, the caller can also do its own version number requirements checking.

HARD-TO-COMPUTE FUNCTIONS: "EXTERNAL" EVALUATION

The normal mode of operation for MPFIT is for the user to pass a

function name, and MPFIT will call the user function multiple times as it iterates toward a solution.

Some user functions are particularly hard to compute using the standard model of MPFIT. Usually these are functions that depend on a large amount of external data, and so it is not feasible, or at least highly impractical, to have MPFIT call it. In those cases it may be possible to use the "(EXTERNAL)" evaluation option.

In this case the user is responsible for making all function *and derivative* evaluations. The function and Jacobian data are passed in through the EXTERNAL_FVEC and EXTERNAL_FJAC keywords, respectively. The user indicates the selection of this option by specifying a function name (MYFUNCT) of "(EXTERNAL)". No user-function calls are made when EXTERNAL evaluation is being used.

**** SPECIAL NOTE **** For the "(EXTERNAL)" case, the quirk noted above does not apply. The gradient matrix, EXTERNAL_FJAC, should be comparable to $-\text{FGRAD}(x,p)/\text{err}$, which is the *opposite* sign of the DP matrix described above. In other words, EXTERNAL_FJAC has the same sign as the derivative of EXTERNAL_FVEC, and the opposite sign of FGRAD.

At the end of each iteration, control returns to the user, who must reevaluate the function at its new parameter values. Users should check the return value of the STATUS keyword, where a value of 9 indicates the user should supply more data for the next iteration, and re-call MPFIT. The user may refrain from calling MPFIT further; as usual, STATUS will indicate when the solution has converged and no more iterations are required.

Because MPFIT must maintain its own data structures between calls, the user must also pass a named variable to the EXTERNAL_STATE keyword. This variable must be maintained by the user, but not changed, throughout the fitting process. When no more iterations are desired, the named variable may be discarded.

INPUTS:

MYFUNCT - a string variable containing the name of the function to be minimized. The function should return the weighted deviations between the model and the data, as described above.

For EXTERNAL evaluation of functions, this parameter should be set to a value of "(EXTERNAL)".

START_PARAMS - An one-dimensional array of starting values for each of the parameters of the model. The number of parameters should be fewer than the number of measurements. Also, the parameters should have the same data type as the measurements (double is preferred).

This parameter is optional if the PARINFO keyword is used (but see PARINFO). The PARINFO keyword provides a mechanism to fix or constrain individual

parameters. If both START_PARAMS and PARINFO are passed, then the starting *value* is taken from START_PARAMS, but the *constraints* are taken from PARINFO.

RETURNS:

Returns the array of best-fit parameters.

Exceptions:

* if /QUERY is set (see QUERY).

KEYWORD PARAMETERS:

AUTODERIVATIVE - If this is set, derivatives of the function will be computed automatically via a finite differencing procedure. If not set, then MYFUNCT must provide the explicit derivatives.
Default: set (=1)
NOTE: to supply your own explicit derivatives, explicitly pass AUTODERIVATIVE=0

BESTNORM - upon return, the value of the summed squared weighted residuals for the returned parameter values, i.e. TOTAL(DEVIATES^2).

BEST_FJAC - upon return, BEST_FJAC contains the Jacobian, or partial derivative, matrix for the best-fit model. The values are an array, ARRAY(N_ELEMENTS(DEVIATES),NFREE) where NFREE is the number of free parameters. This array is only computed if /CALC_FJAC is set, otherwise BEST_FJAC is undefined.

The returned array is such that BEST_FJAC[I,J] is the partial derivative of DEVIATES[I] with respect to parameter PARMS[PFREE_INDEX[J]]. Note that since deviates are (data-model)*weight, the Jacobian of the *deviates* will have the opposite sign from the Jacobian of the *model*, and may be scaled by a factor.

BEST_RESID - upon return, an array of best-fit deviates.

CALC_FJAC - if set, then calculate the Jacobian and return it in BEST_FJAC. If not set, then the return value of BEST_FJAC is undefined.

COVAR - the covariance matrix for the set of parameters returned by MPFIT. The matrix is NxN where N is the number of parameters. The square root of the diagonal elements gives the formal 1-sigma statistical errors on the parameters IF errors were treated "properly" in MYFUNC. Parameter errors are also returned in PERROR.

To compute the correlation matrix, PCOR, use this example:
PCOR = COV * 0

```

      FOR i = 0, n-1 DO FOR j = 0, n-1 DO $
        PCOR[i,j] = COV[i,j]/sqrt(COV[i,i]*COV[j,j])
or equivalently, in vector notation,
        PCOR = COV / (PERROR # PERROR)

```

If NOCOVAR is set or MPFIT terminated abnormally, then COVAR is set to a scalar with value !VALUES.D_NAN.

DOF - number of degrees of freedom, computed as

```
DOF = N_ELEMENTS(DEVIATES) - NFREE
```

Note that this doesn't account for pegged parameters (see NPEGGED). It also does not account for data points which are assigned zero weight by the user function.

ERRMSG - a string error or warning message is returned.

EXTERNAL_FVEC - upon input, the function values, evaluated at START_PARAMS. This should be an M-vector, where M is the number of data points.

EXTERNAL_FJAC - upon input, the Jacobian array of partial derivative values. This should be a M x N array, where M is the number of data points and N is the number of parameters. NOTE: that all FIXED or TIED parameters must **not** be included in this array.

EXTERNAL_STATE - a named variable to store MPFIT-related state information between iterations (used in input and output to MPFIT). The user must not manipulate or discard this data until the final iteration is performed.

FASTNORM - set this keyword to select a faster algorithm to compute sum-of-square values internally. For systems with large numbers of data points, the standard algorithm can become prohibitively slow because it cannot be vectorized well. By setting this keyword, MPFIT will run faster, but it will be more prone to floating point overflows and underflows. Thus, setting this keyword may sacrifice some stability in the fitting process.

FTOL - a nonnegative input variable. Termination occurs when both the actual and predicted relative reductions in the sum of squares are at most FTOL (and STATUS is accordingly set to 1 or 3). Therefore, FTOL measures the relative error desired in the sum of squares. Default: 1D-10

FUNCTARGS - A structure which contains the parameters to be passed to the user-supplied function specified by MYFUNCT via the _EXTRA mechanism. This is the way you can pass additional data to your user-supplied function without using common blocks.

Consider the following example:

```
if FUNCTARGS = { XVAL:[1.D,2,3], YVAL:[1.D,4,9],
```



```

ERRVAL:[1.D,1,1] }
then the user supplied function should be declared
like this:
FUNCTION MYFUNCT, P, XVAL=x, YVAL=y, ERRVAL=err

```

By default, no extra parameters are passed to the user-supplied function, but your function should accept **at least** one keyword parameter. [This is to accomodate a limitation in IDL's `_EXTRA` parameter-passing mechanism.]

GTOL - a nonnegative input variable. Termination occurs when the cosine of the angle between fvec and any column of the jacobian is at most GTOL in absolute value (and STATUS is accordingly set to 4). Therefore, GTOL measures the orthogonality desired between the function vector and the columns of the jacobian. Default: 1D-10

ITERARGS - The keyword arguments to be passed to ITERPROC via the `_EXTRA` mechanism. This should be a structure, and is similar in operation to FUNCTARGS.
Default: no arguments are passed.

ITERPRINT - The name of an IDL procedure, equivalent to PRINT, that ITERPROC will use to render output. ITERPRINT should be able to accept at least four positional arguments. In addition, it should be able to accept the standard FORMAT keyword for output formatting; and the UNIT keyword, to redirect output to a logical file unit (default should be UNIT=1, standard output). These keywords are passed using the ITERARGS keyword above. The ITERPRINT procedure must accept the `_EXTRA` keyword.

NOTE: that much formatting can be handled with the MPPRINT and MPFORMAT tags.
Default: 'MPFIT_DEFPRINT' (default internal formatter)

ITERPROC - The name of a procedure to be called upon each NPRINT iteration of the MPFIT routine. ITERPROC is always called in the final iteration. It should be declared in the following way:

```

PRO ITERPROC, MYFUNCT, p, iter, fnorm, FUNCTARGS=fcnargs, $
  PARINFO=parinfo, QUIET=quiet, DOF=dof, PFORMAT=pformat, $
  UNIT=unit, ...
  ; perform custom iteration update
END

```

ITERPROC must either accept all three keyword parameters (FUNCTARGS, PARINFO and QUIET), or at least accept them via the `_EXTRA` keyword.

MYFUNCT is the user-supplied function to be minimized, P is the current set of model parameters, ITER is the iteration number, and FUNCTARGS are the arguments to be passed to MYFUNCT. FNORM should be the chi-squared value. QUIET is set when no textual output should be

printed. DOF is the number of degrees of freedom, normally the number of points less the number of free parameters. See below for documentation of PARINFO. PFORMAT is the default parameter value format. UNIT is passed on to the ITERPRINT procedure, and should indicate the file unit where log output will be sent (default: standard output).

In implementation, ITERPROC can perform updates to the terminal or graphical user interface, to provide feedback while the fit proceeds. If the fit is to be stopped for any reason, then ITERPROC should set the common block variable ERROR_CODE to negative value between -15 and -1 (see MPFIT_ERROR common block below). In principle, ITERPROC should probably not modify the parameter values, because it may interfere with the algorithm's stability. In practice it is allowed.

Default: an internal routine is used to print the parameter values.

ITERSTOP - Set this keyword if you wish to be able to stop the fitting by hitting the predefined ITERKEYSTOP key on the keyboard. This only works if you use the default ITERPROC.

