THE COBE SOFTWARE CATALOG

December 13, 1994

General Sciences Corporation
Applied Research Corporation
Universities Space Research Association

DOCUMENT SUMMARY

COBE Guest Investigator Handbook: Sent to guest investigators (GIs) before their arrival, this manual contains information on the GI program, the computer equipment earmarked for guests and its use, the Initial Products, and available software.

COBE Guest Investigator Support Software User's Guide: Intended for GIs and other UIMAGE users, this guide provides instructions for using the COBE Guest Investigator Support (CGIS) software, including pointers to on-line documentation, an analysis software tutorial, a complete guide to UIMAGE, examples and instructions for representative UIDL tasks, and directions for running other analysis packages at the Cosmology Data Analysis Center (CDAC).

COBE Analysis Cookbook: Written for Initial Product guest investigators, the DIRBE section of this cookbook expands upon the general instructions of the CGIS User's Guide with specific hints for handling the large DIRBE data set and some typical analysis chores.

Proposer Information Package: Distributed in the packet of materials sent to prospective guest investigators (GIs), this document provides preliminary details on the COBE Initial Products and the planned computing hardware configuration and software configuration for the GI program.

CDAC Orientation Handbook: Intended primarily for new employees at the Cosmology Data Analysis Center or others with similar privileges on the COBECL cluster, this handbook describes the CDAC physical and computing facilities, as well as how to get around in the COBECL environment. Project software data processing and its associated tools are also discussed.

TABLE OF CONTENTS

UIDL ROUTINE PROLOGUES

* indicates routine that uses special software only available on COBE cluster

FITMU / Analysis Tools
FITS2X / General Data I/O
FITSBYTE / General Data I/O
FITSHDR / COBE Data Input
FITSLIST / General Data I/O
FITSPEC / Analysis Tools
FITSTAPE / General Data I/O
FITS_INFO / General Data I/O
FLUX2MAG / Analysis Tools
FULLMAP / Analysis Tools

GAPFILL / Image and Plot
GETNAME / General Data I/O
GETOPT / Utilities
GETPRO / Utilities
GETTOK / Utilities
GET_COORDS / Utilities
GET_DATE / Utilities
GET_RESOLUTION / COBE Data Input *
GFIT / Analysis Tools
GRABFACE / Utilities

HEADFITS / General Data I/O
HELIO / Analysis Tools & Utilities
HOST_TO_IEEE / General Data I/O

IEEE_TO_HOST / General Data I/O
IMSTAT / Analysis Tools
IPINFO / COBE Data Input

LABELIT / Image and Plot
LABEL_DATE / Image and Plot
LATCIRC / Image and Plot
LATCUT / Utilities
LEAPCHK / Utilities

MAG2FLUX / Analysis Tools
MARKMAP / Image and Plot
MERGE / Utilities
MIN_CURVE_SURF / Analysis Tools
MKBGRMOD / Analysis Tools
MKHDR / General Data I/O
MONTHDAYS / Utilities
MOONPOS / Analysis Tools & Utilities
MULTIPOLE / Analysis Tools

NICE_HISTO / Analysis Tools
NYQUIST / Analysis Tools

PIX2DAT / Image and Plot
PIX2XY / Utilities
PIXINFO / Utilities
PLANCK / Analysis Tools
PLTIME / Image and Plot
PRECESS / Utilities
PROJBLD / Image and Plot
PROJGRID / Image and Plot & Utilities
PSIMAGE / Image and Plot
PUT_OWNER_ID / Image and Plot
PXINCIRC / Utilities
PXINPOLY / Utilities
PXINRING / Utilities

QS1_L / Utilities
QS1_R / Utilities

RDFITS_STRUCT / General Data I/O
READCOL / General Data I/O
READFITS / General Data I/O
READFMT / General Data I/O
READ_MAP_FACE / COBE Data Input *
READ_RDF / COBE Data Input *
READ_RMS / COBE Data Input *
READ_SKYMAP / COBE Data Input *
READ_TOD / COBE Data Input *
REMCHAR / Utilities
REPCHR / Utilities
REPLAY / Analysis Tools
REPROJ / Image and Plot
ROUND / Utilities

SCDTT / COBE Data Input
SETUP_PS / Image and Plot
SET_RDF / COBE Data Input
SIGMA / Utilities
SINCE_VERSION / Utilities
SIXPACK / Utilities
SIXUNPACK / Utilities
SKY / Analysis Tools
SKYCUT / Image and Plot
SMOOTHCUBE / Utilities

INSTRUMENT-SPECIFIC UNCONFIGURED SOFTWARE

DIRBE UNCONFIGURED SOFTWARE

- BIWEIGHT_MEAN
- DBC
- DCC
- DEG2XY
- DIRBE_STATS
- FFT_SPLINE
- HLINE
- IMVIM
- KSTWO
- LINEAR_MEAN
- LSQSIG
- ORIG_DATE
- OSLICE
- PLN03
- PLOTCOLOR
- POINT_REMOVER
- PRINT_WIN
- PROJCOOR
- RDPIX2
- RDPIXZ
- RESIS_MEAN
- ROBUST_LINEFIT
- ROBUST_POLY_FIT
- SLICE
- SPEC_INT
- SYSRESP
- TVBOX
- TVELLIPSE
- UNMASK
- UNPACK_WHERE
- VLINE
- WRITE256
- WRITEBMP
- WTD_MEAN

DMR UNCONFIGURED SOFTWARE

- CORRELATION_3_PT_V2
- CROSS_CORRELATION
- CROSS_COVARIANCE
- DAI7_DKS
- DAI7_GENUS
- DAI_BLM_MAIN
- DAIRMS_MAIN

DCOR_MAIN
DSRC4_MAIN
FAST_HZ_MAP
MONTE_CARLO_2PT
MONTE_CARLO_3PT
MONTE_CARLO_HZQ
SIMULATE_HZ_MAP
SMOOTH_MAP_FAST
SMOOTH_SKYMAP
SUBTRACT_MULTIPOLE

FIRAS UNCONFIGURED SOFTWARE

ARRAY_DUPS
CUBE
CONV_DALE
CONV_GMT_SEC
ENG_TEMP_LIST
GETTIM
GLIT_RAT_HISTO
GLITCH_HISTO
GMT_DEFAULT
GRATE2
LINFIT
MTMTIDL
PUT_TIME_STAMP
REDCENT
REDUCE
SAA_CHECK
STR_TO_REAL
XYPIX

GENERAL INFORMATION

COPYING/VIEWING A ROUTINE:

UIDL'S ''GETPRO'' is a simple way to obtain a copy of a UIDL routine or IDL User Library procedure. For example, to put a copy of the IDL Library standard deviation procedure STDEV in the current working directory, type

```
UIDL>  getpro, 'stdev'
```

Users may also use the VMS ''copy'' command to copy a routine to their area. For example, to copy the PIX2XY routine to the [USER.IDL] directory, type

```
$  copy cgis$idl:pix2xy.pro [user.idl]my_pix2xy.pro
```

To simply look at the source code and internal documentation of a program, use the VMS ''type/page'' command. For example,

```
$  type/page cgis$idl:pix2xy.pro
```

Users may execute VMS commands from within IDL/UIDL by preceding the command with a dollar sign ($). For example, to view the contents of a VMS file from within UIDL, type

```
UIDL>  $type/page cgis$idl:pix2xy.pro
```

CHANGING THE IDL PATH:

Users running the unconfigured instrument-specific software can run programs without copying the routines to their own area. This is accomplished by adding the proper directory to the IDL or UIDL path. To do this, type the following commands:

```
UIDL>  oldpath = !path  ;to save the original path to a variable
UIDL>  !path = oldpath + ',disk_name:[directory_name]'
```

Replace the disk and directory names with those given in the table. Now IDL will be able to find the routine when only the routine name is typed.

UIDL ROUTINE PROLOGUES

The following pages contain the routine prologue for each of the UIDL routines found in L3 as of the cover date. The routines are listed in alphabetical order. All of these prologues can be found in UIDL's UHELP Help Tree under the top level given next to the routine name. (Those routines that are categorized under more than one UHELP menu option list the possibilities and separate them with an ampersand (&)). See the *CGIS Software User's Guide* for the UIDL Help Tree categories.

The prologues give detailed information about the routine's functionality, its calling sequence, inputs, outputs, and warnings. Examples of usage are also included.

Because these routines were written by many different individuals, the prologues may differ slightly in content, structure, and/or order of presentation.

NAME/ONE LINE DESCRIPTION OF ROUTINE:

ADDX performs extended word integer addition.

PURPOSE:

Performs extended word integer addition

CALLING SEQUENCE:

sum = addx(addend1, addend2, length)

INPUT:

addend1 -	extended integer (32 * n bits)
addend2 -	extended integer (32 * n bits)
length -	length of extended integer in longwords

OUTPUT:

sum -	extended integer (32 * n bits)

NAME/ONE LINE DESCRIPTION OF ROUTINE:

AREA_AVG returns the pixels and area ave of a polygonal region.

DESCRIPTION:

This procedure determines the pixels contained within a user-defined polygon and calculates the area average for this region. If the pixel list is supplied on the command line, then an average is computed. The polygon can be specified using the cursor on a skycube or reprojection or by specifying the longitude and latitude of the vertices.

CALLING SEQUENCE:

```
avg = area_avg(input,[nat_cube=nat_cube],[weight=weight], $
        [poly_pix=poly_pix],[proj=proj],[coord=coord], $
        [win=win],[w_pos=w_pos],[min=min],[max=max], $
        [face=face_num],[color=color],[sent_val=sent_val], $
        [desc_reg=desc_reg],[/zero])
```

ARGUMENTS (I = input, O = output, [] = optional):

input	I	flt arr	Skycube/face/projection input
[nat_cube]	I	flt arr	Native skycube/face input (unfolded or sixpack)
[weight]	I	int/flt arr	Weighting skycube
[poly_pix]	I/O	int vector	Pixel list
[proj]	I	string	Projection Type: ('A' - Aitoff 'S' - Global Sinusoidal 'M' - Mollweide)
[coord]	I	string	Coordinate System: ('E' - Ecliptic 'G' - Galactic 'Q' - Equatorial)
[win]	I	int	Window # of image
[w_pos]	I	int arr	Image offset within window
[min]	I	flt	Image Scale Minimum
[max]	I	flt	Image Scale Maximum
[face]	I	int	face number (face input only)
[color]	I	int	Region boundary color
[sent_val]	I	int/flt	Sentinel value
[desc_reg]	O	string	Region Description Vector
[/zero]			Suppress zero values in average qualifier
avg	O	flt	Area average vector

WARNINGS:

If displaying an image, an X-windows terminal must be used. It is recommended when accessing DIRBE data to use either a single face or a small (512 by 256) projection. Use of an entire DIRBE sky cube or large projection will slow the routine.

EXAMPLES:

If the user already has a pixel list, 'pix_list', from which they want to extract an average for a skycube, 'incube', they should enter:

```
avg = area_avg(incube,poly_pix=pix_list)
```

In the calculation of the average, each pixel is given equal weight. Pixels with value 0 are included in the average. To exclude these pixels the user should enter:

```
avg = area_avg(incube,poly_pix=pix_list,/zero)
```

In order to include a weighting function, the user should supply a weight skymap of the same dimensions as the input data.

```
avg = area_avg(incube,poly_pix=pix_list,weight=weight)
```

If the skycube contains a sentinel value this can be supplied using then 'sent_val' keyword. Any pixel value less than this is ignored when calculating the average.

```
avg = area_avg(incube,poly_pix=pix_list,sent_val=-32000, $
 weight=weight)
```

If a pixel list is not supplied on the command line, then the user can define a region. When using an X-windows terminal, this region can be defined by marking the vertices with the cursor. Alternatively, the user can define the vertices by their longitudes and latitudes. This is the only option of non-X-windows terminals. If the input array is a reprojection then the original native skycube (unpacked or packed) must be supplied using the 'nat_cube' keyword. If the user desires the pixel lists for these region then they should supply the 'poly_pix' keyword on the command line which will they contain the pixel numbers of each of the regions defined. This will be an integer vector with the different pixel lists for each region separated by a '-1' entry.

For example, if the user has a galactic Aitoff projection, 'repro', of the original 'incube' skycube displayed in X-window 1, and offset from the window origin by 32,32 then the call will be:

```
avg = area_avg(repro,nat_cube=incube,proj='a',coord='g',win=1, $
        w_pos=[32,32],min=0,max=30,poly_pix=pix_list, $
        desc_reg=desc_reg,weight=weight)
```

The 'min' and 'max' keywords are supplied so that the reprojection is properly displayed in the window each time the user is prompted to define a region. The 'desc_reg' keyword is user to define a string vector with as many elements as regions defined and containing a short description of the region supplied by the user.

If the skycube itself is displayed, say in window 0, then the call would be:

```
avg = area_avg(incube,win=0,w_pos=[32,32],min=0,max=30, $
        poly_pix=pix_list,desc_reg=desc_reg,weight=weight)
```

If a single cube face is supplied then the call is:

```
avg = area_avg(cube_face,win=0,w_pos=[32,32],min=0,max=30, $
        face=1,poly_pix=pix_list,desc_reg=desc_reg, $
        weight=face_weight)
```

NAME/ONE LINE DESCRIPTION OF ROUTINE:

AVG calculates the average over one dimension of an array

DESCRIPTION:

IDL function to calculate the average value of an array, or to calculate the average value over one dimension of an array as a function of all the other dimensions.

CALLING SEQUENCE:

RESULT = AVG (array [,dimension])

ARGUMENTS (I = input, O = output, [] = optional):

RESULT	O	flt	Average value of array.
ARRAY	I	Any except string	Input array.
DIMENSION	[I]	int	If specified, dimension to perform average over.

WARNINGS:

1. If DIMENSION is passed, the result is an array with all the dimensions of the input array except for the dimension specified, each element of which is the average of the corresponding vector in the input array.

2. The DIMENSION specified must be valid for the array passed; otherwise the input array is returned as the output array.

EXAMPLE:

If A is an array with dimensions of (3,4,5), then the command
B = AVG(A,1) is equivalent to

```
B = FLTARR(3,5)
FOR J = 0,4 DO BEGIN
    FOR I = 0,2 DO B(I,J) = TOTAL(A(I,*,J)) / 4.0
ENDFOR
```

NAME/ONE-LINE DESCRIPTION:

AVGAREA finds the average over a circle or annulus

DESCRIPTION:

AVGAREA calculates a weighted or unweighted average over a given circle or annulus. A circle can be defined by an input array containing the longitude, latitude, and radius of the circle or X Window users can mark the circle center with a mouse and then enter the radius from the keyboard. An annulus can be defined in the two ways similar to the ways a circle is defined except two radii, the inner and the outer, are entered. The average over the specified area is reported, and the min, max, standard deviation, and pixels in the region can also be returned.

CALLING SEQUENCE:

average =avgarea (data, circle=circle, annulus=annulus, [weights=weights], [facenum=facenum], [in_coords=in_coords], [wpos=wpos], [badval=badval], [win=win], [pixels=pixels], [regmin=regmin], [regmax=regmax], [standev=standev], [noplot=noplot])

Use only ONE of the following in call: circle, annulus

ARGUMENTS: (I=input, O=output, []=optional)

data	I	2-D or 3-D data array
circle	I	May be used one of two ways. If keyword is entered simply as "/circle", the center of the circle will be marked with the mouse and the radius will be entered from the keyboard. The radius must be specified in degrees away from the circle center, which acts as a pole around which the circle is drawn. To enter the coordinates of the center and the radius from the command line, use an array in this form: circle=[center_lon, center_lat, radius(in deg)].
annulus	I	Specified in a manner similar to specifying a circle, except that two radii are entered (an inner then an outer) instead of a single radius. Note: inner radius must be less than outer radius. To enter the annulus info from the command line, use the following form: annulus=[cntr_lon,cntr_lat,in_radius,out_radius].
average	O	Averaged data of the specified region. For 2-D input, one value is returned. For 3-D input, an average over each slice is calculated and returned as an element in the average array.
[weights]	I	Array containing weights. If no weight array is specified, the weights are all set equal to 1. Weight arrays are only 2-D, even if the array is 3-D.
[in_coords]	I	Coordinates that the polygon vertices or circle center are given in.
[facenum]	I	If data array is a face, you may specify the facenum in the call.
[badval]	I	Bad pixel value. These pixels will not be used in the averaging. Default value is 0.
[win]	I	Window number where the display lives. N/A for non-X Window users.
[wpos]	I	2-element array indicating Window POSitioning of the image. Use form wpos=[x,y] where x is the X coordinate of the lower left corner of the image and y is the Y coordinate of the lower left corner of the image. i.e., if the image has a 32 screen pixel border on each side, wpos=[32,32]. Default=[0,0]
[pixels]	O	Pixel list of the pixels in the polygon/circle/annulus.
[regmin]	O	Region (polygon,circle,annulus) minimum.

[regmax]	O	Region (polygon,circle,annulus) maximum.
[standev]	O	Standard deviation over region.
[noplot]	I	Use the /noplot keyword if you do not want the polygon or circle plotted on the image. Non X window users must use this option.

EXAMPLES:

To average a circle on a skycube called DATA using a weight array called WTS and marking the center with the mouse, the call would look like this:

```
UIDL>   circ_avg = avgarea (data, /circle, weights=wts)
```

To do the same as above except specify the coordinates of the center and the radius size in the call, you would type:

```
UIDL>  circ_avg = avgarea (data, circle=[0,0,5], $
UIDL>  in_coords='g', weights=wts)
```

NAME/ONE LINE DESCRIPTION OF ROUTINE:
BAR draws a color bar on the display.

CALLING SEQUENCE:
BAR [,MIN = Min] [,MAX = Max] [,TYPE = Type] [,TITLE = Title] [,THICK=Thick] [,/REV]

ARGUMENTS: (I = input, O = output, [] = optional)

Min	[I]	byt	Minimum color number in bar; default is 0
Max	[I]	byt	Maximum color number in bar; default is 255
Type	[I]	chr	Positioning indicator. TYPE='XT': X bar and label at top (default). TYPE='XB': X bar and label at bottom. TYPE='YR': Y bar and label at right. TYPE='YL': Y bar and label at left.
Title	[I]	chr	Bar title.
Rev	[I]	key	If specified, bar is reversed.

WARNINGS:

1. Works only with devices that allow TV operations.

2. Only values between 0 and 255 are allowed for Min and Max; Min must be less than Max if both are specified.

EXAMPLES:

1. Put a color bar from right to left at top of picture.
   BAR
2. Put a reversed color bar of width 20 pixels of colors 0 through 150 at top of picture.
   BAR,/rev, max = 150
3. Put a reversed color bar at bottom of picture with title 'REVERSED'
   BAR, /rev, type = 'xb', title='REVERSED'
4. Put a color bar to left of picture 30 pixels wide with title 'STRAIGHT'
   BAR, type = 'yl', title='STRAIGHT', thick=30
5. Put a color bar to right of picture showing only colors 50 through 200
   BAR, type = 'yr', min = 50, max = 200

NAME/ONE LINE DESCRIPTION OF ROUTINE:

BATCHCUT extracts skycuts defined by a lon/lat array.

DESCRIPTION:

This procedure extracts and stores skycuts defined by a user-supplied array containing the longitude and latitude of the endpoints. Both great-circle and constant latitude cuts can be specified as well as short (< 180 degrees) and long (> 180) arcs. The input array can be either a native skycube (unfolded or sixpacked format), a single face, or a reprojection.

CALLING SEQUENCE:

batchcut, input, coord=coord, [proj=proj], ll_in=ll_in, [lat_flag=lat_flag], [arc_flag=arc_flag], [face=face_num], [n_pnts=n_pnts], [cs_plot=cs_plot], [cs_titl=cs_titl]

ARGUMENTS (I = input, O = output, [] = optional):

input	I	flt arr	skycube/face/projection input
coord	I	string	Coordinate System: 'E' - Ecliptic 'G' - Galactic 'Q' - Equatorial
[proj]	I	string	Projection Type: 'A' - Aitoff 'S' - Global Sinusoidal 'M' - Mollweide
ll_in	I	flt arr (2,2*n)	Lon/lat input array
[lat_flag]	I	int arr (n elem)	Latitude cut flag
[arc_flag]	I	int arr (n elem)	Arc type flag
[face]	I	int	face number (face inpt only)
[n_pnts]	O	int arr	# points in each plot
[cs_plot]	O	flt arr	storage array for plots
[cs-titl]	O	str arr	description array for plots

WARNINGS:

Care be taken not to specify points diametrically opposed as the great circle is then not uniquely defined.

EXAMPLES:

The skycut specification array, "ll_in", is a 2 by 2*n array, where n is the number of cuts. The first column contains the longitudes in degrees (0-360), the second contains the latitudes (-90 - 90).

To specify three cuts along the longitude lines -20, 0, 20 from -25 to 25 latitude, enter:

```
IDL> ll_in = FLTARR(2,6)
IDL> READ, ll_in
: -20 -25 -20 25  0 -25 0 25  20 -25 20 25 <CR>
```

To extract and store these cuts from a input skycube, 'sky_input', (in either unfolded or sixpacked format), enter at the IDL prompt:

```
batchcut,sky_input,ll_in=ll_in,coord='g', $
n_pnts=npnts,cs_plot=plts,cs_titl=titl
```

The 'coord' keyword tells the procedure that these endpoints are in galactic coordinates. The retrieved data is stored in the array specified by the 'cs_plot' keyword. This data array contains, for each cut, the x and y "vectors" displayed in the plot (the (*,0,*) and (*,1,*) elements, respectively), plus the longitude and latitude values of the arc (the (*,2,*) and (*,3,*) elements). Upon exiting the program, SKYCUT truncates the output arrays to minimize the number of unfilled elements. The actual number of points for a particular cut is stored in an integer vector specified by the 'n_plts' keyword. In addition, the description of the cut, containing the coordinates of the endpoints is stored in the string array specified by the 'cs_titl' keyword. All three storage keywords must be provided on the command line otherwise SKYCUT will not allow plots to be saved for later access.

If the input array is a single face then the face number must be specified through the 'face' keyword. If the input array is a reprojection, then the 'proj' keyword must be specified and the 'coord' keyword must correspond to the coordinate system of the projection. Cuts from reprojections are designated by the string 'PR' in the title, all others with a 'SC'.

"Long" Arcs

By default great circle skycuts use the smaller of the two great circles defined by the endpoints. To extract the larger of the two, the user passes an 'n' element integer array via the 'arc_flag' keyword with a zero specifying the default "short" arc and a '1', the "long" (> 180 deg) arc.

For example, if the user desires a cut along the 0 longitude excluding points within 25 degrees of the galactic plane, as well as the two galactic cuts at -20 and 20 longitude, they would enter:

```
batchcut,sky_input,ll_in=ll_in,coord='g',arc_flag=[0,1,0], $
n_pnts=npnts,cs_plot=plts,cs_titl=titl
```

Latitude Cuts

To specify a latitude cut, the user supplies an 'n' element integer array passed through the 'lat_flag' keyword where '0' specifies the default great circle arc and '1' a latitude cut. To specify a cut along the ecliptic latitude 15 from 90 longitude to 270 longitude in addition to a regular great circle cut (specifed as the third and fourth row of 'll'), enter:

```
IDL> ll = [[90.,15],[270,15],[10,-45],[340,36]]

IDL> batchcut,sky_input,ll_in=ll,coord='e',lat_flag=[1,0], $
        n_pnts=npnts,cs_plot=plts,cs_titl=titl
```

The latitude arc goes from 90 longitude to 270. To specify the arc from 270 longitude to 90 simply exchange the numbers in 'll':

```
IDL> ll = [[270.,15],[90,15],[10,-45],[340,36]]
```

NAME/ONE-LINE DESCRIPTION:

BITRANGE takes an arbitrary range of bits from an array for display

DESCRIPTION:

The routine extracts a user-selected range of up to 8 bits from an assumed 16-bit dynamic range, and creates an output array scaled to a single byte (0-255). With a suitable choice of bits (e.g. chopping out some high-order bits) this operation enhances low-level map features by causing the grey scale or color table to wrap around repeatedly, creating a "color contour" effect.