ITERKEYSTOP - A keyboard key which will halt the fit (and if ITERSTOP is set and the default ITERPROC is used). ITERSTOPKEY may either be a one-character string with the desired key, or a scalar integer giving the ASCII code of the desired key.
Default: 7b (control-g)

NOTE: the default value of ASCII 7 (control-G) cannot be read in some windowing environments, so you must change to a printable character like 'q'.

MAXITER - The maximum number of iterations to perform. If the number of calculation iterations exceeds MAXITER, then the STATUS value is set to 5 and MPFIT returns.

If MAXITER EQ 0, then MPFIT does not iterate to adjust parameter values; however, the user function is evaluated and parameter errors/covariance/Jacobian are estimated before returning.
Default: 200 iterations

MIN_VERSION - The minimum requested version number. This must be a scalar string of the form returned by the VERSION keyword. If the current version of MPFIT does not satisfy the minimum requested version number, then,
MPFIT(/query, min_version='...') returns 0
MPFIT(...) returns NAN
Default: no version number check
NOTE: MIN_VERSION was added in MPFIT version 1.70

NFEV - the number of MYFUNCT function evaluations performed.

NFREE - the number of free parameters in the fit. This includes parameters which are not FIXED and not TIED, but it does include parameters which are pegged at LIMITS.

NITER - the number of iterations completed.

NOCATCH - if set, then MPFIT will not perform any error trapping. By default (not set), MPFIT will trap errors and report them to the caller. This keyword will typically be used for debugging.

NOCOVAR - set this keyword to prevent the calculation of the covariance matrix before returning (see COVAR)

NPEGGED - the number of free parameters which are pegged at a LIMIT.

NPRINT - The frequency with which ITERPROC is called. A value of 1 indicates that ITERPROC is called with every iteration, while 2 indicates every other iteration, etc. Be aware that several Levenberg-Marquardt attempts can be made in a single iteration. Also, the ITERPROC is **always** called for the final iteration, regardless of the iteration number.
Default value: 1

PARINFO - A one-dimensional array of structures.
Provides a mechanism for more sophisticated constraints to be placed on parameter values. When PARINFO is not passed, then it is assumed that all parameters are free and unconstrained. Values in PARINFO are never modified during a call to MPFIT.

See description above for the structure of PARINFO.

Default value: all parameters are free and unconstrained.

PERROR - The formal 1-sigma errors in each parameter, computed from the covariance matrix. If a parameter is held fixed, or if it touches a boundary, then the error is reported as zero.

If the fit is unweighted (i.e. no errors were given, or the weights were uniformly set to unity), then PERROR will probably not represent the true parameter uncertainties.

If you can assume that the true reduced chi-squared value is unity -- meaning that the fit is implicitly assumed to be of good quality -- then the estimated parameter uncertainties can be computed by scaling PERROR by the measured chi-squared value.

DOF = N_ELEMENTS(X) - N_ELEMENTS(PARMS) ; deg of freedom

PCERROR = PERROR * SQRT(BESTNORM / DOF) ; scaled
uncertainties

PFREE_INDEX - upon return, PFREE_INDEX contains an index array
which indicates which parameter were allowed to
vary. I.e. of all the parameters PARMS, only
PARMS[PFREE_INDEX] were varied.

QUERY - if set, then MPFIT() will return immediately with one of
the following values:
1 - if MIN_VERSION is not set
1 - if MIN_VERSION is set and MPFIT satisfies the minimum
0 - if MIN_VERSION is set and MPFIT does not satisfy it
The VERSION output keyword is always set upon return.
Default: not set.

QUIET - set this keyword when no textual output should be printed
by MPFIT

RESDAMP - a scalar number, indicating the cut-off value of
residuals where "damping" will occur. Residuals with
magnitudes greater than this number will be replaced by
their logarithm. This partially mitigates the so-called
large residual problem inherent in least-squares solvers
(as for the test problem CURVI, <http://www.maxthis.com/-curviex.htm>). A value of 0 indicates no damping.
Default: 0

Note: RESDAMP doesn't work with AUTODERIV=0

STATUS - an integer status code is returned. All values greater
than zero can represent success (however STATUS EQ 5 may
indicate failure to converge). It can have one of the
following values:

-18 a fatal execution error has occurred. More information
may be available in the ERRMSG string.

-16 a parameter or function value has become infinite or an
undefined number. This is usually a consequence of
numerical overflow in the user's model function, which
must be avoided.

-15 to -1

these are error codes that either MYFUNCT or ITERPROC
may return to terminate the fitting process (see
description of MPFIT_ERROR common below). If either
MYFUNCT or ITERPROC set ERROR_CODE to a negative number,
then that number is returned in STATUS. Values from -15
to -1 are reserved for the user functions and will not
clash with MPFIT.

0 improper input parameters.

1 both actual and predicted relative reductions
in the sum of squares are at most FTOL.

- 2 relative error between two consecutive iterates is at most XTOL
- 3 conditions for STATUS = 1 and STATUS = 2 both hold.
- 4 the cosine of the angle between fvec and any column of the jacobian is at most GTOL in absolute value.
- 5 the maximum number of iterations has been reached
- 6 FTOL is too small. no further reduction in the sum of squares is possible.
- 7 XTOL is too small. no further improvement in the approximate solution x is possible.
- 8 GTOL is too small. fvec is orthogonal to the columns of the jacobian to machine precision.
- 9 A successful single iteration has been completed, and the user must supply another "EXTERNAL" evaluation of the function and its derivatives. This status indicator is neither an error nor a convergence indicator.

VERSION - upon return, VERSION will be set to the MPFIT internal version number. The version number will be a string of the form "X.Y" where X is a major revision number and Y is a minor revision number.

NOTE: the VERSION keyword was not present before MPFIT version number 1.70, therefore, callers must use exception handling when using this keyword.

XTOL - a nonnegative input variable. Termination occurs when the relative error between two consecutive iterates is at most XTOL (and STATUS is accordingly set to 2 or 3). Therefore, XTOL measures the relative error desired in the approximate solution. Default: 1D-10

EXAMPLE:

```
p0 = [5.7D, 2.2, 500., 1.5, 2000.]
fa = {X:x, Y:y, ERR:err}
p = mpfit('MYFUNCT', p0, functargs=fa)
```

Minimizes sum of squares of MYFUNCT. MYFUNCT is called with the X, Y, and ERR keyword parameters that are given by FUNCTARGS. The resulting parameter values are returned in p.

COMMON BLOCKS:

COMMON MPFIT_ERROR, ERROR_CODE

User routines may stop the fitting process at any time by setting an error condition. This condition may be set in either

the user's model computation routine (MYFUNCT), or in the iteration procedure (ITERPROC).

To stop the fitting, the above common block must be declared, and ERROR_CODE must be set to a negative number. After the user procedure or function returns, MPFIT checks the value of this common block variable and exits immediately if the error condition has been set. This value is also returned in the STATUS keyword: values of -1 through -15 are reserved error codes for the user routines. By default the value of ERROR_CODE is zero, indicating a successful function/procedure call.

```
COMMON MPFIT_PROFILE
COMMON MPFIT_MACHAR
COMMON MPFIT_CONFIG
```

These are undocumented common blocks are used internally by MPFIT and may change in future implementations.

THEORY OF OPERATION:

There are many specific strategies for function minimization. One very popular technique is to use function gradient information to realize the local structure of the function. Near a local minimum the function value can be Taylor expanded about x_0 as follows:

$$f(x) = \underbrace{f(x_0)}_{\text{Order 0th}} + \underbrace{f'(x_0) \cdot (x-x_0)}_{\text{Order 1st}} + \underbrace{(1/2) (x-x_0) \cdot f''(x_0) \cdot (x-x_0)}_{\text{Order 2nd}} \quad (1)$$

Here $f'(x)$ is the gradient vector of f at x , and $f''(x)$ is the Hessian matrix of second derivatives of f at x . The vector x is the set of function parameters, not the measured data vector. One can find the minimum of f , $f(x_m)$ using Newton's method, and arrives at the following linear equation:

$$f''(x_0) \cdot (x_m - x_0) = -f'(x_0) \quad (2)$$

If an inverse can be found for $f''(x_0)$ then one can solve for $(x_m - x_0)$, the step vector from the current position x_0 to the new projected minimum. Here the problem has been linearized (ie, the gradient information is known to first order). $f''(x_0)$ is symmetric $n \times n$ matrix, and should be positive definite.

The Levenberg - Marquardt technique is a variation on this theme. It adds an additional diagonal term to the equation which may aid the convergence properties:

$$(f''(x_0) + \nu I) \cdot (x_m - x_0) = -f'(x_0) \quad (2a)$$

where I is the identity matrix. When ν is large, the overall matrix is diagonally dominant, and the iterations follow steepest descent. When ν is small, the iterations are quadratically convergent.

In principle, if $f''(x_0)$ and $f'(x_0)$ are known then $x_m - x_0$ can be determined. However the Hessian matrix is often difficult or

impossible to compute. The gradient $f'(x_0)$ may be easier to compute, if even by finite difference techniques. So-called quasi-Newton techniques attempt to successively estimate $f''(x_0)$ by building up gradient information as the iterations proceed.

In the least squares problem there are further simplifications which assist in solving eqn (2). The function to be minimized is a sum of squares:

$$f = \text{Sum}(h_i^2) \quad (3)$$

where h_i is the i th residual out of m residuals as described above. This can be substituted back into eqn (2) after computing the derivatives:

$$\begin{aligned} f' &= 2 \text{Sum}(h_i \quad h_i') \\ f'' &= 2 \text{Sum}(h_i' \quad h_j') + 2 \text{Sum}(h_i \quad h_i'') \end{aligned} \quad (4)$$

If one assumes that the parameters are already close enough to a minimum, then one typically finds that the second term in f'' is negligible [or, in any case, is too difficult to compute]. Thus, equation (2) can be solved, at least approximately, using only gradient information.

In matrix notation, the combination of eqns (2) and (4) becomes:

$$h^T \cdot h' \cdot dx = - h^T \cdot h \quad (5)$$

Where h is the residual vector (length m), h^T is its transpose, h' is the Jacobian matrix (dimensions $n \times m$), and dx is $(x_m - x_0)$. The user function supplies the residual vector h , and in some cases h' when it is not found by finite differences (see MPFIT_FDJAC2, which finds h and h'). Even if dx is not the best absolute step to take, it does provide a good estimate of the best *direction*, so often a line minimization will occur along the dx vector direction.

The method of solution employed by MINPACK is to form the $Q \cdot R$ factorization of h' , where Q is an orthogonal matrix such that $Q^T \cdot Q = I$, and R is upper right triangular. Using $h' = Q \cdot R$ and the orthogonality of Q , eqn (5) becomes

$$\begin{aligned} (R^T \cdot Q^T) \cdot (Q \cdot R) \cdot dx &= - (R^T \cdot Q^T) \cdot h \\ R^T \cdot R \cdot dx &= - R^T \cdot Q^T \cdot h \\ R \cdot dx &= - Q^T \cdot h \end{aligned} \quad (6)$$

where the last statement follows because R is upper triangular. Here, R , Q^T and h are known so this is a matter of solving for dx . The routine MPFIT_QRFAC provides the QR factorization of h' , with pivoting, and MPFIT_QRSOL;V provides the solution for dx .

REFERENCES:

Markwardt, C. B. 2008, "Non-Linear Least Squares Fitting in IDL with MPFIT," in proc. Astronomical Data Analysis Software and Systems XVIII, Quebec, Canada, ASP Conference Series, Vol. XXX, eds. D. Bohlender, P. Dowler & D. Durand (Astronomical Society of the

Pacific: San Francisco), p. 251-254 (ISBN: 978-1-58381-702-5)
<http://arxiv.org/abs/0902.2850>
Link to NASA ADS: <http://adsabs.harvard.edu/abs/2009ASPC..411..251M>
Link to ASP: http://aspbooks.org/a/volumes/table_of_contents/411

Refer to the MPFIT website as:
<http://purl.com/net/mpfit>

MINPACK-1 software, by Jorge More' et al, available from netlib.
<http://www.netlib.org/>

"Optimization Software Guide," Jorge More' and Stephen Wright,
SIAM, *Frontiers in Applied Mathematics*, Number 14.
(ISBN: 978-0-898713-22-0)

More', J. 1978, "The Levenberg-Marquardt Algorithm: Implementation
and Theory," in Numerical Analysis, vol. 630, ed. G. A. Watson
(Springer-Verlag: Berlin), p. 105 (DOI: 10.1007/BFb0067690)

MODIFICATION HISTORY:

Translated from MINPACK-1 in FORTRAN, Apr-Jul 1998, CM
Fixed bug in parameter limits (x vs xnew), 04 Aug 1998, CM
Added PERROR keyword, 04 Aug 1998, CM
Added COVAR keyword, 20 Aug 1998, CM
Added NITER output keyword, 05 Oct 1998
D.L Windt, Bell Labs, windt@bell-labs.com;
Made each PARINFO component optional, 05 Oct 1998 CM
Analytical derivatives allowed via AUTODERIVATIVE keyword, 09 Nov 1998
Parameter values can be tied to others, 09 Nov 1998
Fixed small bugs (Wayne Landsman), 24 Nov 1998
Added better exception error reporting, 24 Nov 1998 CM
Cosmetic documentation changes, 02 Jan 1999 CM
Changed definition of ITERPROC to be consistent with TNMIN, 19 Jan 1999 CM
Fixed bug when AUTDERIVATIVE=0. Incorrect sign, 02 Feb 1999 CM
Added keyboard stop to MPFIT_DEFITER, 28 Feb 1999 CM
Cosmetic documentation changes, 14 May 1999 CM
IDL optimizations for speed & FASTNORM keyword, 15 May 1999 CM
Tried a faster version of mpfit_enorm, 30 May 1999 CM
Changed web address to cow.physics.wisc.edu, 14 Jun 1999 CM
Found malformation of FDJAC in MPFIT for 1 parm, 03 Aug 1999 CM
Factored out user-function call into MPFIT_CALL. It is possible,
but currently disabled, to call procedures. The calling format
is similar to CURVEFIT, 25 Sep 1999, CM
Slightly changed mpfit_tie to be less intrusive, 25 Sep 1999, CM
Fixed some bugs associated with tied parameters in mpfit_fdjac, 25
Sep 1999, CM
Reordered documentation; now alphabetical, 02 Oct 1999, CM
Added QUERY keyword for more robust error detection in drivers, 29
Oct 1999, CM
Documented PERROR for unweighted fits, 03 Nov 1999, CM
Split out MPFIT_RESETPROF to aid in profiling, 03 Nov 1999, CM
Some profiling and speed optimization, 03 Nov 1999, CM
Worst offenders, in order: fdjac2, qrfac, qrsolv, enorm.
fdjac2 depends on user function, qrfac and enorm seem to be
fully optimized. qrsolv probably could be tweaked a little, but
is still <10% of total compute time.
Made sure that !err was set to 0 in MPFIT_DEFITER, 10 Jan 2000, CM

Fixed small inconsistency in setting of QANYLIM, 28 Jan 2000, CM
 Added PARINFO field RELSTEP, 28 Jan 2000, CM
 Converted to MPFIT_ERROR common block for indicating error conditions, 28 Jan 2000, CM
 Corrected scope of MPFIT_ERROR common block, CM, 07 Mar 2000
 Minor speed improvement in MPFIT_ENORM, CM 26 Mar 2000
 Corrected case where ITERPROC changed parameter values and parameter values were TIED, CM 26 Mar 2000
 Changed MPFIT_CALL to modify NFEV automatically, and to support user procedures more, CM 26 Mar 2000
 Copying permission terms have been liberalized, 26 Mar 2000, CM
 Catch zero value of zero a(j,lj) in MPFIT_QRFAC, 20 Jul 2000, CM
 (thanks to David Schlegel <schlegel@astro.princeton.edu>)
 MPFIT_SETMACHAR is called only once at init; only one common block is created (MPFIT_MACHAR); it is now a structure; removed almost all CHECK_MATH calls for compatibility with IDL5 and !EXCEPT; profiling data is now in a structure too; noted some mathematical discrepancies in Linux IDL5.0, 17 Nov 2000, CM
 Some significant changes. New PARINFO fields: MPSIDE, MPMINSTEP, MPMAXSTEP. Improved documentation. Now PTIED constraints are maintained in the MPCONFIG common block. A new procedure to parse PARINFO fields. FDJAC2 now computes a larger variety of one-sided and two-sided finite difference derivatives. NFEV is stored in the MPCONFIG common now. 17 Dec 2000, CM
 Added check that PARINFO and XALL have same size, 29 Dec 2000 CM
 Don't call function in TERMINATE when there is an error, 05 Jan 2000
 Check for float vs. double discrepancies; corrected implementation of MIN/MAXSTEP, which I still am not sure of, but now at least the correct behavior occurs *without* it, CM 08 Jan 2001
 Added SCALE_FCN keyword, to allow for scaling, as for the CASH statistic; added documentation about the theory of operation, and under the QR factorization; slowly I'm beginning to understand the bowels of this algorithm, CM 10 Jan 2001
 Remove MPMINSTEP field of PARINFO, for now at least, CM 11 Jan 2001
 Added RESDAMP keyword, CM, 14 Jan 2001
 Tried to improve the DAMP handling a little, CM, 13 Mar 2001
 Corrected .PARNAME behavior in _DEFITER, CM, 19 Mar 2001
 Added checks for parameter and function overflow; a new STATUS value to reflect this; STATUS values of -15 to -1 are reserved for user function errors, CM, 03 Apr 2001
 DAMP keyword is now a TANH, CM, 03 Apr 2001
 Added more error checking of float vs. double, CM, 07 Apr 2001
 Fixed bug in handling of parameter lower limits; moved overflow checking to end of loop, CM, 20 Apr 2001
 Failure using GOTO, TERMINATE more graceful if FNORM1 not defined, CM, 13 Aug 2001
 Add MPPRINT tag to PARINFO, CM, 19 Nov 2001
 Add DOF keyword to DEFITER procedure, and print degrees of freedom, CM, 28 Nov 2001
 Add check to be sure MYFUNCT is a scalar string, CM, 14 Jan 2002
 Addition of EXTERNAL_FJAC, EXTERNAL_FVEC keywords; ability to save fitter's state from one call to the next; allow '(EXTERNAL)' function name, which implies that user will supply function and Jacobian at each iteration, CM, 10 Mar 2002
 Documented EXTERNAL evaluation code, CM, 10 Mar 2002

Corrected significant bug in the way that the STEP parameter, and
 FIXED parameters interacted (Thanks Andrew Steffl), CM, 02 Apr
 2002

Allow COVAR and PERROR keywords to be computed, even in case of
 '(EXTERNAL)' function, 26 May 2002

Add NFREE and NPEGGED keywords; compute NPEGGED; compute DOF using
 NFREE instead of n_elements(X), thanks to Kristian Kjaer, CM 11
 Sep 2002