The calling sequence allows the user to specify the bad pixel value (all such pixels will be set to dark) the range of bits, and the range of data to operate on (by default, the whole good data range). If the input data is integer and has an intrinsic dynamic range smaller than 16 bits, the /NOSTRETCH qualifier will cause the routine to extract bits from the raw data rather than first scaling it to a 16-bit range. Otherwise, the rescaling is done first.

All real numeric data types are accepted as input (I*2, I*4, R*4, BYTE), but the output is always in BYTE format.

CALLING SEQUENCE:

ARROUT = BITRANGE (arrin, [bitlo=...], [bithi=...], [datlo=...],[dathi=...], [badval=...], [/NOSTRETCH])

ARGUMENTS (I=input, O=output, []=optional)

ARROUT	O	byte array	Output map or vector, range 0-255
arrin	I	any array	Input map or vector, any real data
[bitlo]	[I]	int	Lo-order bit default=8, min=0
[bithi]	[I]	int	Hi-order bit default=15, max=15
[datlo]	[I]	int or real	Minimum value of data to consider. Default=minimum value of good data
[dathi]	[I]	int or real	Maximum value of data to consider. Default=maximum value of good data
[badval]	[I]	int or real	Bad (i.e. excluded) data value. Def=0
[/NOSTRETCH]	[I]	qualifier	Do not rescale integer data to 16 bits

WARNINGS:

1. No more than 8 bits can be selected. Allowable bit range is 0-15.
2. Complex numbers will give unpredictable results. Strings and structures are forbidden.

EXAMPLES:

1. Consider a floating point map IN with a bad value of -999 and good data of interest ranging from 0 to 1000. We can "quantize" the display to 8 grey levels (3 bits) showing only the brightest features with the call

OUT = BITRANGE (IN, BADVAL=-999, BITLO=13, BITHI=15)

2. If we're only interested in the upper half of the data range but want to emphasize features in the low end of that range, we can expand the dynamic range to eight bits and chop off the highest order bit, i.e.

OUT = BITRANGE (IN, DATLO=500, BITLO=7, BITHI=14)

(Notice that we didn't need to specify BADVAL since the DATLO in this case would exclude it anyway.) If the input data were integer and we were to give the same call with the /NOSTRETCH qualifier we would get a different result, because the raw data only extend up bit 9 (1000 < 2^10). The previous call would then in effect extract the three high-order bits (7-9).

NAME/ONE LINE DESCRIPTION OF ROUTINE:

BOSE_EIN returns a Bose-Einstein spectrum with a chemical potential.

DESCRIPTION:

IDL function to return the spectral radiance of a Bose-Einstein distribution, i.e., a Planck-like curve with a chemical potential term mu. Output is in units of either MJy/steradian (I_nu) or watts/cm^2/steradian (nu_I_nu). Inputs are the equivalent blackbody temperature, the chemical potential parameter mu, and either frequency (in icm or GHz) or wavelength (in microns). The routine also optionally returns the derivative with respect to both temperature (in units of MJy/sr/K or W/cm^2/sr/K) and mu. Note that for mu=0 the output spectrum is identical to the Planck function with the same temperature.

CALLING SEQUENCE:

RESULT = BOSE_EIN (temperature, mu, nu_or_lambda [,derivs] [,UNITS=units], [,/MJY] [,/WCM2])

ARGUMENTS (I = input, O = output, [] = optional):

RESULT	O	flt [arr]	Spectral radiance at each wavelength. Units: W/cm^2/sr/K if /WCM2 specified MJy/sr if /MJY specfied
TEMPERATURE	I	flt	Temperature of blackbody, in K.
MU	I	flt	B-E chemical potential, unitless
NU_OR_LAMBDA	I	flt [arr]	Frequency or wavelength at which to calculate spectrum. Units are as specified with UNITS keyword.
DERIVS	[O]	flt [arr,2]	Derivative of spectrum with respect to temperature [DERIVS(*,0)] and chemical potential [DERIVS(*,1)].
UNITS	[I]	str	'Microns', 'icm', or 'GHz' to identify units of NU_OR_LAMBDA. Only first character is required. If left out, default is 'microns'.
/MJY	I	key	Sets output units to MJy/sr
/WCM2	I	key	Sets output units to W/cm^2/sr

WARNINGS:

1. One of /MJY or /WCM2 MUST be specified.
2. Routine gives incorrect results for T < 1 microKelvin and wavelengths shortward of 1.e-10 microns. (So sue me).
3. We use a dimensionless mu. It is NOT in units of kT.

EXAMPLE:

To produce a 35 K spectrum in MJy/sr at 2, 4, 6, 8, 10 microns with a chemical potential 0.001:

```
wavelength = 2. + 2.*findgen(5)
temp = 35.
mu = 0.001
spectrum = BOSE_EIN (temp, mu, wavelength, units='micron', /mjy)
```

One could also get back the derivatives into the variable "derivs" including it in the call:

```
spectrum = BOSE_EIN (temp, mu, wavelength, derivs, units='m', /mjy)
```

Note that dB/dT = derivs(*,0) and dB/dmu= derivs(*,1).

NAME/ONE LINE DESCRIPTION OF ROUTINE:

BUILDMAP provides user-friendly access to COBE archive maps.

DESCRIPTION:

This IDL function creates an array of rasterized images (sky cubes or faces) using the READ_SKYMAP and PIX2XY facilities. If not supplied on the command line, the user is prompted for the skymap data file name (including the archive directory), the data fields to be extracted (up to 7 if the AV1 or AV2 switches are specified, up to 8 otherwise), and the face number (-1 if entire cube). The output consists of the raster image array, and if desired, the x and y raster coordinate arrays and the pixel and data arrays (from READ_SKYMAP).

For DIRBE skymaps, this facility can handle multiple observations per pixel in a number of ways. They are (1) a simple average, (2) an average for a given approach vector value, (3) the first observation for the pixel, or (4) the last observation for the pixel.

CALLING SEQUENCE:

outmap = buildmap([file=file],[fields=fields],[face=face],
[multobs=multobs],[x_out=x_out],[y_out=y_out], [pixel=pixel],[data=data],/real,/imag,/sixpack)

ARGUMENTS (I = input, O = output, [] = optional):

[outmap]	O	int/flt/dbl/cpx arr	Rasterized output array
[file]	I	str	Skymap file
[fields]	I	str	Skymap data fields
[face]	I	int	Skymap face
[multobs]	I	str	MultIple observations per pixel qualifier
[x_out]	O	int arr	x-coordinate array
[y_out]	O	int arr	y-coordinate array
[pixel]	O	int arr	returned pixel vector
[data]	O	int/flt/dbl/cpx arr	Returned data list
[real]	I	qualifier	takes real part of complex data
[imag]	I	qualifier	takes imag part of complex data
[sixpack]	I	qualifier	sixpack output switch

Multiple Observations per Pixel (MULOPTS) Values

’avg’ - Average multiple values in pixel
’av1’ - Average multiple values in pixel with approach vector 1
’av2’ - Average multiple values in pixel with approach vector 2
’first’ - Use first observation in pixel
’last’ - Use last observation in pixel (DEFAULT)

WARNINGS:

1. Data fields must be all of the same data type. BUILDMAP uses the first data field specified to determine the data type.

2. BUILDMAP uses a lot of memory. The user should free up as much space as possible before running.

3. If any non-recoverable error occurs in BUILDMAP, the outmap return variable will be set to 0.

EXAMPLES:

1. To build an array of DIRBE skymaps for the photometry channels 1, 2 and 7, from a weekly skymap file for approach vector 1 data:

```
outarr = buildmap(file='dirbe_edit:bpw_wf.week_avg_913611607',
fields='photometry(1:2),photometry(7)', face=-1,multobs='av1',pixel=pixout)
```

The output array, 'outarr' will have dimensions (1024,768,3). The pixel list is stored in the vector, 'pixout'.
Note that the FORTRAN convention is used (subscripts starting with 1) when specifying array fields.

2. To build a high frequency FIRAS (real) spectral skymap in sixpack format:

```
spcmap = buildmap(file='firsky:fcs_ccmsp_rh.ed_8934302_9026408',
fields='chan.spec(5:171)',face=-1,/real/sixpack)
```

NAME/ONE-LINE DESCRIPTION:

BUILDTEE takes 1-6 cube faces and places them into an unfolded T

DESCRIPTION:

This function takes up to six square arrays representing skycube faces and places them into the appropriate position on the unfolded full sky "right T", thus enabling the user to build a full sky map out of single faces. The faces can be supplied in any order in the calling sequence, provided that you also supply an optional FACENUMS array telling which face is which. The FACENUMS array is mandatory if fewer than all six faces are being supplied, in order to know which faces they are. If all six faces are supplied and no FACENUMS array is specified, the input arrays are assumed to be in the order 0,1,2,3,4,5.

Any numerical data type can be accommodated; the output will always be a full sky right T with the same data type as the input. The input faces may also be spectral cubes (e.g. FIRAS data). The user may also supply an optional BADVAL bad pixel value that will fill all unoccupied positions of the output array.

If the /SIXPACK qualifier is used then the routine builds a 3 x 2 "sixpack" instead of a right T.

CALLING SEQUENCE:

outmap = BUILDTEE (face0, [face1,...,face5], [facenums=facenums], [badval=badval], [/sixpack])

ARGUMENTS: (I=input, O=output, []=optional)

OUTMAP	O	any type	Output unfolded right-T (all six faces) with same data type as input faces
FACE0	I	any type	Input square array (possibly with a 3rd, spectrum dimension), holding one face of sky cube. Need NOT be face 0.
[FACE1..5]	[I]	any type	Other cube faces, in any order
[FACENUMS]	[I]	int [arr]	Vector containing face numbers of corresponding input cube faces
[BADVAL]	[I]	int	Bad or unoccupied pixel value, default=0
[/SIXPACK]	[I]	keyword	Flag to pack output into sixpack instead of right T

WARNINGS:

It is up to the user to make sure that his input faces are indeed square in the x and y dimensions.

EXAMPLES:

1. If you have all six faces in order f0...f5 and these contain zeros as their flag value, then the call

```
UIDL> SKYMAP = BUILDTEE (F0,F1,F2,F3,F4,F5)
```

will build a complete skymap. You could (perversely) shuffle the order of the input arrays and still build a correct skymap by providing the order in the FACENUMS array, e.g.:

```
UIDL> SKYMAP = BUILDTEE (F1,F4,F2,F5,F3,F0,FACENUMS=[1,4,2,5,3,0])
```

2. Now suppose you had only two faces, called GALCEN for the galactic center (face 4) and ECLPOLE for the North Ecliptic Pole (face 0), your sentinal value is -99. You can build a full skymap with data in the appropriate two faces by calling:

```
UIDL> SKY = BUILDTEE (GALCEN, ECLPOLE, FACENUMS=[4,0], BADVAL=-99)
```

Most of the map would then be filled with the sentinel value.

NAME/ONE LINE DESCRIPTION OF ROUTINE:

CAPTURE copies an image from a workstation window to disk.

DESCRIPTION:

Capture COPIES everything on a specified window, first into an IDL array of the total window size, using 'tvrd'. This is a byte array scaled from 0 to 255. This is then stored on disk in an unformatted file.

CALLING SEQUENCE:

CAPTURE,outfile,[window]

ARGUMENTS (I = input, O = output, [] = optional):

OUTFILE	O	str	Name of unformatted file to store the output image.
WINDOW	[I]	int	The number identifying the IDL tv window to capture. The IDL window '0' is captured by default, unless otherwise specified.

WARNINGS:

1. The output file is created in the current or default directory. The user should make sure there is enough space.
2. This procedure can only be called from a Workstation (X-Windows) session.

EXAMPLE:

1. To capture IDL default display window (0), and store it in a file named 'output.dat', type:

    CAPTURE, 'output.dat'

2. To capture IDL TV display window No. 1, and store it in a file named 'output.dat', type:

    CAPTURE,1,'output.dat'

NAME/ONE LINE DESCRIPTION OF ROUTINE:

CHECK_FITS checks that the FITS header is appropriate for a given array.

PURPOSE:

Given a FITS array IM, and a associated FITS or SDAS header HDR, this procedure will check that

(1) HDR is a string array, and IM is defined and numeric
(2) The NAXISi values in HDR are appropiate to the dimensions of IM
(3) The BITPIX value in HDR is appropiate to the datatype of IM

If HDR contain a DATATYPE keyword (as in SDAS files), then this is also checked against the datatype of of IM
If the UPDATE keyword is present, then FITS header will be modified, if necessary, to force agreement with the image array

CALLING SEQUENCE:

check_FITS, im, hdr, [dimen, idltype, /UPDATE, /NOTYPE, /SDAS]

INPUTS:

IM - FITS or STSDAS array, (e.g. as read by SXREAD or READFITS)
HDR - FITS or STSDAS header (string array) associated with IM

OPTIONAL OUTPUTS:

dimen - vector containing actual array dimensions
idltype- data type of the FITS array as specified in the IDL SIZE function (1 for BYTE, 2 for INTEGER*2, 3 for INTEGER*4, etc.)

OPTIONAL KEYWORD INPUTS:

/NOTYPE - If this keyword is set, then only agreement of the array dimensions with the FITS header are checked, and not the data type.
/UPDATE - If this keyword is set then the BITPIX, NAXIS and DATATYPE FITS keywords will be updated to agree with the array
/SDAS - If this keyword is set then the header is assumed to be from an SDAS (.hhh) file. CHECK_FITS will then ensure that (1) a DATATYPE keyword is included in the header and (2) BITPIX is always written with positive values.
/FITS - If this keyword is present then CHECK_FITS assumes that it is dealing with a FITS header and not an SDAS header, see notes below.

SYSTEM VARIABLE:

If there is a fatal problem with the FITS array or header then !ERR is set to -1. (If the UPDATE keyword was supplied, and the header could be fixed then !ERR = 0.)

PROCEDURE:

Program checks the NAXIS1 and NAXIS2 parameters in the header to see if they match the image array dimensions.

NOTES:

An important distinction between an STSDAS header and a FITS header is that the BITPIX value in an STSDAS is always positive, e.g. BITPIX=32 for REAL*4 data. Users should use either the /SDAS or the /FITS keyword if it is important whether the STSDAS or FITS convention for REAL*4 data is used. Otherwise, CHECK_FITS assumes that if a DATATYPE keyword is present then it is dealing with an STSDAS header.

NAME/ONE LINE DESCRIPTION OF ROUTINE:
CLEANPLOT resets IDL system plotting variables to defaults.

DESCRIPTION:
CLEANPLOT reinitializes NEARLY ALL IDL system plotting variables back to the default values that existed before the user mucked with them. See WARNINGS section for a list of system plotting variables which are not reset.

CALLING SEQUENCE:
CLEANPLOT

ARGUMENTS (I = input, O = output, [] = optional):
There are NO arguments.

WARNINGS:
1. This does NOT reset the plotting device.
2. This does not change any system variables that don't control plotting.
3. THE FOLLOWING PLOTTING SYSTEM VARIABLES ARE NOT RESET:

!P BLOCK:	!p.channel
	!p.clip
	!p.t, !p.t3d
!X BLOCK:	!x.crange
	!x.s
	!x.window
	!x.tickv, !x.tickname
!Y BLOCK:	(see !X block)
!Z BLOCK:	(see !X block)

EXAMPLE:
To clear all existing plotting labels, axis minima and maxima limits, tick labelling, etc:

UIDL> CLEANPLOT

NAME/ONE-LINE DESCRIPTION:

COLORBAR puts a color bar of any size, anywhere with the mouse

DESCRIPTION:

The user uses mouse clicks to specify the size and location of a colorbar in the active window: three mouse clicks with the left button specify the bar's center, left edge, and top. Through the use of keywords, the user can also: specify the range of colors (default [0,255]); reverse the bar so that the bright edge is on the left or at the bottom; and activate the GUI labeling routine to place labels around the bar.

CALLING SEQUENCE:

COLORBAR, [/LABEL], [/REVERSE], [MIN=<minimum>], [MAX=<max>]

ARGUMENTS:

[/LABEL]	switch	Activates WINLABEL routine, which uses a GUI to allow the user to place whatever text (s)he wants on or around the bar. Default = no label.
[/REVERSE]	switch	Causes dark part to be at the right (or at the top for vertically-oriented bars) Default = dark left or bottom.
[MIN]	byte	Value between 0 and 254 to specify darkest color on bar. Default = 0.
[MAX]	byte	Value between 1 and 255 to specify lightest color on bar. Default = 255.

WARNINGS:

1. MIN must be less than MAX; both must be in [0,255].
2. Routine uses the current active window.

EXAMPLES:

The simplest effective invocation is just COLORBAR. For a rather grey-looking bar that uses only the greyscale range [100,200] say COLORBAR,MIN=100,MAX=200.

NAME/ONE LINE DESCRIPTION OF ROUTINE:

CONV_UNIX_VAX converts UNIX IDL data types to VAX IDL data types.

PURPOSE:

To convert Unix IDL data types to VAX IDL data types. Currently assumes the Unix IDL data type is IEEE standard.

CALLING SEQUENCE:

CONV_UNIX_VAX, variable

PARAMETERS:

variable - The data variable to be converted. This may be a scalar or an array. Valid datatypes are integer, longword, floating point, and double precision. The result of the conversion is passed back in the original variable.

RESTRICTIONS:

Only tested for data from IEEE standard Unix machines (e.g. SUN0)

EXAMPLE:

Read a 100 by 100 matrix of floating point numbers from a data file created on a Sun. Then convert the matrix values into VAX format.

```
IDL> openr,1,'vax_float.dat
IDL> data = fltarr(100,100)
IDL> forrd,1,data
IDL> CONV_UNIX_VAX,data
```

NAME/ONE LINE DESCRIPTION OF ROUTINE:

CONV_VAX_UNIX converts VAX IDL data types to UNIX IDL data types.

PURPOSE:

To convert VAX IDL data types to UNIX (Sun,MIPS,etc.) IDL data types. The architecture is obtained from IDL sys.var. !VERSION.ARCH.

CALLING SEQUENCE:

var_unix = conv_vax_unix(var_vax)

INPUT-OUTPUT PARAMETER:

variable

The data variable to be converted. This may be a scalar or an array. All IDL datatypes are valid (including structures). The result of the conversion is returned by the function.

INPUT KEYWORD:

TARGET_ARCH = name (string) of desired target architecture if using this function on a VAX. Otherwise !VERSION.ARCH is used to determine the conversion.

EXAMPLE:

Read a 100 by 100 matrix of floating point numbers from a data file created on a VAX. Then convert the matrix values into Sun format.

```
IDL> openr,1,'vax_float.dat'
IDL> data = fltarr(100,100)
IDL> forrd,1,data
IDL> data = conv_vax_unix( data )
```

NAME/ONE LINE DESCRIPTION OF ROUTINE:

COORCONV is a shell routine for coordinate conversion procedures

DESCRIPTION:

IDL function that acts as shell around the various coordinate conversion routines. It is the program that the user actually calls. Input and output formats are pixel number, decimal hour/latitude, longitude/latitude, and unit vector.

CALLING SEQUENCE:

outcoor = coorconv(in_coor,infmt=in_fmt,outfmt=out_fmt, inco=in_co, outco=out_co)

ARGUMENTS (I = input, O = output, [] = optional):

in_coor	I	[arr]	Input coordinate array
infmt	I	str	Input coordinate format
outfmt	I	str	Output coordinate format
inco	I	str	Input coord. system or resolution
outo	I	str	Output coord. system or resolution
out_coor	O	[arr]	Output coordinate array

where in_coor & out_coor arrays have the following dimensions for n points:

Pixel	- input_arr(n)
Decimal Hr/deg	- input_arr(n,2)
lon/lat	- input_arr(n,2)
Unit vector	- input_arr(n,3)

infmt & outfmt are specified as follows:

Pixel	- 'P'
Decimal Hr/dec	- 'H'
Lon/Lat	- 'L'
Unit vector	- 'U'

If pixel number is the input format then inco (outco) is specified as follows:

Firas (Res 6)	- 'F' or 'R6'
DMR (Res 6)	- 'D' or 'R6'
Dirbe (Res 9)	- 'B' or 'R9'
Resolution m	- 'Rm' where m is <= 15

(Note: The default coordinate system for pixels is ecliptic.)

otherwise use:	Ecliptic	- 'E'
	Galactic	- 'G'
	Equatorial	- 'Q' (Epoch 2000)

Note: Lower case may also be used.

WARNINGS:

****

Input of (0,0) in ECLIPTIC coordinates will result in a division by zero error.
Additionally, input of coordinates that lie exactly on a pixel boundary will yield UNPREDICTABLE results.

****

In order to optimize throughput, this routine does minimal data checking. Improper input such as "unit" vectors with norm not equal to one or pixel numbers with values outside the range defined by the resolution will be processed without crashing, however the output might lead to unexpected results. Therefore it is up to the user to take proper care when generating and/or passing the input.

The forward and inverse operations between pixel numbers and pixel unit vector coordinates have been verified up to and including resolution 12. (The forward procedure is based on the forward_cube.for program in the upx directory.)

EXAMPLE:

1. To convert an array of sky coordinates in 'L' format from ecliptic coordinates to an array in 'U' format in galactic coordinates:

```
output = coorconv(input,infmt='L',outfmt='U',inco='E',outco='G')
```

2. To convert an array of sky coordinates in 'L' format to an array in 'U' format without changing coordinate systems:

```
output = coorconv(input,infmt='L',outfmt='U')
```

3. To convert an array of sky coordinates in 'H' format in equatorial coordinates to an array of Firas pixel numbers:

```
output = coorconv(input,infmt='H',outfmt='P',inco='Q',outco='F')
```

OR

```
output = coorconv(input,infmt='H',outfmt='P',inco='Q',outco='R7')
```

4. To convert an array of resolution 8 pixel numbers to sky coordinates in 'L' format in galactic coordinates:

```
output = coorconv(input,infmt='P',outfmt='L',inco='R8',outco='G')
```

5. To convert an array of Dirbe pixel numbers to an array of DMR pixels:

```
output = coorconv(input,infmt='P',outfmt='p',inco='B',outco='D')
```

Note: You cannot "up res" pixel numbers from a lower resolution to a higher one.

NAME/ONE LINE DESCRIPTION OF ROUTINE:

CPUSEC returns the amount of cpu time consumed as a FLOAT.

DESCRIPTION:

This routine calls the C run-time library function clock() and returns the number of cpu seconds consumed by the current process as a FLOAT value.

CALLING SEQUENCE:

cpu = cpusec()

ARGUMENTS:

none

WARNINGS:

This is the cumulative total for the current process and does not include subprocesses, but does include all previous processing including login. The value matches that given by Control-T, except in format.

EXAMPLE:

Example 1.

```
UIDL> setlog, 'cpusec_exe', 'cgis$idl:cpusec.exe'
UIDL> linkimage, 'cpusec', 'cpusec_exe', 1
UIDL> print, cpusec()
494.34
```

This means that the current process has consumed 8 minutes and 14.34 seconds of cpu time. Control-T would have given output like the following:

```
NODE::USER 11:42:35 COBE_IDL   CPU=00:08:14.34 PF=367 IO=951 MEM=512
```

Example 2.

```
UIDL> t0 = cpusec()
UIDL> ...routine or steps to be timed...
UIDL> t1 = cpusec()
UIDL> print, 'Routine consumed ', t1-t0, ' CPU seconds.'
```

This example shows how to time a routine or several steps of an IDL session, and would normally be used as part of a procedure rather than run interactively as shown.

NAME/ONE LINE DESCRIPTION OF ROUTINE:

DATAIN is a routine which reads in FITS and other format files in UIDL

DESCRIPTION:

Datain will read FITS, IDL Save Sets, and output FITS or IDL Save Sets. It should be noted that some of the parameters in the list below are instrument specific. This procedure is meant to be used for fields within a FITS file which have been pixelized.

WARNINGS:

1) If the input file is an IDL Save Set then the output will be sent directly to an UIMAGE object. If the user wants to get the save set directly to UIDL command line simply use the RESTORE command.

2) This routine is to be used for data fields which have been pixelized. Some checking is done for FIRAS PDS's file types to see if none of the fields are pixelized. If this is true then the routine FIRASMOD is called to read in the data.

   IF YOU TRY AND READ IN A NON-PIXELIZED FIELD FOR A FILE THAT CONTAINS SOME PIXELIZED FIELD THIS ROUTINE WILL BLOW UP DURING THE PIXELIZATION.

CALLING SEQUENCE:

```
datain, dsname, [,format=format] [,intype=intype]
        [,face=face] [,dsfield=dsfield] [,subscr=subscr]
        [,data=data] [,weights=weights] [,frequency=frequency]
        [,instrume=instrume] [,badpixval=badpixval] [,res=res]
        [,units=units] [,wunits=wunits] [,rms=rms] [,coord=coord]
        [,nu_zero=nu_zero] [,delta_nu=delta_nu] [,num_freq=num_freq]
        [,/dbsig] [,/dbqual] [,/simple]
```

ARGUMENTS (I = input, O = output, [] = optional):

param	I/O	type	description
dsname	I	string	full name of file for input
format=format	I	string	FITS or CISS (default to FITS)
intype=intype	I	string	DIRBE, FIRAS, DMR, or XAB, not needed for most functions
face=face	I	integer	face # 0-5, not needed for full skymap
dsfield=dsfield	I	string	field to read in (if FITS BE)
subscr=subscr	I	string	subscripts to read in if FITS BE format '4:7' (note: fields start at one (1).)
dbsig	I	integer	return sigma from photqual (DIRBE)
dbqual	I	integer	return flag from photqual (DIRBE)
simple	I	integer	set flag if read in simple FITS non-skymap, data
data=data	O	float	array for data to be returned to
weights=weights	O	float	array for weights
frequency=frequency	O	float	array for frequency info
instrume=instrume	O	string	the instrument the data is for
badpixval=badpixval	O	float	"no data" indicator
res=res	O	integer	SKYMAP resolution

units=units	O	string	units for data values
wunits=wunits	O	string	units for weights
rms=rms	O	float	DMR specific
coord=coord	O	string	DMR specific
nu_zero=nu_zero	O	float	FIRAS specific
delta_nu=delta_nu	O	float	FIRAS specific
num_freq=num_freq	O	float	FIRAS specific

<u>DMR specific keywords</u>
rms, coord

<u>FIRAS specific keywords</u>
delta_nu, nu_zero, num_freq

<u>DMR and FIRAS specific keywords</u>
weights
wunits

<u>DIRBE and FIRAS specific keywords</u>
frequency, dbsig
subscr, dbqual

EXAMPLE:

```
datain, 'cgis_fits:dmr_skymap_31a.fits',dsfield='SIGNAL',data=data,$
        instrume=instrume

datain, 'cgis_fits:dmr_skymap_53a.fits',dsfield='SIGNAL+WEIGHTS',$
        data=data,instrume=instrume

datain, 'cgis_fits:fip_sky_lhs.fits',dsfield='SIGNAL',data=data,$
        frequency=frequency

datain, 'cgis_fits:fip_sky_lhs.fits',dsfield='SIGNAL',subscr='5:40',$
        data=data,frequency=frequency

datain, '$CGIS_FITS/dirbe_galactic_plane_maps.fits',data=sigma,$
        dsfield='photqual',face=0,/dbsig
```

NAME/ONE LINE DESCRIPTION OF ROUTINE:
DATAOUT is a routine which writes out FITS files or CISS files

DESCRIPTION:
DATAOUT writes out a FITS file or CISS file from UIDL.

WARNINGS:
IDL Save Set output is equivalent to using the IDL command SAVE, except a full sky T will be stored in sixpack format.

CALLING SEQUENCE:

ARGUMENTS (I = input, O = output, [] = optional):

param	I/O	type	description
datafile	I	string	full name of file for output
weightf=weightf	I	string	full name of file for weights
format=format	I	string	FITS or CISS (default to FITS)
face=face	I	integer	face # 0-5 or 7 for full image
data=data	I	float	array for data input
weights=weights	I	float	array for weights
frequency=frequency	I	float	array for frequency info
instrume=instrume	I	string	the instrument the data is for
badpixval=badpixval	I	float	"no data" indicator
res=res	I	integer	SKYMAP resolution
units=units	I	string	units for data values
wunits=wunits	I	string	units for weights
title=title	I	string	title for data
wtitle=wtitle	I	string	title for weights
rms=rms	I	float	DMR specific
coord=coord	I	string	DMR specific
nu_zero=nu_zero	I	float	FIRAS specific
delta_nu=delta_nu	I	float	FIRAS specific
num_freq=num_freq	I	float	FIRAS specific

<u>DMR specific keywords</u>
rms, coord

<u>FIRAS specific keywords</u>
nu_zero, delta_nu, num_freq

<u>DMR and FIRAS specific keywords</u>
weights, wunits

<u>DIRBE and FIRAS specific keywords</u>
frequency

EXAMPLE:
dataout, 'dmr_skymap_31a.fits',data=data,instrume=instrume

NAME/ONE LINE DESCRIPTION OF ROUTINE:
DBERROR separates the SIGMA and QUAL_FLAG from PhotQual.

DESCRIPTION:
This routines takes as input the PHOT_QUAL field of the DIRBE GPM FITS file and extracts the data quality flag and sigma. The input arrays should be a 1,2 or 3 dimensional array with the last dimension as wavelength (10 bands). The map must be either in skycube/face or sixpack format. The first (least significant) byte of the PHOT_QUAL field is ocmposed of a series of bit flags where a total value of zero indicates the standard fitting method was used. A non-zero value indicates a special condition. The second (most significant) byte gives the standard deviation. The first 8 bands are represented as RELATIVE SIGMAS. If the PHOTOMET values are input as well than the absolute error is returned. The last two bands (140, 240 um) the standard deviations themselves are given.

CALLING SEQUENCE:
DBERROR, photqual, sigma, qual_flag [,photomet=photomet]

ARGUMENTS (I = input, O = output, [] = optional):

PHOTQUAL	I	intarr	PhotQual field from DIRBE image.
PHOTOMET	[I]	fltarr	Photomet field from DIRBE image.
BADVAL	[I]	int	Bad Pixel value (def=0).
QUAL_FLAG	O	intarr	Data Quality flag
SIGMA	O	fltarr	Standard deviations

WARNINGS:
none.

EXAMPLE:
1. To generate the Qual_flag and sigma from the PhotQual using the photomet to produce the absolute error for bands 1-8:

dberror,photqual,sigma,qual_flag,photomet=photomet

NAME/ONE LINE DESCRIPTION OF ROUTINE:

DECOMPRESS_MF decompresses dirbe data read from BLI_TOIRSCLD.

DESCRIPTION:

IDL function to decompress the science data word which was compressed by BTL_COMPRESS_SDM_MF. The algorithm is based on BTL_DECOMPRESS_SDM_MF.

CALLING SEQUENCE:

DATA = DECOMPRESS_MF(indata)

ARGUMENTS:

DATA	O	fltarr	An array of floating point (R*4) numbers with same dimension as INDATA.
INDATA	I	intarr	An array of compressed integers (I*2) of arbitrary dimension.

WARNINGS:

None.

EXAMPLE:

To read DIRBE data from BLI for the 1st hour of 01-January-1990, and convert the data from a compressed I*2 array to a decompressed R*4 array:

```
data = READ_ARCV('bli_toirscld.dadrbsci2','9000100','9000101')
data = DECOMPRESS_MF(data)
```

NAME/ONE-LINE DESCRIPTION:

DIPOLE subtracts a dipole term from a scalar sky map.

DESCRIPTION:

An input intensity map (with optional weights) is the basis of a least-squares fit for the position, monopole value, and dipole amplitude of the CMBR. The routine returns these values as well as a residual map with the dipole subtracted. The user can also specify a range of galactic latitude to exclude from the fit. The residual is subtracted from the whole map.

CALLING SEQUENCE:

DIPOLE, inmap, monopole, diampl, diglon, diglat, [galexc=...], [residmap=residmap], [weights=weights], [badval=badval]

ARGUMENTS (I=input, O=output, []=optional):

inmap	I	flt arr	2-D input skymap, in right-T format
monopole	O	flt	Mean CMBR intensity
diampl	O	flt	Dipole amplitude
diglon	O	flt	Galactic longitude of dipole pole (deg)
diglat	O	flt	Galactic latitude of dipole pole (deg)
[galexc]	[I]	flt	Absolute value of minimum galactic latitude to include in fit (deg), default=0.
[residmap]	[O]	flt arr	2-D output residual map
[weights]	[I]	flt arr	2-D map of weights corresponding to input map, default=uniform wts
[badval]	[I]	flt	Flag value for bad pixels, default=0.

WARNINGS:

1. Only a full sky right-T can be used as input. Pixelization is assumed to be in ecliptic coordinates!
2. It is generally wise to specify a galactic latitude exclusion of 20 degrees or more for believable results.
3. Flagged pixels will show up as zeros in the residual map.

EXAMPLE:

If "inmap" is a map of temperatures in which empty pixels have a value of zero then a simple call for an unweighted fit would be

```
UIDL> dipole, inmap, avtemp, diamp, lon, lat, galexc=20
UIDL> print, avtemp, diamp, lon, lat
        2.725    0.003322   265.1    48.3
```

If, e.g., ", residmap=residual" had also been included in the call then the variable "residual" would contain a residual skymap after subtracting a 2.725 K average and 3.322 mK dipole from inmap.

If in addition to inmap we have a second map, e.g. "sigmas" that contains the weight at each pixel, then we could also have included ", weights=sigmas" in the call.

NAME/ONE LINE DESCRIPTION OF ROUTINE:

DIRBE_GAINS returns the auto or commanded gain ratios per detector

DESCRIPTION:

IDL function to return the auto-gain or commanded gain for a specified time.

CALLING SEQUENCE:

GAIN = DIRBE_GAIN (gtype,ztime)

ARGUMENTS (I = input, O = output, [] = optional)

GAIN	O	fltarr	Array of gain ratios in science detector order.
GTYPE	I	int	Gain selection flag (0 = auto_gain, 1 = commanded_gain).
ZTIME	I	str	Time in zulu format.

WARNINGS:

1. Auto-gain ratios were derived from optimized cold-mission data, but they are currently being applied to all data.
2. Gain ratios are not automatically updated when pipeline ratios change.
3. If out-of-range arguments are supplied, -1 will be returned.

EXAMPLE:

1. To get auto-gain ratios on 01-January-1990:
   auto = DIRBE_GAINS(0,'900010000')
2. To get commanded-gains on 01-January-1990:
   comm = DIRBE_GAINS(1,'900010000')

NAME:

DIRBE_MEPS returns the MUX sequence and prints it out.

DESCRIPTION:

IDL function to extract the mux sequence from the DIRBE archive and generate a table of band versus MUX process number.

CALLING SEQUENCE:

MUX = MEPS(START,STOP,[DS=dataset])

ARGUMENTS (I = input, O = output, [] = optional):

START	I	str	Start time in zulu or VAX format.
STOP	I	str	Stop time in zulu or VAX format.
DS	[I]	str	Dataset to read (Default='BMD_MA'). Valid datasets include 'BMD_MA','NBS_CTOD', and 'BLI_TOIRSCLD'.
MUX	O	bytarr	Array of engineering detector numbers in MUX process order. Mux (0,1,2,3...) = Bands (1a,2a,3a,1b...).

WARNINGS:

1. DIRBE_MEPS will only return data for the first major frame in the requested time range.
2. If an invalid dataset name is specified, -1 is returned.

COMMON BLOCKS:

None.

EXAMPLE:

To get the mux sequence at the start of 1 Jan 1990, type:
mux = dirbe_meps('900010000','900010001')

Note: If the BMD_MA file for 1 Jan 1990 was not online, but the BLI_TOIRSCLD file was online, then you would have to type:

mux = dirbe_meps('900010000','900010001',ds='bli_toirscld')

NAME/ONE LINE DESCRIPTION OF ROUTINE:

DIRBE_RAWTOD returns auto-gain corrected data in process order.

DESCRIPTION:

IDL function to read the DIRBE time-ordered data and do a first-order correction for the auto-gain. The data is returned in process order at 16x auto-gain and full commanded gain.

CALLING SEQUENCE:

DATA = DIRBE_RAWTOD(START,STOP [,DS=dataset])

ARGUMENTS (I = input, O = output, [] = optional):

DATA	O	intarr	Array 16 processes x 256 hmnfs x n MFs.
START	I	str	Start time in zulu or VAX format.
STOP	I	str	Stop time in zulu or VAX format.
DS	[I]	str	Dataset to read (Default='BMD_MA'). Valid datasets include 'BMD_MA' and 'NBS_CTOD'.

WARNINGS:

1. This routine is not accurate for data taken at 1X auto-gain. It uses a simple factor of 16 to correct the gain bit. This routine is meant for first-order examination of warm-era data. For cold-era data, the preferred routine is DIRBE_DATA.
2. Data will be returned for all operating modes.
3. Data will be returned in the order that it was processed, NOT in detector order.

COMMON BLOCKS:

None.

EXAMPLE:

1. To get data from BMD_MA for the 1st hour of 01-January-1990:

```
data = DIRBE_RAWTOD('900010000','900010100')
```

2. If you want to read NBS_CTOD instead:

```
data = DIRBE_RAWTOD('900010000','900010100',ds='nbs_ctod')
```

NAME/ONE LINE DESCRIPTION OF ROUTINE:
DIRBE_READ_PIXEL gets data for a single pixel within a time range

DESCRIPTION:
Retrieves all observations of a user-specified skymap quantity, within a specified time period, for a single pixel.

CALLING SEQUENCE:
data = DIRBE_READ_PIXEL(start,stop,fieldname,pixno[,filename])

ARGUMENTS (I = input, O = output, [] = optional):

START	I	str	The time at which DIRBE_READ_PIXEL starts grabbing data from the skymap(s). 'Start' may be specified by the user either as a 'ZULU' time (e.g., 89365010000) or as a'VAX' time (e.g.,31-dec-1989:01:00:00). The user must enclose the start time within single quotes.
STOP	I	str	The time at which DIRBE_READ_PIXEL stops grabbing data from the skymap(s). Acceptable formats are the same as for 'start'. The user must enclose the stop time within single quotes.
FIELDNAME	I	str	The skymap field of interest. The field name must include both the RDL name, as well as the name of the field within the RDL. For example, 'bci_cirssm.photometry(1)' is a valid 'fieldname' but just 'photometry(1)' is not. Only one element of an array may be retrieved in a call. The user must enclose the fieldname within single quotes.
PIXNO	I	int	The pixel number of interest. Only data for this pixel is retrieved.
FILENAME	[I]	str	An optional parameter. Use ONLY if you wish to get data from a non-archive (RMS) file. 'Filename' is the name of the RMS file. The user must enclose the filename within single quotes.
DATA	O	flt arr	The output IDL variable into which the requested observations are stored. 'Data' can have any variable name.

WARNINGS:
1. USE AT YOUR OWN RISK! Use of the routine READ_SKYMAP is preferred in most cases, since DIRBE_READ_PIXEL does not release virtual memory correctly and can cause your IDL session to bomb with repeated use within a session. However, DIRBE_READ_PIXEL possesses the unique capability of reading more than one input skymap, which READ_SKYMAP cannot do.

2. In theory, DIRBE_READ_PIXEL can handle any arbitrary skymap. For RDLs not in the default CSDR$RDF location, the user must redefine CSDR$RDF to a location which contains the correct RDF.

3. If a user specifies a large enough time period to cause DIRBE_READ_PIXEL to access more than one skymap, the user usually will find that the time ranges of the observations returned are not limited to the input time interval. Because of the way the skymap access works, DIRBE_READ_PIXEL returns ALL observations from the accessed skymap files.

4. Byte, I*2,I*4,R*4 and ADT(time) fields are acceptable to DIRBE_READ_PIXEL. Character strings and R*8 are not.
******NOTE!***** Currently, a bug exists which terminates your session if you try to retrieve an ADT TIME field.

5. DIRBE_READ_PIXEL handles ADT (time) fields a little differently from other field types. READ_PIXEL returns elapsed time (in seconds) from the first observation read. The time of the first observation is stored in a separate IDL character string variable with the name of ZTIME.

EXAMPLE:

The user wants to read the MUX process 1 photometry data for pixel 204353 from the DIRBE daily skymaps (BCI_CIRSSM) located in CSDR$DIRBE_ARCHIVE. All daily skymaps which contain data between 23-nov-1989 00:01:23 and 24-nov-1989 00:00:00 are accessed. The output is stored in IDL variable PROC1.

```
proc1 = DIRBE_READ_PIXEL('23-nov-1989:00:01:23',
'24-nov-1989:00:00:00','bci_cirssm.photometry(1)',204353)
```

NAME/ONE LINE DESCRIPTION OF ROUTINE:

DIRBE_READ_PIXSTR gets time strings through a central pixel

DESCRIPTION:

DIRBE_READ_PIXSTR retrieves time-strings of a user-specified skymap quantity, within a specified time period and radius, which passed through a specified 'central' pixel.

CALLING SEQUENCE:

data = DIRBE_READ_PIXSTR(start,stop,fieldname,pixno,radius[,filename])

ARGUMENTS (I = input, O = output, [] = optional):

START	I	str	The time at which DIRBE_READ_PIXSTR starts grabbing data from the skymap(s). 'Start' may be specified by the user either as a CSDR GMT (e.g., 89365010000) or as a 'VAX' time (e.g.,31-dec-1989:01:00:00). The user must enclose the start time within single quotes.
STOP	I	str	The time at which DIRBE_READ_PIXSTR stops grabbing data from the skymap(s). Acceptable formats are the same as for 'start'. The user must enclose the stop time within single quotes.
FIELDNAME	I	str	The skymap field of interest. The field name must include both the RDL name, as well as the name of the field within the RDL. For example, 'bci_cirssm.photometry(1)' is a valid 'fieldname' but just 'photometry(1)' is not. Only one element of an array may be retrieved in a call. The user must enclose the fieldname within single quotes. CURRENTLY, ONLY BCI_CIRSSM-formatted SKYMAPS ARE ACCEPTABLE!
PIXNO	I	int	The pixel number of the 'bullseye' pixel. All timestrings returned by DIRBE_READ_PIXSTR will pass through this pixel.
RADIUS	I	int	Radius of a roughly circular region on the sky centered on the 'bullseye' pixel. Only observations within this radius are used to construct the time-strings. Units of the radius are specified in DIRBE pixels.
FILENAME	[I]	str	An optional parameter. Use ONLY if you wish to get data from a non-archive (RMS) file. 'Filename' is the name of the RMS file. The user must enclose the filename within single quotes.
DATA	O	flt arr	The output IDL variable into which the requested observations are stored. 'Data' can have any variable name. The output data is stored as a 2-dimensional array: the first dimension is the time-string number, and the second dimension contains all the observations for each time string. Since not all time-strings are the same length, the second dimension is the length of the longest time-string -- shorter time-strings are padded out to 0. Note: because the time string lengths are variable, the location of the 'bullseye' pixel is not the same from string to string.

WARNINGS:

1. USE AT YOUR OWN RISK! DIRBE_READ_PIXSTR does not release virtual memory correctly and can cause your IDL session to bomb with repeated use within a session.

2. DIRBE_READ_PIXSTR can only handle BCI_CIRSSM-formatted skymaps. Furthermore, it is limited to skymaps at DIRBE resolution ONLY.

3. Byte, I*2,I*4,R*4 and ADT(time) fields are acceptable to READ_PIXSTR. Character strings and R*8 are not.
******NOTE!***** Currently, a bug exists which terminates your session if you try to retrieve an ADT TIME field.

4. DIRBE_READ_PIXSTR handles ADT (time) fields a little differently from other field types. DIRBE_READ_PIXSTR returns elapsed time (in seconds) from the first observation read for each time-string. The time of the first observation of each time-string is stored in a separate IDL character string array with the name of ZTIME.

5. The data retrieved are strictly limited by the START and STOP times requested. (This is unlike DIRBE_READ_PIXEL).

EXAMPLE:

The user wants to retrieve all PHOTOMETRY(1) timestrings which pass through DIRBE pixel 49152 during the time interval 9007209400377 to 900739500314. The user has limited the radius of the search region to 6 DIRBE pixels.

```
data = READ_PIXSTR('9007209400377', '9007309500314',
'BCI_CIRSSM.PHOTOMETRY(1)', 49152, 6)
```

The timestrings are returned into IDL variable data.

NAME\ONE LINE DESCRIPTION OF ROUTINE:

DIRBE_TOD reads DIRBE data and corrects for order and gains.

DESCRIPTION:

This routine reads DIRBE data and converts it from process order to science-detector order. If data is read from NBS_CTOD or BMD_MA, DIRBE_TOD will correct for auto gain and commandable gain using pipeline gain correction ratios. If data is read from BLI_TOIRSCLD, DIRBE_TOD will decompress it.

CALLING SEQUENCE:

DATA = DIRBE_TOD(START,STOP [,MFTIME,DET=detnum,FMT=infmt, DS=dataset,OMS=mode,AUTO=again,COMM=cgain])

ARGUMENTS (I = input, O = output, [] = optional):

DATA	O	fltarr	DIRBE Intensities per band. By default, data will be read from BMD_MA and returned for all bands and all operating modes in 1x DNs.
START	I	str	Start time in zulu or VAX format.
STOP	I	str	Stop time in zulu or VAX format.
MFTIME	[O]	strarr	Array of MF times in zulu format.
DET	[I]	int	Detector number (0-15) = (1A,1B,...10). If specified, only one band will be returned.
DS	[I]	str	Dataset to read (Default='BMD_MA'). Valid datasets include 'BMD_MA','NBS_CTOD', and 'BLI_TOIRSCLD'.
FMT	[I]	int	Output format of data array. If present and set to: 0: DATA = FLTARR(256*MF,16) (Default) 1: DATA = FLTARR(16,256,MF) where MF is number of Major Frames read.
OMS	[I]	int	Operating mode. If present and set to: -1: SDM and CAL modes returned (Default) 0: SDM mode data will be returned 1: CAL mode data will be returned
AUTO	[I]	int	Auto-gain correction switch. If set to: 0: Data will be corrected to 1x auto-gain (Default). 1: Data will be corrected to 16x auto-gain. 2: No auto-gain correction, but gain-bit will be factored out.
COMM	[I]	int	Commanded-gain correction switch. If set to: 0: Data corrected for commanded-gain (Default). 1: No commanded-gain correction.

WARNINGS:

1. This routine processes the MUX for the first major frame of data ONLY. It DOES NOT check after that.
2. DIRBE_TOD will not work for mux sequences with redundant processes.
3. BLI CAL mode data is not corrected for commanded gain.
4. COMM and AUTO switches are ignored when reading from BLI_TOIRSCLD.
5. The amount of data which can be read at one time is usually limited to approximately three hours, depending on available memory.

EXAMPLES:

1. To get data from BMD_MA for all bands, both CAL and SDM modes, in 1x DNs for the 1st hour of 01-January-1990:

```
data = DIRBE_TOD('900010000','900010100')
```

2. If just CAL mode is desired:

```
data = DIRBE_TOD('900010000','900010100',oms=1)
```

3. If just CAL mode is desired, you want the array divided into MFs, and you want the MF times:

```
data = DIRBE_TOD('9000100','9000101',mftime,oms=1,fmt=1)
```

4. If just CAL mode is desired, and no commanded gain correction:

```
data = DIRBE_TOD('9000100','9000101',oms=1,comm=1)
```

5. If you want to read BLI data for SDM mode only:

```
data = DIRBE_TOD('9000100','9000101',ds='bli_toirscld',oms=0)
```

6. If you want to read BLI data for band 2A only:

```
data = DIRBE_TOD('9000100','9000101',det=3,ds='bli_toirscld')
```

NAME/ONE-LINE DESCRIPTION:

DOWNRES "coarsens" the resolution of a skycube or cube face.

DESCRIPTION:

"Downsizes" a skycube or cube face NSTEPS in resolution, i.e. turns an order-N cube into one of order N-NSTEPS. Each unit of nsteps is a factor of 2 in spatial resolution. Both 2-D and 3-D maps are handled; if 3-D, the third dimension is assumed to be non-spatial so its resolution is unchanged.

CALLING SEQUENCE:

outmap = DOWNRES (inmap, nsteps, [badval=bad_pixel_value])

ARGUMENTS (I=input, O=output, []=optional):

outmap	O	2D/3D arr	Smoothed map, smaller than inmap by a factor of 2^nsteps in x and y
inmap	I	2D/3D arr	Input map: byte, real or integer
nsteps	I	int	Number of steps to decrease res'n
badval	[I]	flt	The value of pixels assumed to be empty or bad; defaults to 0.