Hopefully PERROR is all positive now, CM 13 Sep 2002

Documented RELSTEP field of PARINFO (!!), CM, 25 Oct 2002

Error checking to detect missing start pars, CM 12 Apr 2003

Add DOF keyword to return degrees of freedom, CM, 30 June 2003

Always call ITERPROC in the final iteration; add ITERKEYSTOP
 keyword, CM, 30 June 2003

Correct bug in MPFIT_LMPAR of singularity handling, which might
 likely be fatal for one-parameter fits, CM, 21 Nov 2003
 (with thanks to Peter Tuthill for the proper test case)

Minor documentation adjustment, 03 Feb 2004, CM

Correct small error in QR factorization when pivoting; document
 the return values of QRFAC when pivoting, 21 May 2004, CM

Add MPFORMAT field to PARINFO, and correct behavior of interaction
 between MPPRINT and PARNAME in MPFIT_DEFITERPROC (thanks to Tim
 Robishaw), 23 May 2004, CM

Add the ITERPRINT keyword to allow redirecting output, 26 Sep
 2004, CM

Correct MAXSTEP behavior in case of a negative parameter, 26 Sep
 2004, CM

Fix bug in the parsing of MINSTEP/MAXSTEP, 10 Apr 2005, CM

Fix bug in the handling of upper/lower limits when the limit was
 negative (the fitting code would never "stick" to the lower
 limit), 29 Jun 2005, CM

Small documentation update for the TIED field, 05 Sep 2005, CM

Convert to IDL 5 array syntax (!), 16 Jul 2006, CM

If MAXITER equals zero, then do the basic parameter checking and
 uncertainty analysis, but do not adjust the parameters, 15 Aug
 2006, CM

Added documentation, 18 Sep 2006, CM

A few more IDL 5 array syntax changes, 25 Sep 2006, CM

Move STRICTARR compile option inside each function/procedure, 9 Oct 2006

Bug fix for case of MPMAXSTEP and fixed parameters, thanks
 to Huib Intema (who found it from the Python translation!), 05 Feb 2007

Similar fix for MPFIT_FDjac2 and the MPSIDE sidedness of
 derivatives, also thanks to Huib Intema, 07 Feb 2007

Clarify documentation on user-function, derivatives, and PARINFO,
 27 May 2007

Change the wording of "Analytic Derivatives" to "Explicit
 Derivatives" in the documentation, CM, 03 Sep 2007

Further documentation tweaks, CM, 13 Dec 2007

Add COMPATIBILITY section and add credits to copyright, CM, 13 Dec
 2007

Document and enforce that START_PARS and PARINFO are 1-d arrays,
 CM, 29 Mar 2008

Previous change for 1-D arrays wasn't correct for
 PARINFO.LIMITED/.LIMITS; now fixed, CM, 03 May 2008

Documentation adjustments, CM, 20 Aug 2008

Change some minor FOR-loop variables to type-long, CM, 03 Sep 2008

Change error handling slightly, document NOCATCH keyword,

document error handling in general, CM, 01 Oct 2008
 Special case: when either LIMITS is zero, and a parameter pushes against that limit, the coded that 'pegged' it there would not work since it was a relative condition; now zero is handled properly, CM, 08 Nov 2008
 Documentation of how TIED interacts with LIMITS, CM, 21 Dec 2008
 Better documentation of references, CM, 27 Feb 2009
 If MAXITER=0, then be sure to set STATUS=5, which permits the the covariance matrix to be computed, CM, 14 Apr 2009
 Avoid numerical underflow while solving for the LM parameter, (thanks to Sergey Koposov) CM, 14 Apr 2009
 Use individual functions for all possible MPFIT_CALL permutations, (and make sure the syntax is right) CM, 01 Sep 2009
 Correct behavior of MPMAXSTEP when some parameters are frozen, thanks to Josh Destree, CM, 22 Nov 2009
 Update the references section, CM, 22 Nov 2009
 1.70 - Add the VERSION and MIN_VERSION keywords, CM, 22 Nov 2009
 1.71 - Store pre-calculated revision in common, CM, 23 Nov 2009
 1.72-1.74 - Documented alternate method to compute correlation matrix, CM, 05 Feb 2010
 1.75 - Enforce TIED constraints when preparing to terminate the routine, CM, 2010-06-22
 1.76 - Documented input keywords now are not modified upon output, CM, 2010-07-13
 1.77 - Upon user request (/CALC_FJAC), compute Jacobian matrix and return in BEST_FJAC; also return best residuals in BEST_RESID; also return an index list of free parameters as PFREE_INDEX; add a fencepost to prevent recursion CM, 2010-10-27
 1.79 - Documentation corrections. CM, 2011-08-26
 1.81 - Fix bug in interaction of AUTODERIVATIVE=0 and .MPSIDE=3; Document FJAC_MASK. CM, 2012-05-08

Add accomodation for PDIF_ITER_DISPLAY and PDIF_ITER_COUNT tagnames to the FCNARGS keyword. If FCNARGS.PDIF_ITER_DISPLAY is set to 1 when mpfit is called, then FCNARGS.PDIF_ITER_COUNT is set during computation of partial derivatives to the index of the partial derivative being computed. This allows the calling program to display the iteration count in real time. 13-May-2013. D. L. Windt, Reflective X-ray Optics, davidwindt@gmail.com

\$Id: mpfit.pro,v 1.82 2012/09/27 23:59:44 cmarkwar Exp \$

(See ./mpfit.pro)

OEPLLOT

[\[Previous Routine\]](#) [\[Next Routine\]](#) [\[List of Routines\]](#)

NAME :

OEPLOT

PURPOSE:

Overplot x vs y, with vertical error bars on y.

CALLING SEQUENCE:

```
OELOT,Y,SIGY
OELOT,X,Y,SIGY
OELOT,Y,SIGY_UP,SIGY_DOWN
OELOT,X,Y,SIGY_UP,SIGY_DOWN
```

INPUTS:

X, Y - 1-D arrays

SIGY - Uncertainty in y, i.e. $Y \pm \text{SIGY}$

SIGY_UP, SIGY_DOWN - +/- uncertainties in Y,
i.e., $Y + \text{SIGY_UP} - \text{SIGY_DOWN}$

KEYWORD PARAMETERS:

BARLINESTYLE = Linestyle for error bars.

plus the IDL keywords color, linestyle, thick, psym,
symsize, noclip, and t3d.

MODIFICATION HISTORY:

D. L. Windt, Bell Laboratories, November 1989
windt@bell-labs.com

(See ./oeplot.pro)

PLOT_MOVIE

[\[Previous Routine\]](#) [\[Next Routine\]](#) [\[List of Routines\]](#)

NAME:

PLOT_MOVIE

PURPOSE:

Display an animated sequence of X-Y plots in a window.

CALLING SEQUENCE:

```
PLOT_MOVIE,X,Y[,Y1,Y2,Y3,Y4]
```

INPUT PARAMETERS:

X = N_x-element array, common to all Y vectors

Y = 2D array, N_x x N_plots

OPTIONAL INPUT PARAMETERS:

Y1, Y2, Y3, Y4 - additional Y arrays to be overplotted; these must all have the same dimensions as Y.

KEYWORD PARAMETERS:

XRANGE- A two-element vector specifying the xrange of the plot. Default = [0, max(x)]

YRANGE- A two-element vector specifying the yrange of the plot. Default = [0 < min(y)*1.05,max(y)*1.05]

COLOR- array of colors for each Y plot

LINESTYLE - array of linestyles for each Y plot

THICK - array of thicknesses for each Y plot

_EXTRA - This keyword is use to pass additional parameters to the plot command.

EXAMPLE:

Make a movie of two 'travelling' sin waves:

```
X=VECTOR(0.,100.,100)
Y=FLTARR(100,30)
for i=0,29 do Y(*,i)=sin((x+i*!pi)/!pi)
Y1=-Y
PLOT_MOVIE,X,Y,Y1
```

MODIFICATION HISTORY:

Written by: David L. Windt, Bell Labs, April 1994

windt@bell-labs.com

(See ./plot_movie.pro)

PLOT_PRINT

[\[Previous Routine\]](#) [\[Next Routine\]](#) [\[List of Routines\]](#)

NAME:

PLOT_PRINT

PURPOSE:

A widget-based interface for creating and printing IDL graphics output files. The widget allows the user to select an output device type (PS, PCL, or HP), and specify whether or not to use color, the color depth (for PS), the plot orientation (landscape or portrait), the size of the plot, whether or not to use Vector or PS fonts, the name of the file to create, whether to send the file to a printer, and the print command.

CALLING SEQUENCE:

PLOT_PRINT, PLOT_PROCEDURE

INPUTS:

PLOT_PROCEDURE - A string containing the name of the procedure - or the executable IDL code - that creates the desired graphics. See EXAMPLE below for more details.

KEYWORD PARAMETERS:

PRINTPARS - a structure of the following form (default values indicated), whose values are used to set the initial value of this quasi-compound widget:

```
PRINTPARS={ $
  device:0, $ ; 0=PS, 1=HP, 2=PCL, 3=CGM
  psfont:0, $ ; 0=use vector fonts, 1=use PS fonts.
  color:1, $ ; 0=B&W, 1=color
  depth:2, $ ; PS resolution: 0=>1, 1=>2, 2=>4, 4=>8.
  orient:0, $ ; 0=portrait, 1=landscape
  size:1, $ ; 0=small, 1=large, 2=custom (i.e., use
    plotsize)
  plotsize:[xsize,ysize,xoffset,yoffset], $
    ; keyword to the DEVICE command,
    ; in inches. only used if size=2.
  file_name:'idl.ps', $ ; default file name
  print:1, $ ; 0=print only to file, 1=send to printer.
  command:'lpr -Plp' ; print command
```

GROUP - the standard GROUP_LEADER keyword, which is passed directly to XMANAGER when the PLOT_PRINT widget is created.