WARNINGS:

1. Pixels whose value is equal to badval are not included in the averaging process. Thus, stripes whose width is smaller than 2^nsteps pixels will appear to have disappeared in the low resolution output map.
2. Input map must have either a 3:4 aspect ratio in y:x (unfolded cube), a 2:3 ratio ("sixpacked" cube), or be square. (In the latter case, we assume we are looking at a single cube face.)
3. Only a single bad value is accepted. Maps with multiple sentinel values (e.g. some DIRBE maps) will be confused unless changed to a single value prior to being "DOWNRES"ed.

EXAMPLE:

outmap = DOWNRES(inmap, 3, badval=-999)

This would convert e.g., a 1024x768 DIRBE map (i.e. order 9) into a (128x96) FIRAS map (order 6).

NAME/ONE LINE DESCRIPTION OF ROUTINE:

DO_MEPS applies MUX sequence to one major frame of DIRBE data.

DESCRIPTION:

Given an input 16x256 DIRBE science data array arranged in process order, plus the MUX sequence appropriate for that data array, DO_MEPS produces an output science data array which has been sorted into detector order (1a,1b,1c,2a...10)

CALLING SEQUENCE:

OUT_DATA = DO_MEPS(IN_DATA, MEPS)

ARGUMENTS (I = input, O = output, [] = optional):

IN_DATA	I int arr	16-band x 256-sample data array. (Typically, the user has obtained this from field DADRBSCI2 of either NBS_CTOD, BMD_MA, BLI_TOIRSCLD)
MEPS	I byt arr	MEPS sequence (user supplied from DAMEPS field of either NBS_CTOD, BMD_MA, BLI_TOIRSCLD).
OUT_DATA	O int arr	16x256 output data array, resorted into in detector order.

WARNINGS:

1. Only works properly if detectors are not sampled redundantly within the MUX.

EXAMPLE:

The user has obtained 1 Major Frame's worth of SDM data and has stored in in a 16x256 integer array called SCIDAT. He/she has also obtained the appropriate MEPS sequence for that major frame (e.g., from BMD_MA.DAMEPS) and stored it in IDL variable MEPS_SEQ. The following call to DO_MEPS will re-order the 256 observations for each of the 16 DIRBE channels into detector order (1a,1b,1c....10), and place them in output array outsci:

outsci = do_meps(scidat,meps_seq)

NAME/ONE LINE DESCRIPTION OF ROUTINE:

IDL$EDIV performs quadword integer division.

PURPOSE:

Performs quadword integer division

CALLING SEQUENCE:

idl$ediv, dividend, divisor, quotient, remainder

INPUT:

dividend -	quadword (2 dim longword array)
divisor -	quadword (2 dim longword array)

OUTPUT:

quotient -	quadword (2 dim longword array)
remainder -	quadword (2 dim longword array)

NAME/ONE LINE DESCRIPTION OF ROUTINE:
EMUL performs quadword integer multiplication.

PURPOSE:
Performs quadword integer multiplication

CALLING SEQUENCE:
emul, multiplicand1, multiplicand2, product

INPUT:

multiplicand1 -	quadword (2 dim longword array)
multiplicand2 -	longword

OUTPUT:

product -	quadword (2 dim longword array)

NAME/ONE LINE DESCRIPTION OF ROUTINE:

FDECOMP decomposes a file name into its components.

DESCRIPTION:

IDL procedure to decompose a file name into its constituent components (disk, directory, file name, file name extension, and version number).

CALLING SEQUENCE:

FDECOMP,filename,disk,dir,name,qual,version

ARGUMENTS (I = input, O = output, [] = optional):

filename	I	str	File name to be decomposed
disk	O	str	Disk name
dir	O	str	Directory name
name	O	str	File name
qual	O	str	File name extension
version	O	str	Version number

WARNINGS:

None.

EXAMPLE:

To decompose the file name
FIRCOADD:[FIRAS.SKYMAP]FCF_SKY_LLSS.ED_8934301_9026410;2:

```
UIDL> filename =
'FIRCOADD:[FIRAS.SKYMAP]FCF_SKY_LLSS.ED_8934301_9026410;2'
UIDL> fdecomp,filename,disk,dir,name,qual,ver
```

FDECOMP returns these strings for the file name components:

```
disk = 'fircoadd'
dir  = '[firas.skymap]'
name = 'fcf_sky_llss'
qual = 'ed_8934301_9026410'
ver  = '2'
```

NAME/ONE-LINE DESCRIPTION:

FIRASMOD: used to read in the FIRAS Calibration Model IPs, FIRAS Line Profile PDSs, and other FIRAS PDS.

DESCRIPTION:

UIDL routine which reads in fields of the FIRAS IP/PDS files. This is mainly used for non-skymap (i.e. not pixelized) fields.

CALLING SEQUENCE:

firasmod, dsname, dsfield, data=data, units=units, message=message

INPUTS:

dsname:	the name of the data file to be read
dsfield:	the name of the field to be read in (TTYPE field in the fits extension header. Pass this parameter in as a question mark ('?') in order to get a list of fieldnames.
data	array for the data that was read in to be returned to
units	var to hold the units of the returned data
message	Y (or not set) for messages to be printed as operations are performed, N to be silent

NAME/ONE LINE DESCRIPTION OF ROUTINE:

FITMU is a Bose_Einstein / Dust Residual fitting facility.

DESCRIPTION:

This IDL function fits spectra to a Bose_Einstein spectrum plus a residual Planck (blackbody) spectrum. The blackbody includes corresponding optical depth and emissivity parameters. Therefore there are five fitting parameter in all. In most cases it is not possible to fit all five at once, therefore parameters can be designated as fixed. The initial values of the parameters are supplied by the user (either using a command line keyword or interactively) as well as their variability (fixed or variable). It is also possible to designate fixed value parameters that are different for each input spectrum. Particular ranges of the spectra can be masked if desired. This is useful for avoiding spectral lines.

Model: fit = BE(T0,mu;freq) + op_depth * (freq)^emiss * P(T1;freq)

where BE(T,mu;freq) is a Bose-Einstein spectrum and P(T;freq) is a Planck (Blackbody) spectrum.

CALLING SEQUENCE:

```
fit = fitmu(freq,spec,[weight],[parm0=parm0],[active=active],
[cmb_temp=cmb_temp],[cmb_mu=cmb_mu],[bb_temp=bb_temp],
[opdep=opdep],[emis=emis],[mask=mask],[tolerance=tolerance],
[out_parm=out_parm],[parm_sig=parm_sig],[residual=residual],
[fail=fail],[chi2=chi2],[n_iter=n_iter],[funits=funits], [/eplees],[/wcm2],[/mjy])
```

ARGUMENTS (I = input, O = output, [] = optional):

freq	I	flt arr	Spectra Frequency (in icm)
spec	I	flt arr	Spectra (in eplees)
[weight]	I	flt arr	Spectral Weights
[parm0]	I	flt arr	Initial Parameter Values
[active]	I	byte arr	Parameter Variability [1 - vary, 0 - fixed]
[cmb_temp]	I	flt arr	CMP Temp value array
[cmb_mu]	I	flt arr	CMB chemical potential
[bb_temp]	I	flt arr	BB Temp value array
[opdep]	I	flt arr	Optical Depth value array
[emis]	I	flt arr	Emissivity value array
[mask]	I	int arr	Masked element array
[tolerance]	I	flt scl	Convergence Tolerance
[out_parm]	O	flt arr	Final Fit Parameter values
[parm_sig]	O	flt arr	Final Fit Parameter sigmas
[residual]	O	flt arr	Fit Residuals
[fail]	O	byte arr	Fit Failure Flag
[chi2]	O	flt arr	Chi_squared/df
[n_iter]	O	int arr	# iterations for fit
fit	O	flt arr	Model fits to spectra
[funits]	I	str	Frequency Units 'icm' (default), 'microns', 'GHz'
[/eplees]	I	qual	Results in eplees (default)
[/wcm2]	I	qual	Results in W/cm^2/sr/K
[/mjy]	I	qual	Results in MJy/sr

The spectrum and weight arrays must have the same number of elements and the same number of dimensions, which can be one (a single spectrum), two (an array of spectra), or three (an array of spectra from a skymap). The last dimension should correspond to the frequency array.

The order of the values in the initial parameter array, 'parm0', is: Bose-Einstein Temperature (in K), Bose-Einstein Chemical Potential, Residual Blackbody Temperature, Residual Optical Depth, Residual Emissivity.

The cmb_temp, cmb_mu, bb_temp, opdep, and emis1 input arrays are used to define fixed parameter values for the individual spectra.

The mask vector contains the indicies of those frequencies where no attempt is made to fit the spectra.

The tolerance value is the minimum relative change in the chi_squared for the fit that must be reached for convergence. The default value is 1e-4. A maximum of 20 iterations is allowed.

If the fit fails to converge in 20 iterations or one of the fitted values becomes negative, then the fit is considered to have failed and the failure flag is set to 1, otherwise it is set to 0. It is, of course, the responsibility of the user to check the final fitted values for credibility.

WARNINGS:

Some initial parameter values may lead to non-convergence. The user is advised to experiment with various sets of initial values on a representative set of spectra to determine a sufficiently robust set. We have found in practice, that the optical depth parameters converge better if they are overestimated.

EXAMPLE:

To fit a set of FIRAS low frequency spectra to a Bose_Einstein spectrum plus a residual dust spectrum using the dust temperatures derived from a previous fit to the high frequency spectra as fixed parameters:

```
fit_be = fitmu(freq_lo,spec_lo,weight_lo, parm0=[2.72,0.001,0,2e-7,1.5],
active=[1,1,0,1,0], bb_temp=hi_temp, out_parm=parm_be,fail=fail_be)
```

The initial values of the parameters are 2.72K for the BE temperature, 0.001 for the BE chemical potential, 2e-7 for the dust optical depth, and 1.5 for the dust emissivity. The BE temperature and chemical potential and the dust optical depth are allowed to vary. The dust emissivity is fixed (active(4) = 0). The dust temperature is read in from the 'hi_temp' array.

If either the 'parm0' or 'active' keywords were missing, then the user would be prompted for their values.

NAME/ONE LINE DESCRIPTION OF ROUTINE:

FITS2X (VMS only) reads FITS hdr, array from tape to IDL arrays

DESCRIPTION:

Read FITS file from tape directly into an IDL image array and FITS header array. One can optionally provide a compression factor to compress the image before writing into the IDL arrays. The procedure is useful for quickly checking the FITS images on a tape. Must mount tape, define logical BEFORE entering IDL.

CALLING SEQUENCE:

FITS2X,unit,outbuf,header,oheader,[nr,nfile,nbox]

ARGUMENTS (I = input, O = output, [] = optional):

unit	I	int	tape unit number
outbuf	O		(as for tape FITS array) data array
header	O		header array after data compression
oheader	O	str arr	header array from tape, before compress.
nr	[I]	int	number of FITS records (2880 bytes) per block; 1-10 allowed; default 1
nfile	[I]	int	file number on tape to be read; default 1
nbox	[I]	int	compression factor; skip every nbox pixels, nbox lines; 1=no compression

WARNINGS:

1. VMS only since uses VMS tape handling.
2. BEFORE invoking IDL must mount the tape as foreign and define MTn as equivalent to the actual name of the tape drive, where n is the unit number input.
3. Will not handle files with groups or parameter blocks.
4. Will only handle BYTE, I*2 (16 bit), or I*4 (32 bit) data.
5. Will only handle one or two-dimensional arrays

EXAMPLE:

To read the first FITS file from a tape mounted on MUA0: on SCINT on which the FITS records have a blocking factor of 10, putting header records in the string array headarr, and the data into datarr of appropriate format (BYTE, I*2, I*4 as on tape), doing no compression:

```
logon to SCINT or CSDR2
$ allocate MUA0:
$ mount/foreign/recordsize=2880,blocksize=28800/density=6250 MUA0:
$ define MT1 MUA0:
$ UIDL
UIDL> FITS2X,1,datarr,headarr,comphead,10
```

NAME/ONE LINE DESCRIPTION OF ROUTINE:

FITSBYTE byteswaps between FITS and internal machine formats.

DESCRIPTION:

To convert an image array from FITS byte ordering to the ordering used by the computer, and vice-versa.

CALLING SEQUENCE:

FITSBYTE, im [, /host]

ARGUMENTS (I = input, O = output, [] = optional):

im	I,O	arr	The image array (both input and output).
HOST	[I]	keywd	If present and non-zero, the image is converted to "host" (machine internal) form. Otherwise, it is converted to "network", or FITS, form.

WARNINGS:

1. This routine alters the input array values on output.

EXAMPLE:

To convert data array DATA from FITS byte ordering to machine-internal ordering:

UIDL> FITSBYTE,data,/host

NAME/ONE LINE DESCRIPTION OF ROUTINE:

FITSHDR reads and displays the primary FITS header.

PURPOSE:

This Procedure will display the primary fits header in a fits file

CALLING SEQUENCE:

FITSHDR, dsname

INPUTS:

dsname the name of the file to display

OUTPUTS:

None, the file is displayed using uscroll

NAME/ONE LINE DESCRIPTION OF ROUTINE:

FITSLIST (VMS) reads FITS files from tape, writes hdrs to dsk/scrn

DESCRIPTION:

Procedure will read FITS files from a tape on the (interactively specified) tape unit. The headers are placed in a file (interactively specified) with the default extension of .LIS. Headers are also displayed at the terminal. FOR VMS IDL ONLY!

CALLING SEQUENCE:

FITSLIST
FITSLIST,update_switch

ARGUMENTS (I = input, O = output, [] = optional):

UPDATE_SWITCH	[I]	byt	If passed and nonzero, => an existing file specified at interactive prompt is opened, and output is appended to end of this file.

WARNINGS:

1. Can only be used with VMS IDL.
2. Tape must be mounted before calling FITSLIST.
3. Interactively prompts for tape unit number on which tape is physically mounted, and name for disk file to which header information is to be written.
4. If the update switch is passed and nonzero, and existing file must be named at the prompt.
5. The FITS tape is not rewound prior to starting the read. This is useful if the tape contains spurious EOF marks.

EXAMPLE:

To read a FITS tape and append the header information to a fill (whose name will be specified interactively:

```
UIDL> FITSLIST,1
```

NAME/ONE LINE DESCRIPTION OF ROUTINE:

FITSPEC is a multiple blackbody fitting facility.

DESCRIPTION:

This IDL function fits spectra to a three blackbody model. One blackbody typically corresponds to the cosmic background while the other two can be used to fit the residual dust. These last two spectra include corresponding optical depth and emissivity parameters. Therefore there are seven fitting parameter in all. In most cases it is not possible to fit all seven at once, therefore parameters can be designated as fixed. The initial values of the parameters are supplied by the user (either using a command line keyword or interactively) as well as their variability (fixed or variable). It is also possible to designate fixed value parameters that are different for each input spectrum. Particular ranges of the spectra can be masked if desired. This is useful for avoiding spectral lines.

```
Model: fit = P(T0;freq) + op_depth1 * (freq/f_char)^emiss1 * P(T1;freq)
                + op_depth2 * (freq/f_char)^emiss2 * P(T2;freq)
```

where P(T;freq) is a Planck (Blackbody) spectrum.

CALLING SEQUENCE:

```
fit = fitspec(freq,spec,[weight],[f_char=f_char],[parm0=parm0],[active=active], $
[cmb_temp=cmb_temp],[bb1_temp=bb1_temp],[bb2_temp=bb2_temp], $
[opdep1=opdep1],[opdep2=opdep2],[emis1=emis1],[emis2=emis2], $
[mask=mask],[tolerance=tolerance],[out_parm=out_parm] $
[parm_sig=parm_sig],[residual=residual],[nodust=nodust], $
[dust=dust],[fail=fail],[chi2=chi2],[n_iter=n_iter], $
[funits=funits],[/eplees],[/wcm2],[/mjy],[/no_show])
```

ARGUMENTS (I = input, O = output, [] = optional):

freq	I	flt arr	Spectra Frequency
spec	I	flt arr	Spectra
[weight]	I	flt arr	Spectral Weights
[f_char]	I	flt	Characteristic Frequency
[parm0]	I	flt arr	Initial Parameter Values
[active]	I	byte arr	Parameter Variability [1 - vary, 0 - fixed]
[cmb_temp]	I	flt arr	CMB Temp value array
[bb1_temp]	I	flt arr	BB 1 Temp value array
[bb2_temp]	I	flt arr	BB 2 Temp value array
[opdep1]	I	flt arr	Optical Depth 1 value array
[opdep2]	I	flt arr	Optical Depth 2 value array
[emis1]	I	flt arr	Emissivity 1 value array
[emis2]	I	flt arr	Emissivity 2 value array
[mask]	I	int arr	Masked element array
[tolerance]	I	flt scl	Convergence Tolerance
[out_parm]	O	flt arr	Final Fit Parameter values
[parm_sig]	O	flt arr	Final Fit Parameter sigmas
[residual]	O	flt arr	Fit Residuals
[nodust]	O	flt arr	Spectrum w/dust fit subtracted
[dust]	O	flt arr	Spectrum w/cmb fit subtracted
[fail]	O	byte arr	Fit Failure Flag
[chi2]	O	flt arr	Chi_squared/df

[n_iter]	O	int arr	# iterations for fit
fit	O	flt arr	Model fits to spectra
[funits]	I	str	Frequency Units: 'icm' (default), 'microns', 'GHz'
[/eplees]	I	qual	Results in eplees (default)
[/wcm2]	I	qual	Results in W/cm^2/sr/K
[/mjy]	I	qual	Results in MJy/sr
[/no_show]	I	qual	Suppress initial parameter display

The spectrum and weight arrays must have the same number of elements and the same number of dimensions, which can be one (a single spectrum), two (an array of spectra), or three (an array of spectra from a skymap). The last dimension should correspond to the frequency array.

The order of the values in the initial parameter array, 'parm0', is: Pure Blackbody Temperature (in K), First Residual Blackbody Temperature, First Residual Optical Depth, First Residual Emissivity, Second Residual Blackbody Temperature, Second Residual Optical Depth, Second Residual Emissivity.

The cmb_temp, bb1_temp, bb2_temp, opdep1, opdep2, emis1, and emis2 input arrays are used to define fixed parameter values for the individual spectra.

The mask vector contains the indices of those frequencies where no attempt is made to fit the spectra.

The tolerance value is the minimum relative change in the chi_squared for the fit that must be reached for convergence. The default value is 1e-4. A maximum of 20 iterations is allowed.

If the fit fails to converge in 20 iterations or one of the fitted values becomes negative, then the fit is considered to have failed and the failure flag is set to 1, otherwise it is set to 0. It is, of course, the responsibility of the user to check the final fitted values for credibility.

WARNINGS:

Some initial parameter values may lead to non-convergence. The user is advised to experiment with various sets of initial values on a representative set of spectra to determine a sufficiently robust set. We have found in practice, that the optical depth parameters converge better if they are overestimated.

EXAMPLE:

To fit a set of high frequency FIRAS spectra to a pure blackbody (for the CMB) and a dust blackbody with the emissivity fixed at 1.5:

```
fit_hi = fitspec(freq_hi,spec_hi,weight_hi, $
        parm0=[2.72,20,2e-5,1.5,0,0,0],f_char=60.0, $
        active=[1,1,1,0,0,0,0],mask=[82,107,108,109], $
        out_parm=parm_hi,fail=fail_hi,parm_sig=parm_sig)
```

The initial values of the parameters are 2.72K for the CMB BB, 20K for the dust BB, 2e-5 for the dust optical depth, and 1.5 for the dust emissivity. Setting the second residual blackbody temperature to zero eliminates it from the model. The first three parameters are allowed to vary. The emissivity is fixed (active(3) = 0). The spectrum values at the four frequency points (82,107,108,109) are ignored (masked) for the fit. The optical depth coefficients give the ratio of the fit to a Planck at the fit temperature at 60 icm. The final fit parameters are returned in the parm_rhss array and their sigmas in the parm_sig array.

If either the 'parm0' or 'active' keywords were missing, then the user would be prompted for their values.

To fit a set of FIRAS low frequency spectra to a CMB BB and a residual dust spectrum using the dust temperatures derived from the high frequency fit as fixed parameters:

```
fit_lo = fitspec(freq_lo,spec_lo,weight_lo, $
        parm0=[2.72,0,2e-5,1.5,0,0,0],f_char=60.0, $
        active=[1,1,0,0,0,0,0], bb1_temp=parm_hi(*,1), $
        out_parm=parm_lo,fail=fail_lo)
```

NAME/ONE LINE DESCRIPTION OF ROUTINE:

FITSTAPE performs magnetic tape FITS format I/O

DESCRIPTION:

Function to perform magnetic tape FITS format I/O. The user is allowed to both read and write to tape, but must thoroughly understand FITS formats and IDL tape i/o before successfully using this routine.

CALLING SEQUENCE:

status=FITSTAPE(command,unit,bitpix,data)

ARGUMENTS (I = input, O = output, [] = optional):

command	I	str	string command from the following choices 'init' - initialization (MUST be first call to FITSTAPE) 'read' - get next 2880 byte data buffer 'eof' - check for end of tape file 'write'- write 2880 byte data buffer 'weof' - empty buffer and write end-of-file
unit	I	int	tape unit number. Valid range is from 0-9. Assignment of the tape unit number is accomplished via the VMS DEFINE command, PRIOR to entering IDL. For more details, see Chapter 13 of the IDL manual, or the EXAMPLE and WARNINGS sections below.
bitpix	[I]	int	Has different meanings, depending on whether the user has chosen the 'INIT' command or the 'READ/WRITE' commands. IF the user has chosen either the 'READ; or 'WRITE' command, then BITPIX is a REQUIRED parameter, and its value is the number of bits/per data element, and is used by the FITSTAPE procedure to control byte swapping. IF the user has chosen the 'INIT' command, the BITPIX parameter gives tape blocking factor, i.e., the number of 2880 byte records per tape record. If not supplied, a default blocking factor of 1 is assumed.
data	I/O	byt arr	2880 byte data array if 'write' command 2880 byte data array if 'read' command
status	O	str	returned as the function value with the following meanings: 'init' = 1 'read' = !err returned by taprd 'write' = 1 'eof' = 1 if at end of file 0 if not at end of file 'weof' = 1

WARNINGS:

1. Use at your own risk. This routine has not been validated.
2. Limited to VMS machines ONLY.
3. PRIOR to entering IDL, the tape must be mounted using the VMS MOUNT/FOREIGN command.
4. PRIOR to entering IDL, a logical name MTn (where n ranges from 0 to 9) must be assigned to the tape drive using the VMS DEFINE command.
5. Be sure to call fitstape with 'init' command first.

EXAMPLE:

The user has formatted his data array into a 2880 byte array, called DATA, as required for FITS tape output. The data has 16 bits per pixel. The user now wants to write DATA to magnetic tape MUA0:, with a tape blocking factor of 1:

BEFORE ENTERING IDL, the user enters the following VMS commands:

```
$ mount/foreign mua0:
$ define mt1 mua0:      (this assigns IDL tape unit 1 to MUA0:)
```

WITHIN IDL:

```
UIDL> status = fitstape('init',1)              ; init tape unit 1 with default blocking factor
UIDL> status = fitstape('write',1,16,data)     ; write data to unit 1
UIDL> status = fitstape('weof',1)              ; write end of file
```

NAME:

FITS_INFO extracts header and data information about a FITS file.

PURPOSE:

Provide information about the contents of a FITS file, (primary header and data if any + extensions). Information is printed at the terminal and/or stored in a common block

CALLING SEQUENCE:

FITS_INFO, filename, [SILENT = , TEXTOUT = , N_ext =]

INPUT:

FILENAME = Scalar or vector string giving the name of the FITS file(s). Can include wildcards such as '*.fits'

COMMON BLOCKS:

DESCRIPTOR = File descriptor string of the form N_hdrrec Naxis IDL_type
Naxis1 Naxis2 ... Naxisn [N_hdrrec table_type Naxis
IDL_type Naxis1 ... Naxisn] (repeated for each extension)
See the procedure RDFITS_STRUCT for an example of the use of this common block

OPTIONAL INPUT KEYWORDS:

SILENT - This key word will suppress display of the file description on the terminal
TEXTOUT - specifies output device.

textout=1	TERMINAL using /more option
textout=2	TERMINAL without /more option
textout=3	<program>.prt
textout=4	laser.tmp
textout=5	user must open file, see TEXTOPEN
textout = filename	(default extension of .prt)

OPTIONAL OUTPUT KEYWORD:

N_ext - Returns an integer scalar giving the number of extensions in the FITS file

EXAMPLE:

Display info about all FITS files of the form '*.fit' in the current directory

```
IDL> fits_info, '*.fit'
```

Any time a *.fit file is found which is *not* a FITS an error message is displayed at the terminal and the program continues

NAME/ONE LINE DESCRIPTION OF ROUTINE:

FLUX2MAG converts flux in ergs/s/cm^2/A to magnitude.