COMMON BLOCKS:

PLOT_PRINT, plot_printpars

where plot_printpars = the current state of the printpars structure.

SIDE EFFECTS:

The returned value of the printpars structure, if passed, is changed to reflect the settings changes made by the

user. Thus, settings shown in the widget upon subsequent calls to `plot_print` with the same `printpars` structure will show the same settings as the last call to `PLOT_PRINT`.

RESTRICTIONS:

Requires widgets. Requires several programs in the `windt` library, including a modified version of `CW_FIELD`.

The `PLOT_PRINT` widget is modal.

EXAMPLE:

This program is intended to be used from within another widget application, where a procedure is already defined that creates the graphics. You can then add a `WIDGET_BUTTON` to this application, labelled "Print", for example, that when pressed calls `PLOT_PRINT`, with the name of the plot creation procedure as the input. i.e., pressing the "Print" button would execute the IDL code `< PLOT_PRINT,"myplot" >`.

For example:

```
PRO MYPLOT_EV,event

widget_control,event.id,get_uvalue=eventval
if eventval eq 'print' then plot_print,'myplot_plot'
if eventval eq 'done' then widget_control,event.top,/destroy
return
end

PRO MYPLOT_PLOT

plot,[1,2],title='My Plot'
return
end

PRO MYPLOT

base=widget_base(mbar=menubar)
file=widget_button(menubar,/menu,value='File')
print=widget_button(file,value='Print...',uvalue='print')
done=widget_button(file,value='Quit',uvalue='done')
window=widget_draw(base,xsize=400,ysize=300)
widget_control,base,/realize
myplot_plot
xmanager,'myplot',base,event='myplot_ev'
return
end
```

Of course, you can call the program right from the command line too, as in

```
IDL> plot_print,'x=vector(0.,!pi,100) & y=sin(5*x) &
plot,x,y,/xstyle'
```

MODIFICATION HISTORY:

David L. Windt, Bell Labs, March 1997

May 1997 - Modified use of MODAL keyword to work with changes in IDL V5.0.

June 1997 - Changed text and field widgets so that it's no longer necessary to hit <return> after entering text. But this requires use of the modified CW_FIELD widget.

January 1998 - Added support for CGM graphics; switched plot_print.device values for HP and PCL output, to be consistent with the SP.PRO routine.

March 1998 - Made some attempt to include a better default print command for HP-UX and Win95 platforms.

May 1998 - The user is now prompted before attempting to write over an existing file.

January 2004 - Various cosmetics.

windt@bell-labs.com

DLW, June 2003

Slight change to label widget displaying status.

windt@astro.columbia.edu

DLW, May 2013 - Improved permission error handling.

davidwindt@gmail.com

(See ./plot_print.pro)

PLOT_TEXT

[\[Previous Routine\]](#) [\[Next Routine\]](#) [\[List of Routines\]](#)

NAME:

PLOT_TEXT

PURPOSE:

Add text in a box to a plot. The text is located in one of 12 possible positions (i.e., upper left corner, lower right corner, etc.)

CALLING SEQUENCE:

PLOT_TEXT,TEXT_ARRAY[,POSITION=POSITION]

INPUTS:

TEXT_ARRAY - a string array of text

KEYWORD PARAMETERS:

POSITION - an integer, specifying the location of the text box:

0: no text is drawn
1: below plot, left
2: below plot, center
3: below plot, right
4: lower left
5: lower center
6: lower right
7: middle left
8: middle center
9: middle right
10: upper left
11: upper center
12: upper right

if not specified, default position=10

CHARSIZE - the charsize value for the text

COLOR - an array of colors to be used for each line of text

NOBOX - set to inhibit drawing a box around the text

BOXPADX - padding in character units, between text and box,
in x. default=2.0

BOXPADY - padding in character units, between text and box,
in y. default=0.5

FONT - Set to an integer from 3 to 20, corresponding to the
Hershey vector font sets, referring to the font used
to display the text. If a font other than !3 is used
in the text string, then FONT should be set
accordingly. (Any font commands embedded in the text
string are ignored.)

BOXFUDGE - A scaling factor, used to fudge the width of the
box surrounding the text. Default=1.0.

BOXCOLOR - set to the color index used to draw the box.
Default is !P.COLOR.

BOXFILL - set to the color index used to fill the box. Omit,
or set to -1 for no fill. No effect if NOBOX=1.

Plus all valid graphics keywords for xyouts and plots

RESTRICTIONS:

When specifying a position of 1,2 or 3, you'll need to (a) use the same charsize value for the plot and for the plot_text, and (b) draw the plot with an extra ymargin(0). i.e., set ymargin(0)=7+n_elements(text_array)

MODIFICATION HISTORY:

David L. Windt, Bell Labs, March 1997
windt@bell-labs.com

May 2011, dlw:

Now using WIDTH keyword from XYOUTS to do an even better job of drawing the box.

October, 1997, dlw:

Now using the TEXT_WIDTH function, in order to do a somewhat better job of drawing the box around the text.

NONPRINTER_SCALE keyword parameter is now obsolete.

BOXFUDGEEX keyword parameter added.

May 2013 - Added BOXCOLOR and BOXFILL.
DLW, davidwindt@gmail.com

(See ./plot_text.pro)

PROFILE_NI

[\[Previous Routine\]](#) [\[Next Routine\]](#) [\[List of Routines\]](#)

NAME:

PROFILE_NI

PURPOSE:

Extract a profile from an image noninteractively.

CALLING SEQUENCE:

Result = PROFILE_NI(IMAGE,COORDS)

INPUTS:

IMAGE - data array. May be any type except complex.

COORDS - 2 x 2 array of x and y coordinate of profile endpoints,
[[X0,Y0],[X1,Y1]]

KEYWORD PARAMETERS:

XSTART, YSTART - starting (x,y) location of lower left corner
of image.

OUTPUTS:

Result = 1-D array of image values along the line from

MODIFICATION HISTORY:

Adapted from PROFILE

D. L. Windt, Bell Laboratories, November 1991.
windt@bell-labs.com

(See ./profile_ni.pro)

PTRS_NEW

[\[Previous Routine\]](#) [\[Next Routine\]](#) [\[List of Routines\]](#)

NAME:

PTRS_NEW

PURPOSE:

This function will make a copy of the passed variable, which can be of any type. However if it's a pointer, an array of pointers, a structure, or any array of structures, then the entire hierarchy will be searched until all pointers are replaced with new pointers, so that the copy will not point to anything that the original passed variable points to.

CALLING SEQUENCE:

DEST=PTRS_NEW(SOURCE)

INPUTS:

SOURCE: Variable of any type.

OUTPUTS:

DEST: the destination variable, which is an exact copy of the source variable SOURCE, except that all pointers contained in the returned variable will be new.

MODIFICATION HISTORY:

David L. Windt, Reflective X-ray Optics, davidwindt@gmail.com
14-May-2013

(See ./ptrs_new.pro)

PWD

[\[Previous Routine\]](#) [\[Next Routine\]](#) [\[List of Routines\]](#)

NAME:

PWD

CATEGORY:

Stupid little convenience routines.

PURPOSE:

Print the current directory, like the Unix 'pwd' command.

CALLING SEQUENCE:

PWD

MODIFICATION HISTORY:

David L. Windt, Bell Labs, February 1998
windt@bell-labs.com

(See ./pwd.pro)

RECROI

[\[Previous Routine\]](#) [\[Next Routine\]](#) [\[List of Routines\]](#)

NAME:

RECROI

PURPOSE:

Define a rectangular region of interest of an image using the
image display system and the cursor/mouse.

CATEGORY:

Image processing.

CALLING SEQUENCE:

Result=RECROI(SX,SY[,XVERTS,YVERTS])

INPUTS:

SX, SY = size of image, in pixels.

KEYWORD PARAMETERS:

X0, Y0 - coordinate of lower left corner of image on display.
if omitted, (0,0) is assumed. Screen device
coordinates.

ZOOM - zoom factor, if omitted, 1 is assumed.

XAXIS, YAXIS - optional 1-d arrays corresponding to the x and
y scales of image. Needed only if XROI and/or
YROI are specified.

XROI, YROI - optional output vectors associated with the
digitized rectangular region of interest. XAXIS
and YAXIS keyword parameters must be supplied.

OUTPUTS:

Result = vector of subscripts of pixels inside the region.

OPTIONAL OUTPUTS:

XVERTS, YVERTS - optional output parameters which will contain
the vertices enclosing the region. Setting
NOREGION inhibits the return of the pixel
subscripts.

COMMON BLOCKS:

Colors is used to obtain the current color table which is modified
and then restored.

SIDE EFFECTS:

For this implementation, bit 0 of each pixel is used to draw ; the
outline of the region. You WILL have to change this to fit
the capabilities and procedures of your display. ; The lowest
bit in which the write mask is enabled is changed.

PROCEDURE:

The write mask for the display is set so that only bit 0 may be
written. Bit 0 is erased for all pixels. The color tables
are loaded with odd values complemented, even values
unchanged. A message is printed, assuming a mouse, indicating

the effect of the three buttons. The operator marks opposite corners of the rectangle.

MODIFICATION HISTORY:

Adapted from DEFROI

D. L. Windt, Bell Laboratories, November 1989
windt@bell-labs.com

(See [./recroi.pro](#))

RECTANGLE

[\[Previous Routine\]](#) [\[Next Routine\]](#) [\[List of Routines\]](#)

NAME:

RECTANGLE

PURPOSE:

Draw a rectangle on a plot.