DESCRIPTION:

IDL function that converts from flux (ergs/s/cm^2/A) to magnitude. Use MAG2FLUX for converting from magnitude to flux.

CALLING SEQUENCE:

RESULT = FLUX2MAG(flux,[zero_pt])

ARGUMENTS (I = input, O = output, [] = optional):

RESULT	O	flt arr	Magnitude. (= -2.5*alog10(flux) - zero_pt).
FLUX	I	flt arr	Flux in ergs/s/cm^2/A.
ZERO_PT	[I]	flt	Zero point level of the magnitude. If not specified, default is zero_pt = 21.1 (Code et al. 1976, Ap. J.).

EXAMPLE:

To calculate the magnitude of a star whose flux is 5.0 E-10 erg/s/cm^2/A using a zero point of 21.0:

```
UIDL> mag=flux2mag(5.0e-10,21.0)
```

NAME/ONE LINE DESCRIPTION OF ROUTINE:
FULLMAP creates a full DSK7 skymap file from a partial one.

PURPOSE:
Creates a full DSK7 skymap file from a partial one.

CATEGORY:
DMR Image Analysis.

CALLING SEQUENCE:
FULLMAP,'Partial_Skymap_Filename'

INPUTS:
Filename = Filename of partial skymap; must be in single-quotes.

OUTPUTS:
None; but it does create an output file named: 'Filename.Ext_FULL'

RESTRICTIONS:
Only works with DSK7_SKY.rdl record structures.

NAME/ONE-LINE DESCRIPTION:

GAPFILL interpolates across gaps in an unfolded skycube.

DESCRIPTION:

GAPFILL uses Delaunay triangulation to interpolate across the gaps in an unfolded skycube. It will work in principle on any array; the only significance of the cube is that the shape of the good data area is known so that the array points outside of the T are ignored. This routine is NOT intended for rigorous work, since the interpolation is only linear and no statistical weighting is employed; it is primarily for creating nicely filled-in maps for presentations and other glitz.

CALLING SEQUENCE:

outmap = GAPFILL (inmap, [BADVAL=flag_value])

ARGUMENTS (I=input, O=output, []=optional):

OUTMAP	O	2-D array	Output interpolated map
INMAP	I	2-D array	Input map with gaps
[BADVAL]	[I]	scalar	Flag value, i.e. the value in the gaps. Default = 0

WARNINGS:

Don't try to do anything serious with the output.

EXAMPLE:

If a DIRBE map "dmap" has gaps containing a sentinel value -16000 then the call

```
Nice_Map = GAPFILL (dmap, badval=-16000)
```

will produce a completely filled-in map in the variable Nice_Map.

NAME/ONE LINE DESCRIPTION OF ROUTINE:

GETNAME searches for specified file name, returns full name & path.

PURPOSE:

GetName searches for a file whose name is specified in the input parameter FileName and returns the fully qualified name and path. The function looks in the directories whose names are contained in the input string array (or string) Paths. Upon the first instance of the file, the directory name and file name are concatonated and returned. A null string is returned if the file is not found. The keyword IDLPath causes GetName to search directories in the default directory and the IDL path (!path) first.

CATEGORY:

I/O Utility

CALLING SEQUENCE:

Name = GetName(FileName [, Paths] [,/IDLPath])

INPUTS:

FileName = a simple file name (no directory spec.). If this is the only parameter used, the default directory is searched.

Paths = a directory name or an array of directory names. The full file name specification of the first occurance of the file is returned. The default directory can be specified using a blank element.

IDLPath = will cause the program to search through the default directory first, then any directories in the IDL path (!path), and finally any directories in Paths (if specified).

OUTPUTS:

A string containing the fully qualified file or a null if the file was not found.

EXAMPLES:

Find and return the name of a file on !path:

```
UIDL>print, getname( "planck.pro", /idlpath )
        CGIS$ROOT:[IDL]PLANCK.PRO4
```

Identify a special path and find a file and its name:

```
UIDL>paths = [ "CGIS$UITREE:", "CSDR$BULLETIN:[IDL]" ]
UIDL>print, getname( "news.asc", paths )
        CSDR$BULLETIN:[IDL]NEWS.PRO4
```

If GetName can not find the file, it returns a null string:

```
UIDL>print, getname( "fudged.data", paths, /idlpath )

UIDL>
```

NAME/ONE LINE DESCRIPTION OF ROUTINE:

GETOPT converts a response string into a scalar or vector.

DESCRIPTION:

Function to convert a user's response string into a valid scalar or vector. Distinct elements in the string may be separated by either a comma or a space. The output scalar or vector can be specified to be either integer or floating point. A null string is converted to a zero. !ERR is set to the number of elements supplied. If an input is invalid, !ERR is set to -1 and the result is set to 999.9.

CALLING SEQUENCE:

output=GETOPT(input,[type],[numopt])

ARGUMENTS (I = input, O = output, []=optional):

OUTPUT	O	flt [int]	Scalar or vector containing the numeric conversion of the fields in the string INPUT. If TYPE is not specified, floating point output is assumed.
INPUT	I	str	String that was input by user in response to a prompt.
TYPE	[I]	str	Specifies integer or floating point output.
NUMOPT	[I]	int	If specified, output vector is truncated to NUMOPT elements. If less than NUMOPT values are supplied the output vector will be padded with zeros. If left out, the number of elements in OUTPUT will equal the number of distinct fields in INPUT.

WARNINGS:

NONE.

EXAMPLE:

1. Convert response string to vector.

```
INPUT = '1, 2, 3'
OUTPUT = GETOPT(INPUT)
PRINT, OUTPUT
        1.00000, 2.00000, 3.00000
```

2. Convert response string to vector.

```
INPUT = '1, 2, 3'
TYPE  = 'I'      ;specify integer format for output
OUTPUT = GETOPT(INPUT,TYPE)
PRINT, OUTPUT
        1, 2, 3
```

3. Convert response string to vector.

```
INPUT = '1, 2, 3'
TYPE  = 'I'                  ;specify integer format for output
NUMOPT = 2                   ;specify truncation to 2 elements
OUTPUT = GETOPT(INPUT,TYPE,NUMOPT)
PRINT, OUTPUT
        1, 2
```

NAME/ONE LINE DESCRIPTION OF ROUTINE

GETPRO copies IDL procedure into current default directory.

DESCRIPTION:

IDL procedure to extract a procedure from an IDL Library or directory given in the !PATH system variable and place it in the current default directory (presumably to be edited by the user).

CALLING SEQUENCE:

GETPRO,proc_name

ARGUMENTS (I = input, O = output, [] = optional):

PROC_NAME	I	str	Character string giving the name of the IDL procedure or function. Do not give an extension.

WARNINGS:

User will be unable to obain source code for a native IDL function or procedure, or for a FORTRAN or C routine added with CALL_EXTERNAL. User must have write privilege to the current directory.

EXAMPLE:

To put a copy of the USER library procedure CURVEFIT on the default directory:

```
GETPRO,'CURVEFIT'
```

NAME/ONE LINE DESCRIPTION OF ROUTINE:

GETTOK retrieves a segment of a character string.

DESCRIPTION:

IDL function to retrieve the first part of a character string until the character CHAR is encountered.

CALLING SEQUENCE:

y = GETTOK(st,char)

ARGUMENTS (I = input, O = output, [] = optional):

Y	O	str	output character string which terminates at CHAR.
ST	I	str	input character string
CHAR	I	str	character at which output character string is to be terminated.

WARNINGS:

NONE.

EXAMPLE:

```
ST='ABC=999'
Y=GETTOK(ST,'=')
PRINT,Y
        ABC
```

NAME:

GET_COORDS

PURPOSE:

A general purpose routine that takes user input for angular coordinates and returns floating-point values. The user may input as floating point or sexigesimal. If user inputs RA then it is the calling procedure's job to convert hours to degrees if needed. Since the input string is parsed character-by-character, ANY character that is not a digit, minus sign or decimal point may be used as a delimiter, i.e. acceptable examples of user input are:

```
1:03:55        -10:15:31
1 3 55.0       -10 15 31
1*3 55         -10abcd15efghij31
1.065278       hello    -10.25861
```

CALLING SEQUENCE:

GET_COORDS, Coords, [PromptString, NumVals]

OPTIONAL INPUT:

PromptString - A string to inform the user what data are to be entered

OUTPUT:

Coords - a 2 element floating array containing the coordinates. The vector [-999,-999] is returned if there has been an error.

OPTIONAL OUTPUT:

NumVals - the number of separate values entered by the user: 2 if the user entered the coordinates as floating point numbers, 6 if the user entered the coordinates as sexigesimal numbers. Some calling procedures might find this information useful so output may be in the same format as the user's input.

NAME/ONE LINE DESCRIPTION OF ROUTINE:
GET_DATE returns the current date in DD/MM/YY format.

PURPOSE:
Return the current date in DD/MM/YY format. This is the format required by the DATE and DATE-OBS keywords

CALLING SEQUENCE:
GET_DATE, dte

INPUTS:
None

OUTPUTS:
dte = An eight character scalar string specifying the current day (0-31), current month (1-12), and last two digits of the current year

EXAMPLE:
Add the current date to the DATE keyword in a FITS header,h

```
GET_DATE,dte
sxaddpar,h,'DATE',dte
```

NAME/ONE LINE DESCRIPTION OF ROUTINE:

GET_RESOLUTION returns the Skymap's resolution.

DESCRIPTION:

GET_RESOLUTION is an IDL function which uses the Cobetrieve Data Server to determine a Skymap dataset's resolution.

CALLING SEQUENCE:

resolution = GET_RESOLUTION ('dataset')

ARGUMENTS (I = input, O = output, [] = optional):

DATASET	I string	The name of the Skymap dataset.
RESOLUTION	O long	The resolution number.

WARNINGS:

None.

EXAMPLE:

The following call returns the resolution of a FIRCOADD:[FIRAS.SKYMAP_2]FCS_CCMSP_LL.ED_8934308_9012014 Skymap dataset.

```
resolution = GET_RESOLUTION
('fircoadd:[firas.skymap_2]fcs_ccmsp_ll.ed_8934308_9012014')
```

The following block of statements may be imbedded into a user's IDL procedure.

```
index = 0
index = GET_RESOLUTION ('user_disk:[temp]test.skymap')
if (index ne 0) then begin
   .
   .
   .
endif
```

NAME:

GFIT

PURPOSE:

Fit the equation y=f(x) where:

F(x) = A0*EXP(-z^2/2)
and
z=(x-A1)/A2

If a baseline linear fit is requested then :

F(x) = A0*EXP(-z^2/2) + A3 + A4*x

If a baseline quadratic fit is requested then :

F(x) = A0*EXP(-z^2/2) + A3 + A4*x + A5*x^2

A0 = height of exp, A1 = center of exp, A2 = sigma (the width).
A3 = constant term, A4 = linear term, A5 = quadratic term.
The parameters A0, A1, A2, A3 are estimated and then CURVEFIT is called.

CATEGORY:

fitting

CALLING SEQUENCE:

Result = GFIT(X, Y [,WEIGHT=WEIGHT] [,A] [,SIGMAA] [,baserem=basetype])

INPUTS:

X: The independent variable. X must be a vector.
Y: The dependent variable. Y must have the same number of points as X.

OPTIONAL INPUT PARAMETERS:

WEIGHT: A row vector of weights, the same lengthh as x and y. If no weighting then weight(i)=1.0
BASEREM: If basetype = quadratic then a quadratic fit is made to a baseline. If basetype = linear then a linear fit is made to a baseline.

OUTPUTS:

The fitted function is returned.

OPTIONAL OUTPUT PARAMETERS:

A: The coefficients of the fit. A is either a 3, 5 or 6 element vector as described under PURPOSE.
SIGMAA: A vector of standard deviations for the parameters in A.

NAME/ONE-LINE DESCRIPTION:

GRABFACE grabs a cube face from an unfolded T or sixpack

DESCRIPTION:

This function extracts one cube face from a dataset which is either in skycube "T" format with a 4:3 aspect ratio or sixpack format with a 3:2 aspect ratio. The format of the dataset need not be stated in the call to the function. The input dataset may also be a spectral cube (e.g. FIRAS data). Any numerical data type can be accommodated; the output will always be a face with the same data type as the input.

CALLING SEQUENCE:

outface = GRABFACE (dataset, facenum)

ARGUMENTS: (I=input, O=output, []=optional)

OUTFACE	O	any type	Output one face with same data type as input dataset
DATASET	I	any type	Input 2-D or 3-D dataset in either sixpack or skycube "T" format
FACENUM	I	int	Face number to be extracted

WARNINGS:

None.

EXAMPLE:

If you have a skycube called FIRAS_SIGNAL then the call

```
UIDL> FACE4 = GRABFACE (FIRAS_SIGNAL, 4)
```

will create a square array containing the data values in face 4, the galactic center.

NAME/ONE LINE DESCRIPTION OF ROUTINE:

HEADFITS read a FITS header from a disk FITS file.

DESCRIPTION:

IDL function to read initial block(s) of a FITS format file on disk, forming a string array containing the 80 character header records up to, and including the END record.

CALLING SEQUENCE:

RESULT = HEADFITS(filename)

ARGUMENTS (I = input, O = output, [] = optional):

RESULT	O	str arr	80 character header lines
FILENAME	I	str	name of the disk FITS file to be read

WARNINGS:

None.

EXAMPLE:

To format the header records into the string array HPRINT, from a disk FITS file FILE001.FIT in the default directory:

```
HPRINT = HEADFITS('FILE001.FIT')
```

NAME:

HELIO computes (low-precision) heliocentric coords for the planets.

DESCRIPTION:

Compute (low-precision) Heliocentric coordinates for the planets. Adapted from the book Celestial Basic.

CALLING SEQUENCE:

HELIO, jd, hms, list, hrad, hlong, hlat

ARGUMENTS (i = input, O = output, [] = optional):

JD	I long	Julian day to compute for, long scalar
HMS	I int [arr]	Hours, minutes, seconds array: H, M, S. Universal time.
LIST	I int [arr/scl]	List of planets array. May be a single number. 1=merc, 2=venus,... 9 = pluto.
HRAD	O flt [arr/scl]	Array of Heliocentric radii (A.U).
HLONG	O flt [arr/scl]	Array of Heliocentric (ecliptic) longitudes (degrees).
HLAT	O flt [arr/scl]	Array of Heliocentric latitudes (degrees).

WARNINGS:

None.

EXAMPLE:

To find the heliocentric positions of Jupiter and Saturn on 1-Jan-1989

```
UIDL> JDCNV,1989,1,1,0,JD                    ;Convert to Julian Date
UIDL> HELIO,JD,[0,0,0],[5,6],hrad,hlong,hlat ;Get radius, long, and lat
```

NAME/ONE LINE DESCRIPTION OF ROUTINE:
HOST_TO_IEEE converts IDL var. from IEEE-754 to host machine type.

PURPOSE:
To translate an IDL variable from IEEE-754 representation (as used, for example, in FITS data), into the host machine architecture.

CALLING SEQUENCE:
HOST_TO_IEEE, data, [IDLTYPE = ,]

INPUT-OUTPUT PARAMETERS:
data - any IDL variable, scalar or vector. It will be modified by HOST_TO_IEEE to convert from host to IEEE representation. Byte and string variables are returned by HOST_TO_IEEE unchanged

OPTIONAL KEYWORD INPUTS:
IDLTYPE - scalar integer (1-7) specifying the IDL datatype according to the code given by the SIZE function. This keyword will usually be used when suppying a byte array that needs to be interpreted as another data type (e.g. FLOAT).

EXAMPLE:
Suppose FITARR is a 2880 element byte array to be converted to a FITS record and interpreted a FLOAT data.

IDL> host_to_ieee, FITARR, IDLTYPE = 4

NAME/ONE LINE DESCRIPTION OF ROUTINE:

IEEE_TO_HOST converts IDL var. from host machine type to IEEE-754.

PURPOSE:

To translate an IDL variable in IEEE-754 representation (as used, for example, in FITS data), into the host machine architecture.

CALLING SEQUENCE:

IEEE_TO_HOST, data, [IDLTYPE = ,]

INPUT-OUTPUT PARAMETERS:

data - any IDL variable, scalar or vector. It will be modified by IEEE_TO_HOST to convert from IEEE to host representation. Byte and string variables are returned by IEEE_TO_HOST unchanged

OPTIONAL KEYWORD INPUTS:

IDLTYPE - scalar integer (1-7) specifying the IDL datatype according to the code given by the SIZE function. This keyword is usually when DATA is a byte array to be interpreted as another datatype (e.g. FLOAT).

EXAMPLE:

A 2880 byte array (named FITARR) from a FITS record is to be interpreted as floating and converted to the host representaton:

IDL> IEEE_TO_HOST, bytarr, IDLTYPE = 4

NAME/ONE-LINE DESCRIPTION:

IMSTAT returns useful statistics on an image.

DESCRIPTION:

IMSTAT calculates a number of useful statistical quantities on an input image. It will print out the values of avg(image), stdev(image), median(image), min(image), max(image), npixels(image)

CALLING SEQUENCE:

IMSTAT(data, avg, stdev, median, min, max, npix)

ARGUMENTS: (I=input, O=output, []=optional)

data	I	1-D or 2-D input array
[avg]	O	average value of data
[stdev]	O	standard deviation of data
[median]	O	median of data
[min]	O	minimum of data
[max]	O	maximum of data
[npix]	O	number of pixels in data array

EXAMPLES:

imstat,image

This prints out statistical values to the screen only.

imstat,image,mean,stdev,med,minval,maxval,npix

This will print out the values to the screen and place them in named variables for later use.

NAME/ONE LINE DESCRIPTION OF ROUTINE:

IPINFO reads the PDS/IP FITS header and displays field and frequency info

DESCRIPTION:

IPINFO reads in the FITS headers and displays some useful information on what's in the file. This can be useful when working at the UIDL level with the PDSs/IPs

CALLING SEQUENCE:

ARGUMENTS (I = input, O = output, [] = optional):

param	I/O	type	description
dsname	I	string	filename to read
freq_only	I	switch	if /freq_only set then only the frequency data will be displayed

WARNINGS:

EXAMPLE:

NAME/ONE LINE DESCRIPTION OF ROUTINE

LABELIT is a graph labeling procedure for IDL Version II.

DESCRIPTION:

At user option, labels top, bottom, left and right sides and a top subtitle. Automatically centers each label. Always prints user ID, date, and time at bottom. If called with no keywords, program will print an informative message regarding the calling sequence and exit.

CALLING SEQUENCE:

LABELIT [, MAINLABEL=mainlabel] [, BOTLABEL=botlabel] [, YLABEL=ylabel] [, SECLABEL=seclabel] [, RIGHTLABEL=rightlabel] [,/NAME_TIME]

ARGUMENTS (I = input, O = output, [] = optional)

Mainlabel	[I]	str	Label at top of graph
Botlabel	[I]	str	Label at bottom of graph
Ylabel	[I]	str	Y-axis label (left hand side)
Seclabel	[I]	str	Secondary label at top
Rightlabel	[I]	str	Y-axis label (right hand side)
Name_Time	[I]	key	Indicator to print name and time; default is 1 IF any other keyword is specified. Defaults to 0 if no keywords specified.

EXAMPLES:

1. Label a graph with 'Original Time Series' as the main title, 'Bin Number' on the x-axis, 'Counts' on the y-axis, 'FIRAS Data' as the subtitle, and 'TIME_SERIES.PS_PLOT' on the right hand side of the y-axis:

   LABELIT,Mainlabel = 'ORIGINAL TIME SERIES', Botlabel = 'Bin Number', Ylabel = 'Counts', Seclabel = 'FIRAS Data', Rightlabel = 'TIME_SERIES.PS_PLOT'

2. Label a graph with 'Original Time Series' as the main title, 'Bin Number' on the x-axis, 'Counts' on the y-axis, no subtitle, and 'TIME_SERIES.PS_PLOT' on the right hand side of the y-axis:

   LABELIT,Mainlabel = 'ORIGINAL TIME SERIES', Botlabel = 'Bin Number', Ylabel = 'Counts', Rightlabel = 'TIME_SERIES.PS_PLOT'

3. Label a graph with 'Original Time Series' as the main title, no x-axis label, and 'Counts' on the y-axis

   LABELIT, Mainlabel ='ORIGINAL TIME SERIES', Ylabel ='Counts'

4. Print only user name and current time on bottom of graph.

   LABELIT, /Name_Time   or LABELIT, /N

5. Print informative message regarding calling sequence.

   LABELIT

NAME:

LABEL_DATE

PURPOSE:

A function to label axes with dates.

CATEGORY:

Plotting.

CALLING SEQUENCE:

To set up:

dummy = LABEL_DATE(DATE_FORMAT='string')

Then to use:

PLOT, x, y, XTICKFORMAT='LABEL_DATE'

INPUTS:

No explicit user defined inputs. When called from the plotting routines, the input parameters are (Axis, Index, Value)

KEYWORD PARAMETERS:

DATE_FORMAT = a format string which may contain the following:

%M for month (3 character abbr)
%N for month (2 digit abbr)
%D for day of month,
%Y for 4 digit year.
%Z for last two digits of year.
%% is %.
Other characters are passed directly thru.
For example, '%M %D, %Y' prints DEC 11, 1993
'%M %2Y' yields DEC 93
'%D-%M' yields 11-DEC
'%D/%N/%Y' yields 11/12/1993
'%M!C%Y' yields DEC on the top line, 1993 on the bottom (!C is the new line graphic command).

MONTHS = The names of the months, a twelve element string array.

If omitted, use Jan, Feb, ..., Dec.

OUTPUTS:

The date string which is then plotted.

EXAMPLE:

For example, to plot from Jan 1, 1993, to July 12, 1994:

```
Start_date = julday(1, 1, 1993)
End_date = julday(7, 12, 1994)
Dummy = LABEL_DATE(DATE_FORMAT='%N/%D')  ;Simple mm/dd
x = findgen(end_date+1 - start_date) + start_date ;Time axis
PLOT, x, sqrt(x), XTICKFORMAT = 'LABEL_DATE', XSTYLE=1
```

(Plot with X axis style set to exact.)

NAME/ONE LINE DESCRIPTION OF ROUTINE:

LATCIRC generates a 'latitude' circle

DESCRIPTION:

This IDL function generates a circle a about a given 'polar' point. The circle center is defined by the pole and the boundary by the number of degrees, along a great circular arc, away.

CALLING SEQUENCE:

circ = latcirc(pole=pole,angle=angle,vec_lc,[res=res], [arc_inc=arc_inc],arc_inc_actl)

ARGUMENTS (I = input, O = output, [] = optional):

pole	I	int/flt vec	n element vector where n=1 for pixel entry, n=2 for lon/lat entry, and n=3 for unit vector entry.
angle	I	flt	
[vec_lc]	O	flt arr	circular boundary in unit vector format
[res]	I	int	Pixel resolution
[arc_inc]	I	flt	Arc length increment in degrees (Default: 1 deg)
arc_inc_actl	O	flt	Actual arc length incr
circ	O	int/flt arr	circular boundary in same format as input

WARNINGS:

None.

EXAMPLES:

To draw a circle five degress about the point longitude 30, latitude 25, in two degree increments:

```
circ = latcirc(pole=[30,25],angle=5,arc_inc=2.0)
```

To draw a circle 15 degress about the point centered on DIRBE pixel 1002, in one degree increments, returning the unit vector values in the array uv_circ:

```
circ = latcirc(pole=1002,res=9,angle=15,uv_circ)
```

NAME/ONE LINE DESCRIPTION OF ROUTINE:

LATCUT selects data from a map in a specified latitude range.