CALLING SEQUENCE:

RECTANGLE,X0,Y0,XLENGTH,YLENGTH

INPUTS:

X0, Y0 - Points specifying a corner of the rectangle.

XLENGTH, YLENGTH - the lengths of the sides of the rectangle,
in data coords.

KEYWORD PARAMETERS:

FILL = set to fill rectangle.

FCOLOR = fill color.

Graphics keywords: CHARSIZE,COLOR,LINESTYLE,NOCLIP,
T3D,THICK,Z,LINE_FILL,ORIENTATION,DEVICE

MODIFICATION HISTORY:

D. L. Windt, Bell Laboratories, September 1990.

Added device keyword, January 1992.

windt@bell-labs.com

(See `./rectangle.pro`)

REC_IMAGE

[\[Previous Routine\]](#) [\[Next Routine\]](#) [\[List of Routines\]](#)

NAME :

REC_IMAGE

PURPOSE :

Extract a rectangular portion of a previously displayed image.

CALLING SEQUENCE :

SMALL_IMAGE=REC_IMAGE(BIG_IMAGE)

INPUTS :

BIG_IMAGE = array containing original image

OUTPUTS :

SMALL_IMAGE = portion of big_image

PROCEDURE :

RECROI is used to digitize a portion of the image.

MODIFICATION HISTORY :

David L. Windt, Bell Labs, Feb. 1992.
windt@bell-labs.com

(See `./rec_image.pro`)

ROI_WIDTH

[\[Previous Routine\]](#) [\[Next Routine\]](#) [\[List of Routines\]](#)

NAME :

ROI_WIDTH

PURPOSE :

Measure the width of a region of curve that has been previously

plotted. The region is defined to be within ymin and ymax of a digitized region of the curve.

CALLING SEQUENCE:

Result=ROI_WIDTH(XAXIS,YAXIS)

INPUTS:

XAXIS - the x axis variable which has been plotted.

YAXIS - the y axis variable which has been plotted.

KEYWORD PARAMETERS:

YMIN - minimum value of digitized region of interest.

YMAX - maximum value of digitized region of interest.

NOHIGHLIGHT - set to inhibit highlighting the region of interest.

H_COLOR - the color index for highlighting the region of interest. Default is 7 (Yellow).

H_THICK - the thickness for highlighting the region of interest.

NOLABEL - set to inhibit labelling fwhm.

MANUAL - set to disable automatic location selection for labels.

L_HEADER - string specifying the label header. Default-''.

L_COLOR - color index for the label.

L_FORMAT - format string for label (eg. '(f4.2)').

UNITS - string specifying units along x axis.

CHARSIZE - size of label text.

PSYM - psym

OUTPUTS:

Result - the full-width-half-max of the region of interest of the curve, in x-axis data units.

OPTIONAL OUTPUT PARAMETERS:

ROI - the subscripts of the digitized region of interest.

WIDTH_ROI - the subscripts of the region between the ymin and ymax points.

LINE_PTS - a 4-element array containing the coordinates of the line drawn on the plot: [x0,x1,y0,y1]

LABEL - the label for the plot.

L_POS - a two element array containing the x,y coordinates of the label, in data coords.

SIDE EFFECTS:

TEK_COLOR is used to load in the tektronix colors.
The region of interest of the curve is highlighted.
The width is labelled.

RESTRICTIONS:

The data must be plotted prior to calling ROI_WIDTH

PROCEDURE:

The user is asked to digitize the endpoints of the region of interest with the mouse. The region is highlighted, and the width is labelled.

MODIFICATION HISTORY:

D. L. Windt, Bell Laboratories, October 1990.
windt@bell-labs.com

(See ./roi_width.pro)

ROTATION

[\[Previous Routine\]](#) [\[Next Routine\]](#) [\[List of Routines\]](#)

NAME:

ROTATION

PURPOSE:

Rotate two vectors by a specified amount.

CALLING SEQUENCE:

ROTATION,X,Y,DEG,NX,NY

INPUTS:

X,Y :original data point pairs

DEG :degrees to rotate.

OUTPUTS:

Nx, Ny = rotated point pairs.

MODIFICATION HISTORY:

Jeff Bennett, U of Colorado

(See ./rotation.pro)

ROT_MAT

[\[Previous Routine\]](#) [\[Next Routine\]](#) [\[List of Routines\]](#)

NAME:

ROT_MAT

PURPOSE:

Return a 2D rotation matrix.

CALLING SEQUENCE:

Result = ROT_MAT(Angle)

INPUTS:

Angle - Rotation angle in degrees.

OUTPUTS:

Result = [[cos(Angle*!dtr), -sin(Angle*!dtr)],
 [sin(Angle*!dtr), cos(Angle*!dtr)]]

EXAMPLE:

To rotate a vector X by an angle Theta:

X=[0.1,0.9]
X_=ROT_MAT(Theta)##X

MODIFICATION HISTORY:

David L. Windt, December 2003

windt@astro.columbia.edu

(See ./rot_mat.pro)

RXO_COLOR

[\[Previous Routine\]](#) [\[Next Routine\]](#) [\[List of Routines\]](#)

NAME:

RXO_COLOR

PURPOSE: Load a color table for the first 32 colors, exactly as the TEK_COLOR procedure does. However this procedure uses different colors for the top 16 color indices, presenting a broader choice of colors for plotting.

CATEGORY:

Graphics.

CALLING SEQUENCE:

RXO_COLOR [[, Start_index] , Ncolors]

INPUTS:

Start_index = optional starting index of palette. If omitted, use 0.

Ncolors = Number of colors to load. 32 is the max and the default.

KEYWORD PARAMETERS:

None.

OUTPUTS:

No explicit outputs.

COMMON BLOCKS:

Colors.

SIDE EFFECTS:

Ncolors color indices, starting at Start_index are loaded with the Tektronix 4115 default color map.

RESTRICTIONS:

None.

PROCEDURE:

Just copy the colors. This table is useful for the display of graphics in that the colors are distinctive.

Basic colors are: 0 - black, 1 - white, 2 - red, 3 - green, 4 - blue, 5 - cyan, 6 - magenta, 7 - yellow, 8 - orange, etc.

MODIFICATION HISTORY:

DMS, Jan, 1989.

DMS, June, 1992. Added colors common.

DMS, Apr, 1993, Added start_index and ncolors.

D.Windt, May 2013: same as TEK_COLOR, but top 16 colors are different.

(See ./rxo_color.pro)

SECONDS2CLOCK

[\[Previous Routine\]](#) [\[Next Routine\]](#) [\[List of Routines\]](#)

NAME:

SECONDS2CLOCK

PURPOSE:

Convert a time value from seconds to a string of the form
"DAYS:HOURS:MINUTES:SECONDS.FRACTIONAL_SECONDS"

CALLING SEQUENCE:

Result = SECONDS2CLOCK(Time)

INPUTS:

Time - Time value in seconds.

KEYWORDS:

SECONDS_FORMAT = IDL format code used to display value of
SECONDS.FRACTIONAL_SECONDS. Default is '(F4.1)'.

OUTPUTS:

This function returns a string constant.

EXAMPLE:

X=SECONDS2CLOCK(106272.)

MODIFICATION HISTORY:

David L. Windt, Columbia University, 10-Jul-2003
windt@astro.columbia.edu

(See ./seconds2clock.pro)

SHIFT_PLOT

[\[Previous Routine\]](#) [\[Next Routine\]](#) [\[List of Routines\]](#)

NAME:

SHIFT_PLOT

PURPOSE:

Interactively slide a previously plotted array using the mouse.

CALLING SEQUENCE:

SHIFT_PLOT,X[,Y,SHIFT=SHIFT]

INPUTS:

X,Y - array variables

KEYWORD PARAMETERS:

Same as for oplot

OPTIONAL OUTPUT PARAMETERS:

SHIFT - the shift along the x-axis

PROCEDURE:

MENUS is used to get input. The previously plotted array is first erased, then oplot'ed, with the incremental shift.

MODIFICATION HISTORY:

David L. Windt, Bell Labs, February, 1990
windt@bell-labs.com

(See ./shift_plot.pro)

SHOW_CT

[\[Previous Routine\]](#) [\[Next Routine\]](#) [\[List of Routines\]](#)

NAME:

SHOW_CT

PURPOSE:

Make a window and show the first 32 colors in the current color table.

CALLING SEQUENCE:

SHOW_CT

MODIFICATION HISTORY:

David L. Windt, Bell Labs, November 1989
windt@bell-labs.com

DLW, November, 1997 - Removed default window position values, so that the window is now visible on any

size display.

(See `./show_ct.pro`)

SINC

[\[Previous Routine\]](#) [\[Next Routine\]](#) [\[List of Routines\]](#)

NAME:

SINC

PURPOSE:

Function to return the value of the SINC function,
i.e., $\sin(x)/x$.

CALLING SEQUENCE:

Result = SINC(X)

INPUTS:

X - Input value. Scalar or array.

OUTPUTS:

Result - Value of $\sin(X)/X$.

PROCEDURE:

Straightforward; except Result is explicitly set to
one when $X=0$.

MODIFICATION HISTORY:

David L. Windt, Bell Laboratories, May 1997

March 1999:

Returned X values are no longer changed when $X=1$.

DLW (thanks to Paul Woodford.)

windt@bell-labs.com

(See `./sinc.pro`)

SINCSQUARE_FIT

[\[Previous Routine\]](#) [\[Next Routine\]](#) [\[List of Routines\]](#)

NAME:

SINCSQUARE_FIT

PURPOSE:

Fit $y=f(x)$ where:
 $F(x) = a_0 * (\sin(a_1 * (x - a_2)) / (a_1 * (x - a_2)))^2 + a_3$
Estimate the parameters a_0, a_1, a_2, a_3 and then call curvefit.

CALLING SEQUENCE:

YFIT = SINC_FIT(X,Y,A)

INPUTS:

X - independent variable, must be a vector.

Y - dependent variable, must have the same number of points as x.

OUTPUTS:

YFIT = fitted function.

OPTIONAL OUTPUT PARAMETERS:

A = Fit coefficients. A four element vector as described above.

MODIFICATION HISTORY:

Adapted from GAUSSFIT

D. L. Windt, Bell Laboratories, March, 1990
windt@bell-labs.com

(See ./sincsquare_fit.pro)

SMALL_WINDOW

[\[Previous Routine\]](#) [\[Next Routine\]](#) [\[List of Routines\]](#)

NAME:

SMALL_WINDOW

PURPOSE:

Make a 500x400 graphics window.

CATEGORY:

Stupid little convenience routines.

CALLING SEQUENCE:

SMALL_WINDOW [,WINDOW_NUM]

OPTIONAL INPUT PARAMETERS:

WINDOW_NUM - the window number.

KEYWORD PARAMETERS:

LAPTOP - set this to make a nice window (400x300) for a
small-screened laptop

MODIFICATION HISTORY:

David L. Windt, Bell Laboratories, March 1990.
windt@bell-labs.com

(See ./small_window.pro)

SP

[\[Previous Routine\]](#) [\[Next Routine\]](#) [\[List of Routines\]](#)

NAME:

SP

PURPOSE:

Execute SET_PLOT, and optionally some handy settings.

CALLING SEQUENCE:

SP[,DEVICE,N_PLOTS]

OPTIONAL INPUTS:

DEVICE = 0 for set_plot,'PS'
1 for set_plot,'HP'
2 for set_plot,'PCL'
3 for set_plot,'X'
4 for set_plot,'MAC'
5 for set_plot,'WIN'
6 for set_plot,'SUN'
7 for set_plot,'TEK'


```
8 for set_plot,'CGM'
```

if DEVICE is not set, the graphics device will be set to the platform-dependent default.

```
N_PLOTS = 1 for !p.multi=0
          2 for !p.multi=[0,1,2]
          3 for !p.multi=[0,1,3]
          4 for !p.multi=[0,2,2]
```

KEYWORD PARAMETERS:

SMALL - Set to make a small plot.

LANDSCAPE - Set for landscape mode when device=0,1, or 2.

FULL_PAGE - Set for full page plotting when device=0, 1, or 2. Only has an effect when in portrait mode (landscape=0) for PS and PCL devices. FULL_PAGE is set automatically if N_PLOTS is greater than 1.

HARDWARE - Set for hardware fonts.

FILE - Name of output file.

ISOTROPIC - Set for isotropic (equal x and y) scaling.

COLOR - Set to enable color for PS and PCL devices.

PLOTSIZE - A four-element array specifying the [XSIZE,YSIZE,XOFFSET,YOFFSET] keywords (in INCHES) to the DEVICE procedure. If PLOTSIZE is set, then the SMALL and FULL_PAGE keywords are ignored. If PLOTSIZE is not set, then default values are used for these parameters that make decent-looking plots on 8-1/2 x 11" paper.

MODIFICATION HISTORY:

David L. Windt, Bell Labs November 1989

Added DEVICE=4, November 1990.

Added ISOTROPIC keyword, August 1991.

Added COLOR keyword, Sept 1991.

Added pcl support, completely changed device<->number mapping, and changed functionality of small/full_page/landscape/size keywords, May 1997.

DLW, September 1997: On Unix platforms, if DEVICE is not set, the graphics device is set to 'X' if the IDL_DEVICE environment variable is not defined.

DLW, January 1998: Added support for CGM graphics; this routine will do nothing more than issue the SET_PLOT,'CGM' command, but is included for compatability with the PLOT_PRINT routine. When using the CGM device, you will likely want to

set the color table entry for !p.color to black; otherwise you'll get a white plot on a white background.

Also, fixed bug that caused graphics output to anything but PS to fail! (Doh!)

DLW, November 2002: Keywords to the DEVICE procedure are no longer abbreviated, for compatibility with IDL 5.5. Hey, kids, here's a little IDL programming tip: even though IDL will let you, never, ever abbreviate any keywords, at least if you want code that lasts!

windt@astro.columbia.edu

(See ./sp.pro)

SQUARE_PLOT

[\[Previous Routine\]](#) [\[Next Routine\]](#) [\[List of Routines\]](#)

NAME:

SQUARE_PLOT

PURPOSE:

Define !p.region so plots come out with aspect ratio of 1.

CALLING SEQUENCE:

SQUARE_PLOT

KEYWORD PARAMETERS:

CENTER - set to center plot in window.

MODIFICATION HISTORY:

David L. Windt, Bell Laboratories, December 1991.
windt@bell-labs.com

(See ./square_plot.pro)

SYM

[\[Previous Routine\]](#) [\[Next Routine\]](#) [\[List of Routines\]](#)

NAME:

SYM

PURPOSE:

This function provides a convenient way to utilize the USERSYM procedure to create an extended choice of plotting symbols, and is intended to be used directly with the PSYM keyword to PLOT, OPLOT, etc.

CALLING SEQUENCE:

Result=SYM(NUMBER)

INPUTS:

NUMBER - symbol number

0 : dot if /FORCE_DOT is set; otherwise this is the same as setting PSYM=0.
1 : filled circle
2 : filled upward triangle
3 : filled downward triangle
4 : filled diamond
5 : filled square
6 : open circle
7 : open upward triangle
8 : open downward triangle
9 : open diamond
10 : open square
11 : plus
12 : X
13 : star
14 : filled rightfacing triangle
15 : filled leftfacing triangle
16 : open rightfacing triangle
17 : open leftfacing triangle

KEYWORD PARAMETERS:

FORCE_DOT - When setting NUMBER=0, set this keyword to use the dot plotting symbol. Otherwise, setting NUMBER=0 is the same as setting PSYM=0.

OUTPUTS:

The function returns the symbol number to be used with the PSYM keyword in the PLOT, OPLOT, etc. commands

SIDE EFFECTS:

The USERSYM procedure is used to create a symbol definition.

EXAMPLE:

To produce a plot using open circles as plotting symbols:

PLOT,X,Y,PSYM=SYM(6)

MODIFICATION HISTORY:

Martin Schultz, Harvard University, 22 Aug 1997: VERSION 1.00

D. Windt, windt@astro.columbia.edu

January 2004: Now possible to use negative values of NUMBER, to make plots with lines connecting the symbols. Added the FORCE_DOT keyword.

(See ./sym.pro)

SYMBOLS

[\[Previous Routine\]](#) [\[Next Routine\]](#) [\[List of Routines\]](#)

NAME:

SYMBOLS

PURPOSE:

Create custom plotting symbols

CALLING SEQUENCE:

SYMBOLS, SYMBOL_NUMBER, SCALE

INPUTS:

SYMBOL_NUMBER:

- 1 = open circle
- 2 = filled circle
- 3 = arrow pointing right
- 4 = arrow pointing left
- 5 = arrow pointing up
- 6 = arrow pointing down
- 7 = arrow pointing up and left (45 degrees)
- 8 = arrow pointing down and left
- 9 = arrow pointing down and right.
- 10 = arrow pointing up and right.
- 11 through 18 are bold versions of 3 through 10
- 19 = horizontal line
- 20 = box
- 21 = diamond
- 22 = triangle
- 30 = filled box
- 31 = filled diamond
- 32 = filled triangle

SCALE - size of symbols.

KEYWORD PARAMETERS:

COLOR - color of symbols

SIDE EFFECTS:

The desired symbol is stored in the user buffer and will be plotted if !P.PSYM = 8.

MODIFICATION HISTORY:

Jeff Bennett, U of Colorado, 198?

(See ./symbols.pro)

TEXT_WIDTH

[\[Previous Routine\]](#) [\[Next Routine\]](#) [\[List of Routines\]](#)

NAME:

TEXT_WIDTH

PURPOSE:

Function to determine the actual displayed width (approximately!) of a string of text, in normalized character units, accounting for the fact that non-equal spacing is used when such a string is displayed on a plot using XYOUTS.

This function is used, for example, by the PLOT_TEXT and LEGEND procedures to ~correctly draw a box around displayed text.

CALLING SEQUENCE:

Result=TEXT_WIDTH(TEXT_STRING)

INPUTS:

TEXT_STRING - a string of text

KEYWORD PARAMETERS:

FONT - Set to an integer from 3 to 20 (corresponding to the Hershey vector font sets,) referring to the font that will be used to display the text. (Any font commands embedded in the text string are ignored.)

RESTRICTIONS:

This function hardly works perfectly, especially when the text string contains a mix of fonts; superscripts and subscripts

will really mess things up as well. But it comes close in many instances.

PROCEDURE:

A table of normalized character widths (determined using the !3 font) is used to determine the width of the text string. In order to account for the use of IDL font manipulation commands, the '!' symbol and the character immediately following it are not counted, except for the case of two consecutive '!' symbols.

EXAMPLE:

Determine the width of a text string:

```
width=TEXT_WIDTH('!3This is some displayed text',font=3)
```

MODIFICATION HISTORY:

David L. Windt, Bell Labs, October 1997
windt@bell-labs.com

(See [./text_width.pro](#))

TRACK_PLOT

[\[Previous Routine\]](#) [\[Next Routine\]](#) [\[List of Routines\]](#)

NAME:

TRACK_PLOT

PURPOSE:

A procedure to plot X vs Y in a widget, track the cursor position, and interactively display the Y(X) value.