DESCRIPTION:

LATCUT takes as input a pair of latitudes that it uses to define a band (or 2 bands) on the sky. If a data array is supplied it returns a list of data from that array that fall in the specified latitude range. If, instead of a data array a resolution number is supplied, LATCUT will return a list of pixel numbers in the specified latitude range. The function does not care what in order the latitudes are specified; it sorts out which is the larger one. The default condition is that absolute value of the latitudes is used so that data from both the northern and southern latitude bands is returned; the /NOABS keyword turns this effect off so that the user can specify a range, e.g., [-5, +15], or a region around one pole.

CALLING SEQUENCE:

```
DATALIST =    LATCUT (lat0, lat1, [data=data], [res=res], $
              [/noabs], [coords=coords],[pixels=pixels])
```

ARGUMENTS (I = input, O = output, [] = optional):

DATALIST	O	arr	List of latitude-selected data (if "data" keyword used, or pixel list (if "res" keyword used).
LAT0	I	flt	Lower or upper latitude limit
LAT1	I	"	Other latitude boundary
[DATA]	[I}	array	Skymap, either in sixpack format or unfolded T.
[RES}	[I}	int	Resolution of skycube; required only if DATA array is not supplied.
[/NOABS]	[I]	keywd	If specified, only the specified latitude range is used; otherwise absolute value is taken so that both northern and southern bands are returned.
[COORDS]	[I]	string	Coordinate system in which to interpret latitudes. Only allowed values are: 'G': galactic (Default) 'E': ecliptic 'Q': equatorial
[PIXELS]	[O]	int arr	Optional output latitude-selected pixel list. Identical to function output if DATA array not specified.

WARNINGS:

1. One and only one of DATA or RES must be specified.

EXAMPLES:

For an array MAP from which we wish to extract all data within |b|<15, we could say

```
output = LATCUT(0,15,data=map)            or
output = LATCUT(-15,15,data=map,/noabs)
```

We could also have added pixels=<variable name> to the call to get the list of pixel numbers corresponding to the data. If we wish to know which pixels fall within 10 deg of the southern ecliptic pole on a DIRBE-resolution skycube, we say

```
pixlist = LATCUT(-80,-90,/noabs,res=9,coords='e')
```

NAME/ONE LINE DESCRIPTION OF ROUTINE:

LEAPCHK determines if a year is a leap year.

PURPOSE:

Determines where year is a leap year

CALLING SEQUENCE:

leap = leapchk(yr)

INPUT:

yr - year to check

OUTPUT:

leap - leap year flag (-1 if leap, 0 if non-leap)

NAME/ONE LINE DESCRIPTION OF ROUTINE:

MAG2FLUX converts magnitude to flux in ergs/s/cm^2/A.

DESCRIPTION:

IDL function that converts from magnitude to flux (ergs/s/cm^2/A). Use FLUX2MAG for converting from flux to magnitude.

CALLING SEQUENCE:

result = MAG2FLUX(mag,[zero_pt])

ARGUMENTS (I = input, O = output, [] = optional):

RESULT	O	flt [arr]	Flux in ergs/s/cm^2/A. (= 10^(-0.4*(mag + zero_pt)))
MAG	I	flt [arr]	Magnitude.
ZERO_PT	[I]	flt	Zero point level of the magnitude. If not specified, default is zero_pt = 21.1 (Code et al. 1976, Ap. J.).

EXAMPLE:

To calculate fluxes for stars of magnitude 1.0, 2.0, and 3.0 using a zero point of 21.0:

```
mag = [1.0,2.0,3.0]
flux = mag2flux(mag,21.0)
```

NAME/ONE-LINE DESCRIPTION:

MARKMAP places labels on images given coordinates in any system

DESCRIPTION:

MARKMAP places labels on an image given a file of coordinates and corresponding labels. The image may be a skycube, face, or a reprojection in any coordinate system. The coordinates given in the file may also be in any coordinate system. Coordinates and labels may also be entered from the keyboard or positions may be marked with the mouse.

CALLING SEQUENCE:

markmap, image, [proj=proj], [pcoords=pcoords], [file=file], [fcoords=fcoords], [facenum=facenum], [win=win], [entry=entry], [color=color], [psym=psym], [charsize=charsize], [charthick=charthick], [labelpos=labelpos], [llx=llx], [lly=lly]

ARGUMENTS: (I=input, O=output, []=optional)

image	I	2-D data array
[proj]	I	Projection type. If data is a skycube or face, proj need not be specified. Reprojections must be 'A', 'M', or 'S' for Aitoff, Mollweide, or Global Sinusoidal. Need not be in capitals.
[pcoords]	I	Projection coordinate system. Must be 'E', 'G', or 'Q' for Ecliptic, Gal, or Equatorial. If the projection type is either 'skycube' or 'face', pcoords is set to 'E'. Need not be in caps.
[file]	I	Name of the file holding the labels and coordinates. File needs to be of one of these two forms: HR MIN SEC DEG DEC_MIN DEC_SEC LABEL, i.e., 7 columns for coords in RA Dec Sexigesimal or LONG LAT LABEL, i.e., 3 columns for coords in long/lat format. Labels cannot have spaces in them if they are read in from a file; labels entered from the keyboard may contain spaces. Accepted values for label coordinates are: All latitudes: (-90, +90) Decimal longitudes: (-360, +360) RA hr: (0, 24) If the program finds coordinates that do not match one of these criterion, this line will not be plotted; the others, however, will be displayed.
[fcoords]	I	File coordinate system. Must be 'E', 'G', or 'Q' for Ecliptic, Gal, or Equatorial. Equatorial coords can be specified as "QD" for Decimal coords or "QS" for Sexigesimal coords. Need not be in caps.
[facenum]	I	If data array is a face, you may specify the facenum in the call.
[win]	I	Window number where the display lives.
[entry]	I	Specifies coordinate and label entry mode. 'file' or 'keyboard' gets the input from a file or keyboard entry; 'cursor' allows you to type the label in and then click the spot on the the image where you want it to appear. If the keyword FILE is set, you DO NOT have to specify that entry='file'.
[color]	I	Color to plot labels with. 0-255
[psym]	I	IDL plot symbol. 1=plus sign (default), 2=asterisk, 3=period, 4=diamond, 5=triangle, 6=square, 7=X

[charsize]	I	Size of the printed labels and symbols. Default=1
[charthick]	I	Thickness of the printed labels. Default=1
[labelpos]	I	The position of the printed label relative to the plot symbol. Possible choices are 'T' for top, 'B' for bottom, 'L'/'R' for left and right. The default is to place the label above the symbol (T).
[llx]	I	X coordinate of the lower left corner of the image. i.e., if the image has a 32 screen pixel border on each side, llx=32. Def=0
[lly]	I	Y coordinate of the lower left corner of the image. Def=0

EXAMPLE:

If you have an Aitoff/Galactic reprojection in a data array called DATA and a file with ecliptic coords, and you would like the labels placed to the right of the points, the call would look like

```
UIDL>  markmap, data, proj='a', pcoords='g', file='myfile' $
UIDL>  fcoords='e', labelpos='r'
```

NAME/ONE LINE DESCRIPTION OF ROUTINE:

MERGE combines two vectors into one, eliminating duplicates.

DESCRIPTION:

This function combines two integer vectors together. Each value occurs only once. The second vector can be empty in which case duplicates in the first vector are eliminated. The output vector is sorted in ascending order.

CALLING SEQUENCE:

vec = merge(vec1,vec2)

ARGUMENTS (I = input, O = output, [] = optional):

vec1	I	integer vector
vec2	I	integer vector
vec	O	integer vector

EXAMPLES:

a = [2,1,5,4], b = [3,2,3,5,7]

c = merge(a,b)

c = [1,2,3,4,5,7]

d = merge(b)

d = [2,3,5,7]

Note: The merged vector will always be sorted in ascending order.
Therefore, DO NOT USE MERGE if order must be preserved.

NAME:

MIN_CURVE_SURF

PURPOSE:

Interpolate a regular or irregularly gridded set of points with a minimum curvature spline surface.

CATEGORY:

Interpolation, Surface Fitting

CALLING SEQUENCE:

Result = min_curve_surf(z [, x, y])

INPUTS:

X, Y, Z = arrays containing the x, y, and z locations of the data points on the surface. Need not necessarily be regularly gridded. For regularly gridded input data, X and Y are not used, the grid spacing is specified via the XGRID or XVALUES, and YGRID or YVALUES, keywords, and Z must be a two dimensional array. For irregular grids, all three parameters must be present and have the same number of elements.

KEYWORD PARAMETERS:

Input grid description:

REGULAR = if set, the Z parameter is a two dimensional array, of dimensions (N,M), containing measurements over a regular grid. If any of XGRID, YGRID, XVALUES, YVALUES are specified, REGULAR is implied. REGULAR is also implied if there is only one parameter, Z. If REGULAR is set, and no grid (_VALUE or _GRID) specifications are present, the respective grid is set to (0, 1, 2, ...).

XGRID = contains a two element array, [xstart, xspacing], defining the input grid in the X direction. Do not specify both XGRID and XVALUES.

XVALUES = if present, Xvalues(i) contains the X location of Z(i,j). Xvalues must be dimensioned with N elements.

YGRID, YVALUES = same meaning as XGRID, XVALUES except for the Y locations of the input points.

Output grid description:

GS = spacing of output grid.
If present, GS a two-element vector [XS, YS], where XS is the horizontal spacing between grid points and YS is the vertical spacing. The default is based on the extents of X and Y. If the grid starts at X value Xmin and ends at Xmax, then the default horizontal spacing is (Xmax - Xmin)/(NX-1). YS is computed in the same way. The default grid size, if neither NX or NY are specified, is 26 by 26.

BOUNDS = a four element array containing the grid limits in X and Y of the output grid: [Xmin, Ymin, Xmax, Ymax].
If not specified, the grid limits are set to the extend of X and Y.

NX = Output grid size in the X direction. Default = 26, need not be specified if the size can be inferred by GS and BOUNDS.

NY = Output grid size in the Y direction. See NX.

XOUT = a vector containing the output grid X values. If thisparameter is supplied, GS, BOUNDS, and NX are ignored for the X output grid. Use this parameter to specify irregular spaced output grids.

YOUT = a vector containing the output grid in the Y direction.
If this parameter is supplied, GS, BOUNDS, and NY are ignored for the Y output grid.

XPOUT, YPOUT = arrays containing X and Y values for the output points. With these keywords, the output grid need not be regular, and all other output grid parameters are ignored. XPOUT and YPOUT must have the same number of points, which is also the number of points returned in the result.

OUTPUTS:

A two dimensional floating point array containing the interpolated surface, sampled at the grid points.

EXAMPLE:

IRREGULARLY GRIDDED CASES

Make a random set of points that lie on a gaussian:

```
n = 15          ;# random points
x = RANDOMU(seed, n)
y = RANDOMU(seed, n)
z = exp(-2 * ((x-.5)^2 + (y-.5)^2))  ;The gaussian
```

get a 26 by 26 grid over the rectangle bounding x and y:

```
r = min_curve_surf(z, x, y)       ;Get the surface.
```

Or: get a surface over the unit square, with spacing of 0.05:

```
r = min_curve_surf(z, x, y, GS=[0.05, 0.05], BOUNDS=[0,0,1,1])
```

Or: get a 10 by 10 surface over the rectangle bounding x and y:

```
r = min_curve_surf(z, x, y, NX=10, NY=10)
```

REGULARLY GRIDDED CASES

```
z = randomu(seed, 5, 6)             ;Make some random data
```

interpolate to a 26 x 26 grid:

```
CONTOUR, min_curve_surf(z, /REGULAR)
```

NAME/ONE LINE DESCRIPTION OF ROUTINE:

MKBGRMOD is a general 2-D background modeling tool.

DESCRIPTION:

IDL function to generate a 2-D background model from a rasterized map. The type of background model returned is determined by the combination of input keyword parameters. Some possibilities include:

Model Type	Keywords/Values
Quintic Interpolation To Pixels Specified Interactively Using the Mouse.	None
As Above But Including An nth-Order Polynomial Surface Fit to the Interpolated Background.	Fit=n, where n>0.
Lower Envelope Determined Using An NxN Spatial Minimum Filter.	Fsize=N, where N>1.
As Above But Including An nth-Order Polynomial Surface Fit to the Lower Envelope.	Fsize=N, where N>1, Fit=n, where n>0.
Median Filtered Using An NxN Spatial Median Filter.	Fsize=N, where N>1, Filter='median'.
Iterative Surface Fitting and Source Subtraction. At each iteration, the routine first calculates an intermediate background model using a surface fit. The background model is then subtracted from the original map, and the standard deviation of the residuals is determined. All pixels greater than M standard deviations above the background are located, a SRCWIDTH x SRCWIDTH patch centered on each deviant pixel is sentinelized, and the next iteration begins. Iteration stops when no more sources are found.	Cut=M, where M>0, Fit=n, where n>0.
As Above But Surface Fit is Done to the NxN Lower Envelope.	Cut=M, where M>0, Fit=n, where n>0, Fsize=N, where N>1.
As Above But Using An Averaging Filter With No Surface Fitting.	Cut=M, where M>0, Fsize=N, where N>1, Filter='avg'.

CALLING SEQUENCE:

bgr = MKBGRMOD(map,dsrcmap,fit=fit,cut=cut,fsize=fsize,
filter=filter,srcwidth=srcwidth,/nocorn,/noplot, badval=badval)

ARGUMENTS (I = input, O = output, [] = optional):

map	I	arr	2-D array containing the input map
fit	[I]	scl	Order of polynomial surface fit (per dimension)
fsize	[I]	scl	Size of Spatial Filter in Pixels
filter	[I]	str	Name of filter function. Default='min' (i.e., the lower envelope), but any function which takes an array as input and returns a scalar is valid.
cut	[I]	scl	Number of Standard Deviations Used to ID a Source
srcwidth	[I]	scl	Size of pixel patch to be removed when a source is found. (Default=3, the central pixel and its nearest neighbors)
badval	[I]	scl	All data at this value will be ignored. (Default=0.0)
nocorn	[I]	scl	When selecting background points with the mouse, MKBGRMOD automatically includes the map corners. Specifying /nocorn will supress this.
noplot	[I]	scl	By default, MKBGRMOD will plot the original map and the background model. Specifying /NOPLOT will suppress the plotting. The option is not valid for interactive modeling.
bgr	O	arr	2-D Array containing the final background model.
dsrcmap	[O]	arr	If the CUT option is specified, DSRCMAP will contain the final source subtracted map.

WARNINGS:

Plotting requires a TV terminal, and the input map must be smaller than 512x512. The iterative procedure may become unstable, particularly with high-order surface fits or low sigma cuts. Specifying the interpolated model with /NOCORN and a surface fit is not recommended. The surface fit requires that all pixels are sampled, but without the corners MKBGRMOD may have to extrapolate to fill the edges.

EXAMPLES:

1. To specify background points with the mouse:

    bgr = MKBGRMOD(map)

The input map will be displayed on th screen. Press the left mouse button to select points. Select the last point with the right mouse button.

2. To generate a 2nd-Order polynomial fit to the 4x4 lower envelope while ignoring any data equal to -16000:

    bgr = MKBGRMOD(map,fsize=4,fit=2,badval=-16000)

3. To generate an iteratively source subtracted model which uses a 4th-order polynomial surface fit to the 3x3 lower envelope and a source cut of 5 sigma:

    bgr = MKBGRMOD(map,dsrcmap,cut=5,fit=4,fsize=3)

NAME/ONE LINE DESCRIPTION OF ROUTINE:

MKHDR makes a FITS image header with required keywords.

PURPOSE:

Make a FITS image header with required keywords. If an image array is supplied, then the header will be appropiate to that array. Otherwise, the user can specify the dimensions and datatype.

CALLING SEQUENCE:

```
MKHDR,HEADER                    ;Prompt for image size and type
MKHDR,HEADER,IM                 ;Make header appropriate to image IM
MKHDR,HEADER,TYPE,NAXISX        ;Specify values of required keywords
```

OPTIONAL INPUTS:

IM - If IM is a vector or array then the header will be made appropiate to the size and type of IM. (IM does not have to be the actual data; it can be a dummy array of the same type and size as the data.)

TYPE - If more than 2 parameters are supplied, then the second parameter is intepreted as an integer giving the IDL datatype e.g. 1 - LOGICAL*1, 2 - INTEGER*2, 4 - REAL*4, 3 - INTEGER*4

NAXISX - Vector giving the size of each dimension (NAXIS1, NAXIS2, etc.).

OUTPUT:

HDR - image header, (string array) with required keywords BITPIX, NAXIS, NAXIS1, ... Use PUTAST to add astrometry keywords to header

RESTRICTIONS:

(1) Group format is not allowed - the keyword SIMPLE is always 'T'
(2) Any data already in the header parameter before calling MKHDR will be destroyed.

EXAMPLE:

Create a minimal FITS header, HDR, for a 30 x 40 x 50 INTEGER*2 array

```
MKHDR,HDR,2,[30,40,50]
```

Alternatively, if the array already exists as an IDL variable, ARRAY,

```
MKHDR,HDR,ARRAY
```

NAME/ONE LINE DESCRIPTION OF ROUTINE:

MONTHDAYS gives number of days in a month, given year and month.

PURPOSE:

Gives number of days in a month given year and month.

CALLING SEQUENCE:

DAYS = MONTHDAYS(YR,MON)

INPUTS:

YR = year, MON = month number (jan = 1).
If MON is 0 then return month days for all of year as array.

OUTPUTS:

DAYS = number of days in month (like 31).
Or array for MON = 0.

EXAMPLES:

DAYS = MONTHDAYS(84,2)
Returns the value 29 in the variable DAYS.

COMMON BLOCKS:

None

RESTRICTIONS:

Only for 1901 to 2099.

NAME/ONE LINE DESCRIPTION OF ROUTINE:

MOONPOS returns the apparent RA and Declination of the Moon.

PURPOSE:

To compute the apparent Right Ascension and Declination of the Moon at a given date or a given array of dates.

CALLING SEQUENCE:

MOONPOS, Dte, Ra, Dec

ARGUMENTS (I = input, O = output, [] = optional):

DTE	I	flt [arr]	Modified Julian Date: i.e. Julian Date - 2400000.5
RA	O	dbl [arr]	Right Ascension. Units: radians
DEC	O	dbl [arr]	Declination. Units: radians.

WARNINGS:

1. The Julian Date must be input as the modified Julian Date, given by JD-2400000.5. False output obtained in any other format.
2. The outputs are not of high accuracy. The Right Ascension is will rarely be in error by more than 0.3 degrees, and the declination will rarely be in error by more than 0.2 degrees.
3. The RA and Dec are returned for the epoch of date, not 1950.0 or 2000.0.

EXAMPLE:

To find the RA and DEC of the Moon on JD 2447891.5 (0h UT, Jan 0 1990):

```
MJD=47891.0
MOONPOS, MJD, RA, DEC              ; outputs RA and DEC in radians.
print, ra*180/!pi/15 & dec*180/!pi  ; get RA in hours, DEC in deg.
        21.058254        -17.208063
```

NAME/ONE-LINE DESCRIPTION:

MULTIPOLE subtracts dipole + quadrupole terms from a scalar sky map.

DESCRIPTION:

An input intensity map (with optional weights) is the basis of a least-squares fit for the position, monopole value, dipole amplitude, quadrupole amplitude and quadrupole coefficients of the CMBR. The routine returns these values as well as a residual map with the dipole plus quadrupole subtracted. The user can also specify a range of galactic latitude to exclude from the fit. The residual is subtracted from the whole map. A pure quadrupole map is also created. The routine assumes that the chi-square per degree of freedom of the fit is 1 and computes errors on the dipole amplitude and direction.

CALLING SEQUENCE:

MULTIPOLE, inmap, monopole, monopole_sig diampl, diampl_sig, diglon, diglat,$
digl_sig, [galexc=...], [residmap=residmap], [weights=weights],$
[badval=badval],[qparms=qparms],[qsigma=qsigma],[quadmap=quadmap]

ARGUMENTS (I=input, O=output, []=optional):

inmap	I	flt arr	2-D input skymap, in right-T format
monopole	O	dble	Mean CMBR intensity
monopole_sig	O	dble	Sigma of CMBR intensity
diampl	O	dlbe	Dipole amplitude
diampl_sig	O	dlbe	Sigma of dipole amplitude
diglon	O	dlbe	Galactic longitude of dipole pole (deg)
diglat	O	dlbe	Galactic latitude of dipole pole (deg)
digl_sig	O	dlbe	Sigma of dipole direction (deg)
[galexc]	[I]	flt	Absolute value of minimum galactic latitude to include in fit (deg), default=0.
[residmap]	[O]	flt arr	2-D output residual map
[weights]	[I]	flt arr	2-D map of weights corresponding to input map, default=uniform wts
[badval]	[I]	flt	Flag value for bad pixels, default=0.
[qparms]	[O]	dlbe	Quadrupole coefficients
[qsigma]	[O]	dlbe	Sigma's of quadrupole coefficients
[quadmap]	[O]	dlbe	2-D output of quadrupole map

WARNINGS:

1. Only a full sky right-T can be used as input. Pixelization is assumed to be in ecliptic coordinates!
2. It is generally wise to specify a galactic latitude exclusion of 20 degrees or more for believable results.
3. Flagged pixels will show up as zeros in the residual map.

EXAMPLE:

If "inmap" is a map of temperatures in which empty pixels have a value of zero then a simple call for an unweighted fit would be

```
UIDL> multipole,inmap,avtemp,tsig,diamp,ampsig,lon,lat,dir_sig,$
UIDL> galexc=20,qparms=qparms
UIDL> print,avtemp,tsig,diamp,ampsig,lon,lat,dir_sig,qparms
        2.7256  0.000005  0.003345  0.000016  265.60  48.34  0.24
```

-0.0368 0.0254 -0.0266 -0.0196 0.0120 0.0278

If, e.g., "residmap=residual" had also been included in the call then the variable "residual" would contain a residual skymap after subtracting a 2.7256 K average plus a 3.345 mK dipole plus a quadrupole from inmap.

If in addition to inmap we have a second map, e.g. "sigmas" that contains the weight at each pixel, then we could also have included ", weights=sigmas" in the call.

COMMON BLOCKS:

None

PROCEDURE:

There is a fair amount of coordinate conversion done, so the first thing that is done is to determine the resolution of the skymap from its dimensions. The procedure is:

(1) Determine map resolution
(2) "De-rasterize" the data into a 1-D pixel list
(3) Cull pixels with flag values from the list
(4) Cull low galactic latitude pixels from the list
(5) Convert pixel numbers of remaining pixels to unit vectors
(6) Using unit vectors, do a SVD fit to the dipole +quadrupole in Cartesian coordinates, i.e. Model = T0 + Td.u + Tq.uq, where the second term is the dot product of the dipole (of amplitude Td) and the unit vector u and the third term is the dot product of the quadrupole (of amplitude Tq) and the Legendre P2 unit vector u.
(7) Compute sigma's using output from SVD matrices.
(8) Convert the best-fit unit vector into galactic coords (for dipole)
(9) Compute the sigma for the direction by recomputing the unit vector for the dipole + sigmas and for the dipole - sigmas.
(10) Create a dipole + quadrupole map and subtract from the input map if a residual map is desired
(11) Create a quadrupole model map if desired.

LIBRARY CALLS:

coorconv
pix2xy
xy2pix

NAME/ONE LINE DESCRIPTION OF ROUTINE:

NICE_HISTO returns a histogram and corresponding x-vector.

DESCRIPTION:

Nice_histo is designed to make creation and plotting of histograms easier. The program takes an array of data, scales it between -1 and 1, forms a histogram with that data with the (optional) desired number of bins, and also returns a scale vector of the same length as the histogram running between the minimum and maximum values in the original array. The number of bins defaults to 200 if not specified. The user may then easily plot the histogram with the true scaling of the array displayed.

CALLING SEQUENCE:

NICE_HISTO, dataset, return_histo [, scale] [,num_bins]

ARGUMENTS: (I = input, O = output, [] = optional)

DATASET	I	flt or dbl arr	The vector containing the information to be put into histogram form.
NUM_BINS	[I]	int	Number of bins to use in the histogram (defaults to 200).
RETURN_HISTO	O	int [arr]	The returned histogram
SCALE	[O]	dbl [arr]	The returned scaling vector

WARNINGS:

None.

EXAMPLES:

1. To produce a histogram having 200 bins from vector MYDATA, returning the histogram vector MY_HISTODATA and scaling vector SCALE_VEC:

```
NICE_HISTO,MYDATA,MY_HISTODATA,SCALE_VEC
```

2. To produce a histogram having 500 bins from vector MYDATA, returning the histogram vector MY_HISTODATA and scaling vector SCALE_VEC:

```
NICE_HISTO,MYDATA,MY_HISTODATA,SCALE_VEC,500
```

In both the above the examples, the user may then plot the histogram using the scaling vector to show the binned values using the command

```
PLOT, SCALE_VEC, MY_HISTODATA
```

NAME/ONE LINE DESCRIPTION OF ROUTINE:

NYQUIST returns the Nyquist frequency of a FIRAS spectrum

DESCRIPTION:

IDL function to return the Nyquist frequency for a specified FIRAS instrument state in either wavenumbers or hertz. This routine incorporates the frequency shift due to the on-orbit sampling rate of the instrument.

CALLING SEQUENCE:

FNYQ = NYQUIST (fakeit, ngroup, mtm_speed, iunits)

ARGUMENTS (I = input, O = output, [] = optional):

FNYQ	O	flt	Nyquist frequency for the specified instrument state Units: Icm if IUNITS = 0 Hz if IUNITS = 1
FAKEIT	I	int	MTM scanning/fakeit mode FAKEIT = 0 indicates scanning mode FAKEIT = 1 indicates fakeit mode
NGROUP	I	int	Adds per group for the spectrum. NGROUP has a range of 1-12
MTM_SPEED	I	int	MTM scan speed MTM_SPEED = 0 indicates slow speed MTM_SPEED = 1 indicates fast speed
IUNITS	I	int	Frequency units flag IUNITS = 0 indicates wavenumbers IUNITS = 1 indicates hertz

WARNINGS:

1. If given an "impossible" instrument state (e.g., MTM speed neither fast nor slow), the routine returns a value of -1.
2. Fakeit mode data do not have a meaningful conversion to wavenumbers, so hz must be specified (IUNITS = 1) when FAKEIT=1.

EXAMPLE:

To determine the Nyquist frequency for the LLSS scan mode in icm:

```
fakeit    = 0
ngroup    = 3
mtm_speed = 0
iunits    = 0
fnyq = nyquist (fakeit, ngroup, mtm_speed, iunits)
```

Alternatively, one can invoke the function as follows:

```
fnyq = (0, 3, 0, 0)
```

In either case, NYQUIST returns a value of 145.1103 icm to the variable fnyq.

NAME/ONE LINE DESCRIPTION OF ROUTINE:

PIX2DAT extracts data for a given set of raster coordinates.

DESCRIPTION:

This IDL function creates a data array given either a list of pixels or a set of x and y raster coordinates and a raster image (sky cube or face). The skycube can be in either unfolded or sixpack formate. This routine is the "complement" to PIX2XY. The program assumes a right oriented, ecliptic coordinate input raster.

CALLING SEQUENCE:

```
data =  pix2dat(pixel=pixel,raster=raster) OR
data =  pix2dat(x_in=x_in,y_in=y_in,raster=raster)
```

ARGUMENTS (I = input, O = output, [] = optional):

pixel	I	int/long arr	Input pixel list
x_in	I	int/long arr	x coord list
y_in	I	int/long arr	y coord list
raster	I	int/flt/dbl arr	input raster array
data	O	int/flt/dbl arr	output data array
[no_c]	I	qualifier	Suppress c-module usage switch

Either the pixel keyword or the x_in AND y_in keywords must be specified.

The x and y coordinates for a single face are those within the face, not those for the face positioned within a unfolded sky cube. For example, for a FIRAS face the coordinates range from 0 to 31.

WARNINGS:

None.

EXAMPLE:

1. To extract the data from a raster for all pixels with galactic latitude less than 30 degrees:

```
pixel = indgen(6144)

ll = coorconv(pixel,infmt='p',inco='f',outfmt='l',outco='g')

pix30 = pixel(where(abs(ll(*,1)) lt 30.0))

dat30 = pix2dat(pixel=pix30,raster=sky_cube)
```

Alternatively, we can use x and y coordinate arrays generated with pix2xy:

```
pix2xy,pix30,x30,y30,res=6

dat30 = pix2dat(x_in=x30,y_in=y30,raster=sky_cube)
```

2. To extract from a face:

```
face4 = sky_cube(0:31,32:63)              ; extract face 4 from sky cube

fij = pix2fij(pix30,6)                    ; get face and coordinates

pix4 = pix30(where(fij(*,0) eq 4))        ; extract face 4 pixels

dat4 = pix2dat(pixel=pix4,raster=face4)

pix2dat,pix4,data=dat4,raster=face4_30,res=6,face=4
```

3. To extract along a galactic meridian of a DIRBE map:

```
ll = fltarr(361,2)                 ; allocate long-lat array

ll(*,0) = 0.                       ; chose central meridian

ll(*,1) = 90 - findgen(361)*0.5    ; generate points along meridian at half degree
                                   intervals.

pix = coorconv(ll,infmt='l',inco='g',outfmt='p',outco='b') ; get pixels along meridian

dat = pix2dat(pixel=pix,raster=band_01)
```

NAME/ONE LINE DESCRIPTION OF ROUTINE:

PIX2XY creates a sky cube or face from a pixel/data list.

DESCRIPTION:

This IDL procedure creates a raster image (sky cube or face) given a pixel list and data array (provided, for example, by READ_SKYMAP). The data array can be either a vector or two-dimensional array. In the latter case, the data for each raster image can be stored in either the columns or rows. The procedure also returns the x and y raster coordinates of each pixel. The rasterized images can be viewed with the IDL TV or TVSCL commands. Only right oriented, ecliptic coordinate images are built.

CALLING SEQUENCE:

PIX2XY,pixel,x_out,y_out,res=res,[data=data],[raster=raster], [/face],[/sixpack]

ARGUMENTS (I = input, O = output, [] = optional):

pixel	I	int/long arr	Input pixel list
[x_out]	O	int arr	x-coordinate array
[y_out]	O	int arr	y-coordinate array
res	I	int	cube resolution
[data]	I	int/flt/dbl arr	input data array
[raster]	O	int/flt/dbl arr	rasterized output array
[face]	I	qualifier	Face output qualifier
[sixpack]	I	qualifier	Sixpack output qualifier

WARNINGS:

None.

EXAMPLE:

1. To create x and y coordinate arrays from a FIRAS (res 6) pixel list without returning a rasterized image (since no data is provided):

```
pix2xy,firas_pixel,x_out,y_out,res=6
```

2. To create a rasterized DIRBE (res 9) sixpack from a pixel and data list without returning the coordinate (x,y) arrays:

```
pix2xy,dirbe_pixel,data=dirbe_data,res=9,raster=image,/sixpack
```

3. To create a rasterized DIRBE face from a pixel and data list:

```
pix2xy,dirbe_pixel,data=dirbe_data,res=9,raster=image,/face
```

NAME/ONE-LINE DESCRIPTION:

PIXINFO returns lon/lat, x/y, pixel number, and data value

DESCRIPTION:

PIXINFO allows the user to choose different points on an image with the mouse and find out the points' x/y coordinates, longitude/latitude in any coordinate system, pixel number, and data value. Longitudes and latitude are returned in an (n,2) array. X/Y coordinates, pixel number, and data value may also be returned by using the "xout", "yout", "pixels", and "datval" keywords. Users have the option of removing the marks (refreshing the image).

CALLING SEQUENCE:

info = PIXINFO (pixinfo, data, outcoord=outcoord, win=win, proj=proj, pcoords=pcoords, $
face=face, wpos=wpos, pixels=pixels, xout=xout, yout=yout, datval=datval)

ARGUMENTS: (I=input, O=output, []=optional)

data	I	2-D or 3-D data array
[outcoords]	I	Coordinates that the longitudes and latitudes will be given in.
[win]	I	Window number where the display lives.
[proj]	I	Projection type if data array is a reprojection. Use proj='a' for Aitoff, proj='s' for Global Sinusoidal, proj='m' for Mollweide
[pcoords]	I	Coordinates of the projection. If data array is a skycube or face, pcoords is assumed to be ecliptic. For reprojections, use pcoords='g' for galactic coords, pcoords='e' for ecliptic, pcoords='q' for equatorial
[face]	I	If data array is a face, you may specify the face number in the call.
[wpos]	I	2-element array indicating Window POSitioning of the image. Use form wpos=[x,y] where x is the X coordinate of the lower left corner of the image and y is the Y coordinate of the lower left corner of the image. i.e., if the image has a 32 screen pixel border on each side, wpos=[32,32]. Default=[0,0]
[pixels]	O	Pixel list of the pixels selected.
[xout]	O	List of X coordinates of selected points.
[yout]	O	List of Y coordinates of selected points.
[datval]	O	List of data values of selected points.

EXAMPLE:

To return the galactic longitude/latitude of some points on a skycube (data array=CUBE) displayed in IDL window 1, type

```
UIDL>  lons_lats=PIXINFO (cube, outcoord='g', win=1)
```

Click left mouse button to mark the points and right mouse button to quit. The galactic coordinates are held in LONS_LATS.

NAME/ONE LINE DESCRIPTION OF ROUTINE:

PLANCK returns the spectral radiance of a blackbody.

DESCRIPTION:

IDL function to return the spectral radiance of a blackbody, i.e. the Planck curve, in units of either MJy/steradian (I_nu) or watts/cm^2/steradian (nu_I_nu). The blackbody temperature and either frequency (in icm or GHz) or wavelength (in microns) are inputs to the function. The routine also optionally returns the derivative with respect to temperature, in units of MJy/sr/K or W/cm^2/sr/K.

CALLING SEQUENCE:

RESULT = PLANCK (temperature, nu_or_lambda [,dBdT] [,UNITS=units], [/MJY], [/WCM2])

ARGUMENTS (I = input, O = output, [] = optional):

RESULT	O	flt [arr]	Spectral radiance at each wavelength. Units: W/cm^2/sr/K if /WCM2 specified, MJy/sr if /MJY specfied
TEMPERATURE	I	flt	Temperature of blackbody, in K.
NU_OR_LAMBDA	I	flt	Frequency or wavelength at which to calculate spectrum. Units are as specified with UNITS keyword.
dBdT	[O]	flt [arr]	Derivative of Planck with respect to temperature.
UNITS	[I]	str	'Microns', 'icm', or 'GHz' to identify units of NU_OR_LAMBDA. Only first character is required. If left out, default is 'microns'.
/MJY	I	key	Sets output units to MJy/sr
/WCM2	I	key	Sets output units to W/cm^2/sr

WARNINGS:

1. One of /MJY or /WCM2 MUST be specified.
2. Routine gives incorrect results for T < 1 microKelvin and wavelengths shortward of 1.e-10 microns. (So sue me).

EXAMPLE:

To produce a 35 K spectrum in MJy/sr at 2, 4, 6, 8, 10 microns:

```
wavelength = 2. + 2.*findgen(5)
temp = 35.
blackbody = planck(temp, wavelength, units='micron', /mjy)
```

One could also get back the derivative by including it in the call:

```
blackbody = planck(temp, wavelength, deriv, units='m', /mjy)
```

NAME/ONE LINE DESCRIPTION OF ROUTINE:

PLTIME adds a date and time to a pre-existing plot.

DESCRIPTION:

Adds a date and time to a pre-existing plot. Date is written vertically in the lower right hand corner. PLTIME should be called after the main plotting commands.

CALLING SEQUENCE:

PLTIME

ARGUMENTS:

None.

WARNINGS:

PLTIME works best with high resolution devices (e.g. laser printers).

EXAMPLES:

Add a date and time to a plot of the integers:

```
Plot,findgen(20)
PLTIME
```

NAME:

PRECESS

PURPOSE:

Precess coordinates from EQUINOX1 to EQUINOX2. For interactive display, one can use the procedure ASTRO which calls PRECESS.

CALLING SEQUENCE:

PRECESS, ra, dec, [equinox1, equinox2, PRINT =]

INPUT - OUTPUT:

RA - Input right ascension in DEGREES (scalar or vector)
DEC - Input declination in DEGREES (scalar or vector)
NOTE: The input RA and DEC are modified by PRECESS to give the values after precession.

OPTIONAL INPUTS:

EQUINOX1 - Original equinox of coordinates, numeric scalar. If omitted, PRECESS will query for EQUINOX1 and EQUINOX2.
EQUINOX2 - Equinox of precessed coordinates.

OPTIONAL INPUT KEYWORDS:

PRINT - If this keyword is set and non-zero, then the precessed coordinates are displayed at the terminal.

RESTRICTIONS:

Accuracy of precession decreases for declination values near 90 degrees. PRECESS should not be used more than 2.5 centures from 1900.

EXAMPLE:

Precess the 1950 coordinates of Eps Ind (RA = 21h 59m,33.053s, DEC = (-56d, 59', 33.053") to equinox 1975.

```
IDL> precess, ten(21,59,33.053)*15., ten(-56,59,33.053),1950,1975,/print
```

NAME/ONE LINE DESCRIPTION OF ROUTINE:

PROJBLD generates projection lookup tables.

DESCRIPTION:

This IDL procedure generates the projection lookup tables that are used by the USRPROJ facility to create and display reprojections of the native skycube. The user enters half the dimension of the vertical component of the projection array (all such arrays have a 2:1 aspect ratio), the projection type (Aitoff, Global Sinusoidal, Mollweide), the reference coordinate system (Ecliptic, Galactic, Equatorial), and the set of Euler angles defining the user coordinate system. The procedure returns the three lookup tables (in unfolded skycube coordinates) to be used by the USRPROJ facility.

CALLING SEQUENCE:

pro projbld, n,proj=proj,coord=coord,[euler=euler],ilut,jlut,mask

ARGUMENTS (I = input, O = output, [] = optional):

n	I	int	half dimension of vertical components of proj arrays
proj	I	string	Projection Type: ('A','S','M')
coord	I	string	Reference Coordinate System: ('E','G','Q')
[euler]	I	flt arr	Euler angle vector (3 elem) Default: [0,0,0]
[ilut]	O	int arr	I*2 array containing skycube column numbers for reproj. Dimensions: (4*n,2*n)
[jlut]	O	int arr	I*2 array containing skycube row numbers for reproj. Dimensions: (4*n,2*n)
[mask]	O	byte arr	Face number / Boundary mask (face # within boundary of projection, 255 outside boundary.)

WARNINGS:

1. To generate a 512 by 256 set of lookup tables takes about 90 sec of CPU time and 1283 blocks of storage (IDL save set).

2. A 1024 by 512 set takes about 380 CPU sec and 5123 blocks.

EXAMPLES:

The Euler angles are defined in terms of a left-handed coordinate system with positive angles specifying clockwise rotations of the coordinate axes. The first element gives the rotation about the z-axis of the reference coordinate system, the second, the rotation about the new y-axis and the third, the rotation about the z-axis of the current system.

1. To generate a 256 by 128 Aitoff galactic-centered projection:

```
projbld,64,proj='a',coord='g',ia,ja,ma
```

2. To generate a 512 by 256 "anti-ecliptic" Mollweide projection:

```
projbld,128,proj='m',coord='e',euler=[180,0,0],ae_i,ae_j,ae_mask
```

3. To generate a 512 by 256 "CMB dipole (l = 267.7, b = 47)" Global Sinusoidal projection:

```
projbld,128,proj='s',coord='g',euler=[267.7,43,0],dp_i,dp_j, dp_mask
```

4. To generate a 1024 by 512 "Leo-centered (RA 10h30m dec 20)" Aitoff projection:

```
projbld,128,proj='a',coord='q',euler=[157.5,-20,0],leo_i,leo_j, leo_mask
```

NAME/ONE LINE DESCRIPTION OF ROUTINE:

PROJGRID overlays coordinate grids on projections.

DESCRIPTION:

This IDL procedure generates a coordinate grid that is overlaid on a currently displayed image. The user must provide the projection, which may be an unfolded skycube or an Aitoff, Global Sinusoidal, or Mollweide projection, in an existing window. In the latter case, the image array can be of any size but it must have a 2:1 aspect ratio. In addition, the grid coordinate system does not have to be the same as the projection coordinate system.

CALLING SEQUENCE:

pro projgrid,image,[proj=proj_type],[coord=coord_type], [gcoord=gcoord],[w_pos=w_pos], [merd=merd],[para=para],[color=color]

ARGUMENTS (I = input, O = output, [] = optional):

image	I	flt arr	Image
[proj]	I	string	Projection Type: ('A','S','M')
[coord]	I	string	Projection Coordinate System: ('E','G','Q')
[gcoord]	I	string	Grid Coordinate System: ('E','G','Q')
[w_pos]	I	int arr	Offset of image in window (Default: [0,0])
[merd]	I	flt arr	Meridian (longitude) array (Default: [0,60,120,180]) (merd = -1 if none)
[para]	I	flt arr	Parallel (latitude) array (Default: [0,30,60]) (para = -1 if none)
[color]	I	int	Grid color (0-255)

Note: The meridian and parallel entries are "symmetrized", i.e., the default parallels are drawn at -60, -30, 0, 30, 60.

WARNINGS:

The user must set the window (with the WSET command) before running PROJGRID.

EXAMPLE:

1. To display an equatorial coordinate overlay on a skycube called, 'cube', with no offset in the display window:

```
projgrid,cube,gcoord='q'
```

In this case it is not necessary to give the projection type as the program can recognize a skycube by its 4:3 aspect ratio.

2. To display an ecliptic coordinate overlay with the default meridians (0,60,120,180) and parallels (0,30,60) on an Galactic Aitoff reprojection, 'ait_img', offset within the window by [32,32].

```
projgrid, ait_img,proj='a',coord='g',gcoord='e',w_pos=[32,32]
```

3. To display a galactic coordinate overlay with the meridians at (0,45,90,135,180) and no parallels on a Equatorial Mollweide reprojection, 'mol_img', with zero window offset and grid color 28.

```
projgrid, mol_img,proj='m',coord='q',gcoord='g',
```

```
merd=[0,45,90,135,180),para=-1,color=128
```

If the 'proj', 'coord', or 'gcoord' keywords are not provided, the user is prompted for them.

NAME/ONE LINE DESCRIPTION OF ROUTINE:

PSIMAGE sets optimum size and displays (tvscl) image in postscript

DESCRIPTION:

Sets up an optimised display size for post_script output, proportional to the ratio of the x/y axes of the image; then displays the input array in this window, using the IDL display procedure 'tvscl'. The 'optimised' window selects the largest possible display size for letter size post-script printout, keeping the x/y ratio of the image the same as the original. The image is rotated if y is larger than x, before display. 'tvscl', which is used to display the image, defaults to the maxima and the minima of the image, unless otherwise specified by the user. Optional output file name can be put in if desired filename is other than 'idl.ps'. An optional scaling parameter can be used if the output should be less than full letter size. The output file can be sent directly to a postscript printer.

CALLING SEQUENCE:

```
PSIMAGE, image [,min] [,max] [,OUTPUT=output] [,SCALE=scale]
        [,COLOR=color] [,/HSV] [,/HLS] [,/SEIKO]
```

ARGUMENTS (I = input, O = output, [] = optional):

IMAGE	I	flt [arr]	Name of input array to be displayed.
MIN	[I]	flt	User specified minima for tvscl display; Default is minima of array.
MAX	[I]	flt	User specified maxima for tvscl display; Default is maxima of array.
OUTPUT	[O]	str	If specified, 'output' is the name for postscript format output. Default is 'idl.ps'. The postscript format output file can be sent to a postscript printer directly.
SCALE	[I]	flt	If specified (has to be less than 1.0), the image covers less than the full size letter page. Default is SCALE = 1.0.
COLOR	[I]	int	Number of IDL color table to use for color output. PS file must be sent to a color printer. Set to -1 to use defined PS color table.
R(G,B)COLOR	[I]	intarr	RGB colors to be used for defining output color table in TVLCT. PS file must be sent to a color printer. Not compatible with COLOR keyword.
HSV	[I]	int	Set this keyword if using HSV color system for input to TVLCT.
HLS	[I]	int	Set this keyword if using HLS color system for input to TVLCT.
SEIKO	[I]	int	Set this keyword for proper margins if using a SEIKO color printer.

WARNINGS:

1. Scaling Factor has to be less than 1.0, as this routine is set for maximum display size.

2. The default procedure with no optional keywords will produce a black and white plot.

3. The procedure sets the device type back to the previous one if the device type was other than post-script, ex. 'X', 'tek' or 'regis'. But if the previous device was also 'ps', then the changed x/y sizes and offsets are retained. The user should remember to

reset 'ps' device parameters to those desired, unless psimage is used again to optimize the display scale.

4. The output file is created in the default or current directory. As postscript files may be quite large, the user should make sure there is enough available space.

5. The postscript output uses 8 bits_per_pixel resolution.

6. The default size of the output has been optimized for a TEK2SD color printer. A keyword is available for optimization on a Seiko color printer.

EXAMPLE:

1. To produce a black and white display for image image1, letting the default maxima and minima of the array be the display scale, and let the default 'idl.ps' file be the output :

```
psimage, image1
```

2. To produce a display for image array image1, scaled to 0.8 times the maximum possible letter-size output display, using 0 and 20 as minima and maxima of the display scale, and output the post-script file to a file called 'out.ps':

```
psimage,image1,0,20, output = 'out.ps', scale= 0.8
```

3. To produce a display for image image1, letting the default maxima and minima of the array be the display scale, using IDL color table 3 (Red Temperature), and let the default 'idl.ps' file be the output :

```
psimage, image1, color=3
```

4. To produce a display for image image1, letting the default maxima and minima of the array be the display scale, using user defined colors (hsv) in the HSV color system, and send output to the file image1.ps:

```
psimage, image1, rcolor=h, gcolor=s, bcolor=v, /hsv
```

NAME/ONE LINE DESCRIPTION OF ROUTINE:

Put_owner_id puts owner ID and system time on hard-copy plots.

PURPOSE:

Put owner ID and system time on hard-copy plots

CATEGORY:

Graphic display.

CALLING SEQUENCE:

Put_owner_id

INPUTS:

None

KEYWORD PARAMETERS:

None

OUTPUTS:

No explicit outputs.

NAME/ONE LINE DESCRIPTION OF ROUTINE:

PXINCIRC generates pixels within a given circle

DESCRIPTION:

This IDL function generates the pixel numbers of those pixels whose centers fall within a given angular radius of a given point. The angle is expressed in degrees and the center can be given in pixel number, unit vector, or long/lat.

CALLING SEQUENCE:

px = pxincirc(cen=cen,ang=ang,res=res,[coord=coord])

ARGUMENTS (I = input, O = output, [] = optional):

cen	I	int/flt vector	Circle center
coord	I	string	Center coordinate system ['E' (default), 'G', 'Q']
ang	I	flt	Angular radius (in degrees)
res	I	int	Resolution
px	O	int/long arr	Pixels within circle

WARNINGS:

EXAMPLE:

To extract the pixels within 10 degrees of a circle centered on pixel 3457 for a DIRBE (res=9) skymap:

```
px = pxincirc(cen=3457,ang=10,res=9)
```

To extract the pixels within 30 degrees of a circle centered on [30,-23] galactic coordinates for a FIRAS (res=6) skymap:

```
px = pxincirc(cen=[30,-23],coord='g',ang=30,res=9)
```

NAME/ONE LINE DESCRIPTION OF ROUTINE:

PXINPOLY generates pixels within a given spherical polygon

DESCRIPTION:

This IDL function generates the pixel numbers of those pixels whose centers fall within a given spherical polygon, that is, points on a sphere that are connected with great circle segments.

CALLING SEQUENCE:

px = pxinpoly(vertices=vertices,res=res,[coord=coord])

ARGUMENTS (I = input, O = output, [] = optional):

vertices	I	int/flt vector	Vertices of polygon (n) for pixel (n,2) for lon/lat (n,3) for unit vector
coord	I	string	Center coordinate system ['E' (default), 'G', 'Q'] (Ignored for pixel input)
res	I	int	Resolution
px	O	int/long arr	Pixels within polygon

WARNINGS:

The great circular arcs connecting the vertices are, by default, less than 180 degrees.