CALLING SEQUENCE:

```
TRACK_PLOT,X,Y
```

INPUTS:

X, Y - 1-D arrays

KEYWORD PARAMETERS:

WXSIZE - Draw widget X size, in pixels. (Default=640)

WYSIZE - Draw widget Y size, in pixels. (Default=480)

CROSSHAIR - Set this to display a crosshair at the current Y(X) value.

plus all valid IDL PLOT keywords.

RESTRICTIONS:

Requires widgets. Requires use of the VALUE_TO_INDEX function in the windt library.

EXAMPLE:

Create some X,Y data and plot it using TRACK_PLOT:

```
X=VECTOR(0.,100.,256)
Y=SIN(X/5.)*EXP(-X/20.)
TRACK_PLOT,X,Y
```

MODIFICATION HISTORY:

D. L. Windt, Bell Laboratories, August 1997
windt@bell-labs.com

March, 1998 - Added crosshair display option and CROSSHAIR keyword.

(See ./track_plot.pro)

TWOSCOMPLEMENT

[\[Previous Routine\]](#) [\[Next Routine\]](#) [\[List of Routines\]](#)

NAME:

TWOSCOMPLEMENT

PURPOSE:

Taking the Two's Complement of an integer

CATEGORY:

Math, Hardware, CAMAC

CALLING SEQUENCE:

```
twoscomp = TwosComplement( int )
```

INPUT:

int - raw encoder value (8, 16 or 32 bit integer)

OUTPUT:

twoscomp - Two's complement of input.

KEYWORDS:

Optional Input:

NBITS - # of bits; throw away this bit if there is a carry after adding 1 to the complement. Default is determined by data type.

IfNeg - Only return the Two's Complement if value negative
ALGORITHM:

Taking the Two's Complement of a k-Digit Bitstring:

1. Complement the bitstring; i.e., change all 0s to 1s and all 1s to 0s; retain all leading 0s in your result.
2. Add 1 to this binary number (if there is a carry of 1 into the (k+1)st position, throw it away so that the result is still k-digits).
3. The result from (2) is the two's complement of the bitstring

COMMENT:

Works in many cases, but sign bit may get extended in some applications

MODIFICATION HISTORY:

5-Jun-00 WMD Added Nbits & IfNeg Keywords

(See [./twoscomplement.pro](#))

VALUE_TO_INDEX

[\[Previous Routine\]](#) [\[Next Routine\]](#) [\[List of Routines\]](#)

NAME:

VALUE_TO_INDEX

PURPOSE:

Given a (1D) ARRAY and a scalar VALUE, determine the array INDEX corresponding to the element of the array that is closest in magnitude to the specified value.

CALLING SEQUENCE:

Index=VALUE_TO_INDEX(ARRAY, VALUE)

INPUTS:

ARRAY = 1D array of values

VALUE = scalar value

EXAMPLE:

ARRAY=findgen(100)/99.*5 ; create an array of 100 pts from 0 to 5.

Index=VALUE_TO_INDEX(ARRAY,3.125) ; find the element of ARRAY whose value
; is closest to 3.125.

In this case, Index=62 (i.e., ARRAY(62)=3.13131)

MODIFICATION HISTORY:

David L. Windt, Bell Labs, March 1997.

May 1998 - Realized that this function is hardly necessary, as one can just make wise use of the min function and the !c system variable. Duh!

windt@bell-labs.com

(See ./value_to_index.pro)

VECTOR

[\[Previous Routine\]](#) [\[Next Routine\]](#) [\[List of Routines\]](#)

NAME:

VECTOR

PURPOSE:

Make a vector of PTS points, with values ranging from MIN to MAX.

CALLING SEQUENCE:

Result = VECTOR(MIN,MAX,PTS)

INPUTS:

MIN - Starting value for vector.

MAX - Ending value for vector.

PTS - Number of points.

KEYWORDS:

LOGARITHMIC - set for logarithmic spacing between points.
[MIN and MAX must be positive, i.e., gt 0]

OUTPUTS:

This function returns a vector of PTS points, ranging from MIN to MAX. The returned vector is of the same type as MIN/MAX.

EXAMPLE:

X=VECTOR(5.,100.,1000)

This example returns a 1-D Floating point array X, made up of 1000 points, ranging from 5. to 100.

```
X=VECTOR(5.d,100.d,1000)
```

This example returns a 1-D Double point array X, made up of 1000 points, ranging from 5. to 100.

MODIFICATION HISTORY:

David L. Windt, Bell Labs, June 1993.

March, 1997- modified code so returned vector is same type as MAX. added LOGARITHMIC keyword.

May, 1998 - corrected a bug which occurred when LOGARITHMIC was set and PTS=1.

October, 1998 - corrected a bug which, when LOGARITHMIC was set, had caused the log of MIN and MAX to be returned if these parameters are passed as named variables (rather than constants.)

windt@bell-labs.com

(See **./vector.pro**)

WRITE_MPEG

[\[Previous Routine\]](#) [\[Next Routine\]](#) [\[List of Routines\]](#)

NAME:

WRITE_MPEG

PURPOSE:

Write a sequence of images as an mpeg movie

CATEGORY: utility

CALLING SEQUENCE:

```
WRITE_MPEG, 'movie.mpg', ims
```

INPUTS:

ims: sequence of images as a 3D array with dimensions [sx, sy, nims]

where `sx` = xsize of images
 `sy` = ysize of images
 `nims` = number of images

OPTIONAL INPUTS:

KEYWORD PARAMETERS:

`delaft`: if set delete temporary array after movie was created
 you should actually always do it otherwise you get
 problems with permissions on multiuser machines (since
 /tmp normally has the sticky bit set)
 `rep`: if given means repeat every image 'rep' times
 (as a workaround to modify replay speed)

OUTPUTS: None

OPTIONAL OUTPUTS:

COMMON BLOCKS:

SIDE EFFECTS:

 creates some files in TMPDIR which are only removed when
 the DELAFT keyword is used

RESTRICTIONS:

 depends on the program `mpeg_encode` from University of
 California, Berkeley, which must be installed in `/usr/local/bin`

PROCEDURE:

 writes a parameter file based on the dimensions of the image
 array + the sequence of images in ppm format into a
 temporary directory; finally spawns `mpeg_encode` to build the
 movie

EXAMPLE:

MODIFICATION HISTORY:

 Mon Nov 18 13:13:53 1996, Christian Soeller
 <csoelle@mbcsg1.sghms.ac.uk>

 grabbed original from the net and made slight modifications

(See `./write_mpeg.pro`)

XDISPLAYFILE

[\[Previous Routine\]](#) [\[Next Routine\]](#) [\[List of Routines\]](#)

NAME:

XDISPLAYFILE

PURPOSE:

Display an ASCII text file using widgets and the widget manager.

CATEGORY:

Widgets.

CALLING SEQUENCE:

XDISPLAYFILE, Filename

INPUTS:

Filename: A scalar string that contains the filename of the file to display. The filename can include a path to that file.

KEYWORD PARAMETERS:

BLOCK: Set this keyword to have XMANAGER block when this application is registered. By default the Xmanager keyword NO_BLOCK is set to 1 to provide access to the command line if active command line processing is available.

Note that setting BLOCK for this application will cause all widget applications to block, not only this application. For more information see the NO_BLOCK keyword to XMANAGER.

DONE_BUTTON: the text to use for the Done button. If omitted, the text "Done with <filename>" is used.

EDITABLE: Set this keyword to allow modifications to the text displayed in XDISPLAYFILE. Setting this keyword also adds a "Save" button in addition to the Done button.

FONT: The name of the font to use. If omitted use the default font.

GROUP: The widget ID of the group leader of the widget. If this keyword is specified, the death of the group leader results in the death of XDISPLAYFILE.

HEIGHT: The number of text lines that the widget should display at one time. If this keyword is not specified, 24 lines is the default.

TEXT: A string or string array to be displayed in the widget instead of the contents of a file. This keyword supercedes the FILENAME input parameter.

TITLE: A string to use as the widget title rather than the file name or "XDisplayFile_RXO".

WIDTH: The number of characters wide the widget should be. If this

keyword is not specified, 80 characters is the default.

WTEXT: Output parameter, the id of the text widget. This allows setting text selections and cursor positions programmatically.

OUTPUTS:

No explicit outputs. A file viewing widget is created.

SIDE EFFECTS:

Triggers the XMANAGER if it is not already in use.

RESTRICTIONS:

None.

PROCEDURE:

Open a file and create a widget to display its contents.

MODIFICATION HISTORY:

Written By Steve Richards, December 1990

Graceful error recovery, DMS, Feb, 1992.

12 Jan. 1994 - KDB

If file was empty, program would crash. Fixed.

4 Oct. 1994 MLR Fixed bug if /TEXT was present and /TITLE was not.

2 jan 1997 DMS Added DONE_BUTTON keyword, made Done button align on left, removed padding.

14-Oct-2003 D.L. Windt, The text widget can now be resized interactively by the user.

(See ./xdisplayfile_rxo.pro)

XWD2GIF

[\[Previous Routine\]](#) [\[List of Routines\]](#)

NAME:

XWD2GIF

PURPOSE:

Convert an XWD image file to a GIF image file.

CALLING SEQUENCE:

XWD2GIF[,FILE=FILE]

KEYWORDS:

FILE - The name of the XWD file. If the XWD file is called file.xwd, then the newly created gif file will be called file.gif.

PROCEDURE:

The procedure is just a simple interface to the READ_XWD and
WRITE_GIF routines.

MODIFICATION HISTORY:

David L. Windt, Bell Labs, May 1998.

windt@bell-labs.com

(See ./xwd2gif.pro)