Polygons must be simply connected, that is, the edges must not cross. PXINPOLY checks for intersecting segments and returns [-1] if this is the case.

EXAMPLE:

To extract the pixels within a polygon defined by the vertices in the array 'vert' for a DIRBE (res=9) skymap:

```
vert = [45,1010,2032,27]

px = pxinpoly(vertices=vert,res=9)
```

<u>Longitude/Latitude</u>

```
vert = [[10,12,45,42],[-5,15,25,0]]   ; "Galactic coordinates"
```

Note: Longitude are specified in the first row, latitude in the second

```
px = pxinpoly(vertices=vert,res=9,coord='g')
```

NAME/ONE LINE DESCRIPTION OF ROUTINE:

PXINRING generates pixels within a given ring

DESCRIPTION:

This IDL function generates the pixel numbers of those pixels whose centers fall between two angular radii of a given point. The angles are expressed in degrees and the center can be given in pixel number, unit vector, or long/lat.

CALLING SEQUENCE:

```
px = pxinring(cen=cen,in_ang=in_ang,out_ang=out_ang,res=res, [coord=coord])
```

ARGUMENTS (I = input, O = output, [] = optional):

cen	I	int/flt vector	Circle center
coord	I	string	Center coordinate system ['E' (default), 'G', 'Q']
in_ang	I	flt	Inner angular radius (in degrees)
out_ang	I	flt	Outer angular radius (in degrees)
res	I	int	Resolution
px	O	int/long arr	Pixels within ring

WARNINGS:

EXAMPLE:

To extract the pixels within a ring defined by two circles, one 10 degrees, the other 20 degrees about a point centered on pixel 3457 for a DIRBE (res=9) skymap:

```
px = pxinring(cen=3457,in_ang=10,out_ang=20,res=9)
```

To extract the pixels within a ring defined by two circles, one 20 degrees, the other 25 degrees about a point at 30 degrees galactic longitude, -23 degrees galactic latitude for a FIRAS (res=6) skymap:

```
px = pxinring(cen=[30,-23],coord='g',in_ang=20,out_ang=25,res=6)
```

NAME/ONE LINE DESCRIPTION OF ROUTINE:

QS1_L shifts quadword integer 1 bit to left.

PURPOSE:

Shift quadword integer 1 bit to left

CALLING SEQUENCE:

a = qs1_l(a)

INPUT:

a - quadword integer (2 dim longword array)

OUTPUT:

a - shifted quadword integer

NAME/ONE LINE DESCRIPTION OF ROUTINE:

QS1_R shifts quadword integer 1 bit to right.

PURPOSE:

Shift quadword integer 1 bit to right

CALLING SEQUENCE:

a = qs1_r(a)

INPUT:

a - quadword integer (2 dim longword array)

OUTPUT:

a - shifted quadword integer

NAME:

RDFITS_STRUCT reads an entire FITS file into an IDL structure.

PURPOSE:

Read an entire FITS file into an IDL structure. Each header, image or table array is placed in a separate structure.

CALLING SEQUENCE:

RDFITS_STRUCT, filename, struct, [/NODELETE, /SILENT]

INPUT:

FILENAME = Scalar string giving the name of the FITS file

OPTIONAL KEYWORD:

NODELETE - RDFITS_STRUCT creates a temporary file with the name temp_'fitsname'.pro which contains the IDL structure definition. Normally, this temporary file is deleted -- set the /NODELETE keyword to keep it.
SILENT - Set this keyword to suppress informational displays at the terminal.

OUTPUT:

struct = structure into which FITS data is read. The primary header and image are placed into tag names HDR0 and IM0. Any extensions are placed into the tag names HDRi, TABi

PROCEDURES USED:

FITS_INFO, HEADFITS, GETTOK, READFITS, STRN, FDECOMP

METHOD:

The procedure FITS_INFO is used to determine whether a primary image exists and the number of extensions. The number and type of structure tags required is written to a temporary file and assigned to an appropiate HEADFITS or READFITS call. The temporary file is executed using CALL_PROCEDURE.

EXAMPLE:

Read the FITS file 'm33.fits' into an IDL structure, st

```
IDL> rdfits_struct, 'm33.fits', st
IDL> help, /str, st                ;Display info about the structure
```

RESTRICTIONS:

The current algorithm is not particularly efficient.

NAME:

READCOL reads a free-format ASCII data file with columns of data.

PURPOSE:

Read a free-format ASCII data file with columns of data into IDL variables. Lines of data not meeting the specified format (e.g. comments) are ignored. Columns may be separated by commas or spaces. Use READFMT to read a fixed-format ASCII file.

CALLING SEQUENCE:

readcol, name, v1, [v2, v3, v4, v5,v6, v7, v8,v9,v10,v11,v12,v13,v14,v15, FORMAT = , DEBUG = , SILENT = , SKIPLINE = , NUMLINE =]

INPUTS:

NAME - Name of ASCII data file, scalar string. In VMS, an extension of .DAT is assumed, if not supplied.

OPTIONAL INPUT KEYWORDS:

FORMAT- scalar string containing a letter specifying an IDL type for each column of data to be read. Allowed letters are A - string data, B - byte, D - double precision, F- floating point, I - integer, L - longword, and X - skip a column. Columns without a specified format are assumed to be floating point. Examples of valid values of FMT are

'A,B,I'	First column to read as 6 character string, then 1 column of byte data, 1 column integer data
'L,L,L,L'	Four columns will be read as longword arrays.
' '	All columns are floating point

If a FORMAT keyword string is not supplied, then all columns are assumed to be floating point.

SILENT - Normally, READCOL will display each line that it skips over. If SILENT is set and non-zero then these messages will be suppressed.

DEBUG - If this keyword is non-zero, then additional information is printed as READCOL attempts to read and interpret the file.

SKIPLINE - Scalar specifying number of lines to skip at the top of file before reading. Default is to start at the first line.

NUMLINE - Scalar specifying number of lines in the file to read. Default is to read the entire file

OUTPUTS:

V1,V2,V3,...V15 - IDL vectors to contain columns of data. Up to 15 columns may be read. The type of the output vectors are as specified by FORMAT.

EXAMPLES:

Each row in a file POSITION.DAT contains a star name and 6 columns of data giving an RA and Dec in sexigesimal format. Read into IDL variables. (NOTE: The star names must not contain internal spaces.)

```
IDL> FMT = 'A,I,I,F,I,I,F'
IDL> READCOL,'POSITION',F=FMT,name,hr,min,sec,deg,dmin,dsec
```

The HR,MIN,DEG, and DMIN variables will be integer vectors.

Alternatively, all except the first column could be specified as floating point.

```
IDL> READCOL,'POSITION',F='A',name,hr,min,sec,deg,dmin,dsec
```

To read just the variables HR,MIN,SEC

```
IDL> READCOL,'POSITION',F='X,I,I,F',HR,MIN,SEC
```

RESTRICTIONS:

This procedure is designed for generality and not for speed. If a large ASCII file is to be read repeatedly, it may be worth writing a specialized reader.

Columns to be read as strings must not contain spaces or commas, since these are interpreted as column delimiters. Use READFMT to read such files.

Numeric values are converted to specified format. For example, the value 0.13 read with an 'I' format will be converted to 0.

NAME/ONE LINE DESCRIPTION OF ROUTINE:

READFITS reads FITS disk file data,header into IDL

DESCRIPTION:

IDL function to read standard FITS data residing on disk into an IDL array. The user has the option of placing the FITS header information into an IDL string array. Several optional keywords may be invoked to control how the FITS data is returned into the IDL output array.

CALLING SEQUENCE:

Result = READFITS(filename [,header] [,/NOSCALE] [,NaNvalue=value] [,/SILENT] [,EXTEN_NO=value] [POINTLUN=value]

ARGUMENTS (I = input, O = output, [] = optional):

RESULT	O	arr	Array constructed from FITS file. Whether the returned result is byte, integer, etc. is determined from the FITS input file.
FILENAME	I	str	The name of the file to be read.
HEADER	[O]	str arr	Header from the FITS file.
NOSCALE	[I]	key	If present and non-zero, then the ouput data will not be scaled using the optional BSCALE and BZERO keywords in the FITS header. Default is to scale.
NANVALUE	[I]	key	This scalar is only needed on VAX architectures. It specifies the value to translate any IEEE "not a number" values in the FITS data array. It is needed because VAXen do not recognize the "not a number" convention.
SILENT	[I]	key	Normally, READFITS will display the size of the array at the terminal. The SILENT keyword will suppress this display.
EXTEN_NO	[I]	key	scalar integer which specifies the FITS extension to read. For example, specify EXTEN = 1 or /EXTEN to read the first FITS extension. READFITS reads extensions via internal recursive calls to READFITS.
POINTLUN	[I]	key	Scalar integer which specifies the position in the file at which the next I/O operation will occur. The file position is specified as the number of bytes from the start of the file. THIS KEYWORD IS USED INTERNALLY WHEN DEALING WITH READING EXTENSIONS. USERS NORMALLY WOULD NEVER USE THIS KEYWORD.

WARNINGS:

1. Cannot handle GROUP FITS.
2. Users with versions of IDL before 2.2.1 should remove the call to the TEMPORARY function near the end of the program.

EXAMPLE:

Read a FITS file TEST.FITS into an IDL image array, IM and FITS header array, H. Do not scale the data with BSCALE and BZERO.

```
IM = READFITS('TEST.FITS',H,/NOSCALE)
```

If the file contain a FITS extension, it could be read with

```
tab = READFITS( 'TEST.FITS', htab, EXTEN_NO = 1)
```

NAME:

READFMT reads a fixed format ASCII data file into IDL variables.

PURPOSE:

Quickly read a fixed format ASCII data file into IDL variables. Lines of data not meeting the specified format (e.g. comments) are ignored. To read a free format ASCII data file use the procedure READCOL

CALLING SEQUENCE:

READFMT, name, fmt, v1,[v2, v3, v4, ..., v25 , /SILENT, /DEBUG, SKIPLINE= , NUMLINE =]

INPUTS:

NAME - Name of ASCII data file. An extension of .DAT is assumed, if not supplied.
FMT - scalar string containing a valid FORTRAN read format. Must include a field length specification. Cannot include internal parenthesis. A format field must be included for each output vector. Multiple format fields are allowed, but the repetition factor must be less than 100, (.i.e. 19X is allowed but 117X is illegal).

Examples of valid FMT values are
FMT = 'A7,3X,2I4' or FMT = '1H ,5I7,2A7'
Examples of INVALID FMT values are
FMT = 'A7,B3' ;'B' is not a valid FORTRAN format
FMT = 'A7,2(I3,F5.1)' ;Internal parenthesis not allowed
FMT = 'A7,F,I' ;Field length not included

OUTPUTS:

V1,V2,V3,V4... - IDL vectors to contain columns of data. Up to 25 output vectors may be read. The type of the output vectors are specified by FMT.

OPTIONAL KEYWORD INPUTS:

SILENT - If this keyword is set and non-zero, then certain terminal output is suppressed while reading the file
DEBUG - Set this keyword to display additional information while reading the file.
SKIPLINE - Scalar specifying number of lines to skip at the top of file before reading. Default is to start at first line
NUMLINE - Scalar specifying number of lines in the file to read. Default is to read the entire file

EXAMPLES:

Each row in a fixed-formated file POSITION.DAT contains a 5 character star name and 6 columns of data giving an RA and Dec in sexigesimal format. A possible format for such data might be

```
IDL> FMT = 'A5,2I3,F5.1,2x,3I3'
```

and the file could be quickly read with

```
IDL> READFMT,'POSITION', fmt, name, hr, min, sec, deg, dmin, dsec
```

NAME will be a string vector,SEC will be a floating point vector, and the other vectors will be of integer type.

RESTRICTIONS:

This procedure is designed for generality and not for speed. If a large ASCII file is to be read repeatedly, it may be worth writing a specialized reader.

NOTES:

When reading a field with an integer format I<n>, the output vector is

byte - if n = 1
integer*2 - if 1 < n < 5
integer*4 - in all other cases

NAME/ONE LINE DESCRIPTION OF ROUTINE:

READ_MAP_FACE returns data for a face from a Skymap dataset.

DESCRIPTION:

READ_MAP_FACE is an IDL function which uses the Cobetrieve Data Server to extract data from a Skymap archive datasets. It reads a single field from a Skymap stored in quad-tree binary format, and rasterizes those pixel observations which lie on a specific map face into an output array. Refer to the "CSDR Data Base Manual" for complete descriptions of fields contained in the COBE Skymap archive datasets.

CALLING SEQUENCE:

data = READ_MAP_FACE ('dataset','rdf.field',face)

ARGUMENTS (I = input, O = output, [] = optional):

DATASET	I	string	Skymap dataset to be accessed. If the dataset parameter does not include an archive specification, the default archive location will be determined from the DAFS database.
RDF.FIELD	I	string	Record Defintion File and the field of the specified Skymap dataset to be accessed.
FACE	I	int	The particular face of the Skymap dataset to be accessed.
DATA	O	arr	An IDL array. The X-Axis of the array has the 'sky_looking' orientation.

WARNINGS:

READ_MAP_FACE only retrieves a single field from a Skymap archive dataset. To access multiple fields, refer to the documentations for the READ_SKYMAP function. Also, VAX Absolute Date and Time (ADT) and any string type (i.e. GMT) fields are not supported by this routine.

EXAMPLE:

The following call returns face 0 of CHAN.SPEC field data for the CACFR:[FIRAS.SKYMAP]FCS_CCMSP_LL.TD_8934319_9012105 Skymap dataset.

```
data = READ_MAP_FACE
('cacfr:[firas.skymap]fcs_ccmsp_ll.td_8934319_9012105','fcs_ccmsp.chan.spec',0)
```

NAME/ONE LINE DESCRIPTION OF ROUTINE:

READ_RDF returns field descriptions from RDF files.

DESCRIPTION:

An IDL function which uses the Cobetrieve Data Server to return an alphabetical list of field names from Record Definition Files.

CALLING SEQUENCE:

names = read_rdf ('dataset')

ARGUMENTS (I = input, O = output, [] = optional):

DATASET	I	str	The name of the dataset.
NAMES	O	str array	The listing of the field names.

EXAMPLE:

The following call lists the array of field names for the dataset fcs_ccmsp:

```
arr = read_rdf('fcs_cmmsp')
```

The following block of statements may be imbedded into a user's IDL procedure:

```
arr = read_rdf ('fcs_cmmsp')
print, arr
```

NAME/ONE LINE DESCRIPTION OF ROUTINE:

READ_RMS returns data for multiple fields from an RMS dataset.

DESCRIPTION:

READ_RMS is an IDL function which uses the Cobetrieve Data Server to extract data from an RMS archive dataset. User must define a Record Definition File by calling SET_RDF procedure prior to invoking this function.

CALLING SEQUENCE:

status = READ_RMS ('dataset','field[,...]',data[,...])

ARGUMENTS (I = input, O = output, [] = optional):

DATASET	I	string	The RMS dataset specification.
FIELD[,...]	I	string	Field in the dataset to be retrieved. Multiple fields must be separated by commas.
DATA[,...]	O	arr[,...]	IDL array in which READ_RMS stores the data for the specified field. Multiple data variables must be separated by commas, and the total number of data variables must match the number of fields specified.
STATUS	O	int	Integer condition value. A value of 1 is returned for success.

WARNINGS:

READ_RMS retrieves up to 8 fields from an RMS archive dataset.

EXAMPLE:

The following call retrieves NAME and VERSION fields data from the CSDR$ANCIL_REFERENCE:XAC_SHARP_H2.HDR RMS archive dataset.

```
status = READ_RMS ('csdr$ancil_reference:xac_sharp_h2.hdr', 'name,version',d1,d2)
```

The following block of statements may be imbedded into a user's IDL procedure:

```
status = 0
status = READ_RMS ('csdr$this:and.that','one,two,three',d1,d2,d3)
if (status eq 1) then begin
   .
   .
   .
endif
```

NAME/ONE LINE DESCRIPTION OF ROUTINE:

READ_SKYMAP returns data for multiple fields from a Skymap dataset.

DESCRIPTION:

READ_SKYMAP is an IDL function which uses the Cobetrieve Data Server to extract data from a Skymap archive dataset, and return the data in pixel order. This routine automatically returns the pixel number. User may override the default Record Definition File specified in the DAFS database by calling the SET_RDF procedure. Refer to the "CSDR Data Base Manual" for complete descriptions of fields contained in COBE Skymap archive datasets.

CALLING SEQUENCE:

status = READ_SKYMAP ('dataset','field[,...]',pixel,data[,...],[startpixel=n],[count=n])

ARGUMENTS (I = input, O = output, [] = optional):

DATASET	I	string	The Skymap dataset specification. If the dataset parameter does not include an archive specification, the default archive location will be determined from the DAFS database.
FIELD[,...]	I	string	Field in the dataset to be retrieved. Multiple fields must be separated by commas.
STARTPIXEL	I	keyword	Reference start pixel number. The keyword parameter 'n' must be an integer. If this keyword is omitted, 'startpixel' defaults to zero.
COUNT=n	I	keyword	Total number of pixels. The keyword parameter 'n' must be an integer. If this keyword is omitted, the total number of pixels defaults to the number of pixels in the whole map.
FACE=n	I	keyword	Number of face to be returned. Overrides startpixel and count.
/RESOL	I	keyword	Requests READ_SKYMAP to return the skymap's resolution. This requires an IDL variable to receive the resolution.
PIXEL	O	arr	IDL variable into which READ_SKYMAP stores the pixel numbers corresponding to the data returned
DATA[,...]	O	arr[,...]	IDL array(s) into which READ_SKYMAP stores the data for the specified field. Multiple data variables must be separated by commas, and there must be enough data variables for the number of fields specified.
[RES]	O	int	IDL variable which READ_SKYMAP will set to the skymap's resolution. NOTE THAT THIS WILL BE THE FIRST VARIABLE AFTER THOSE USED FOR DATA.
STATUS	O	int	Integer condition value. A value of 1 is returned for success.

WARNINGS:

READ_SKYMAP retrieves up to 8 fields plus the pixel number field from a Skymap archive dataset.

EXAMPLE:

The following call retrieves PIXEL, TEMPERATURE, and TIME fields from the CSDR$DMR_REFERENCE:DSK_SKY31A.MISSION Skymap dataset.

```
status = READ_SKYMAP ('csdr$dmr_reference:dsk_sky31a.mission',$
'temperature,time',pixel,data1,data2,startpixel=0,count=1024)
```

This call requests the return of the "photometry" field from Face 0 of a skymap, and that the skymap resolution be returned in IRES.

```
istat = READ_SKYMAP $
('dataset='dirbe_pds1:[dirbe_edit]bci_sad.skymp_cal_913620211', 'photometry', pix,$
data, ires, FACE=0, /RESOL)
```

The following block of statements may be imbedded into a user's IDL procedure.

```
status = 0
status = READ_SKYMAP ('who:that.be', 'this,and,that', pix, d1, d2, d3, startpixel=0,$
count=65536)

if (status eq 1) then begin
        .
        .
        .
endif
```

NAME/ONE LINE DESCRIPTION OF ROUTINE:
READ_TOD returns data for multiple fields from a Time-ordered dataset.

DESCRIPTION:
READ_TOD is an IDL function which uses the Cobetrieve Data Server to extract data from a Time-ordered archive dataset. User may override the default Record Definition File specified in the DAFS database by calling the SET_RDF procedure. Refer to the "CSDR Data Base Manual" for complete descriptions of fields contained in COBE Time-ordered archive datasets.

CALLING SEQUENCE:
status = READ_TOD ('dataset','field[,...]','start','stop',data[,...])

ARGUMENTS (I = input, O = output, [] = optional):

DATASET	I	string	The Time-ordered dataset. If the dataset parameter includes a archive logical name specification, the first five characters of the logical name must be 'CSDR$'. Refer to the "COBETRIEVE Programmer's Manual" for details on archive dataset naming conventions. If you omit the archive logical name, the default archive location will be determined from the DAFS database.
FIELD[,...]	I	string	Field in the dataset to be retrieved. Multiple fields must be separated by commas.
START	I	string	The start time of the time-ordered dataset in either the GMT format (yydddhhmmssmmm) or the ADT format (dd-mmm-yyyy hh:mm:ss.cc). If parts of the time fields are omitted, missing values default to zero.
STOP	I	string	The stop time of the time-ordered dataset in the GMT format (yydddhhmmssmmm) or the ADT format (dd-mmm-yyyy hh:mm:ss.cc). If parts of the time fields are omitted, missing values default to zero.
DATA[,...]	O	arr[,...]	IDL array in which READ_TOD stores the data for the specified field. Multiple data variables must be separated by commas, and the total number of data variables must match the number of fields specified.
STATUS	O	int	Integer condition value. A value of 1 is returned for success.

WARNINGS:
READ_TOD retrieves up to 8 fields from a Time-ordered archive dataset.

EXAMPLE:
The following call retrieves CT_HEAD.GMT and CT_HEAD.TIME fields data from the CSDR$FIRAS_EDIT:FDQ_SDF_RH time-ordered archive dataset.

```
status = READ_TOD ('csdr$firas_edit:fdq_sdf_rh',
'ct_head.gmt,ct_head.time','901090903','90109100',d1,d2)
```

The following block of statements may be imbedded into a user's IDL procedure.

```
status = 0
status = READ_TOD ('csdr$this:andwhat','who,what,when','1-jan-1990',
'2-jan-1990 20',this,that,huh)
if (status eq 1) then begin
  .
  .
  .
endif
```

NAME:

REMCHAR removes all appearances of a character from a string

PURPOSE:

Remove all appearances of character (char) from string (st)

CALLING SEQUENCE:

REMCHAR, ST, CHAR

INPUTS:

ST - String from which character will be removed.
CHAR- Character to be removed from string.

EXAMPLE:

If a = 'a,b,c,d,e,f,g' then
IDL> remchar,a, ','
will give a = 'abcdefg'

NAME/ONE LINE DESCRIPTION:

REPCHR replaces one character with another in a text string.

PURPOSE:

Replace all occurences of one character with another in a text string. (Use the procedure REPSTR to replace more than one character.)

CALLING SEQUENCE:

NEW = REPCHR(OLD, C1, [C2])

INPUTS:

OLD = text string to edit, scalar or vector
C1 = character to replace.

OPTIONAL INPUTS:

C2 = character to insert (def = ' ' = space).

OUTPUTS:

NEW = edited string.

EXAMPLE:

If OLD = 'THIS_IS_THE_TEXT', C1 = '_'
then NEW = REPCHR(OLD,C1) ==> NEW = 'THIS IS THE TEXT'

NAME/ONE LINE DESCRIPTION OF ROUTINE:

REPLAY redisplays an image previously stored using 'capture.pro'

DESCRIPTION:

This procedure reads an image from an unformatted file on disk previously saved by 'capture', and redisplays it on an IDL display window. The display window is automatically sized to the restored image array size.

CALLING SEQUENCE:

REPLAY, file, [image]

ARGUMENTS (I = input, O = output, [] = optional):

FILE	I	str	Name of unformatted file where 'capture' saved the IDL image
IMAGE	[I]	bytarr	Optional name of IDL array into which saved file is to be restored. Default name is 'image'.

WARNINGS:

1. This procedure can only be called from a Workstation (X-Windows) session.
2. The specified disk file produced by 'Capture' must exist.

EXAMPLE:

To redisplay an image stored in disk file 'output.dat' (as created by capture), and save the image in an array called data.

```
REPLAY, 'output.dat', data
```

NAME/ONE LINE DESCRIPTION OF ROUTINE:
REPROJ is a general reprojection facility.

DESCRIPTION:
This is an IDL procedure that creates reprojections of the native skycubes. The actual display of the projection can be suppressed if desired in which case the rasterized array can be returned as output. The skycube may be in either unfolded or sixpacked format. Note that a single face may also be used for input. If any of the following parameters are not specified on the command line then the user is prompted for them: projection type (Quad Cube, Aitoff, Global Sinusoidal, Mollweide), coordinate system (Ecliptic, Galactic, Equatorial), projection size (Small: 512 x 256, Large: 1024 x 512), and low and high values to be used in the display scaling (the defaults are the input min and max). Meridians and parallels can be overlaid on either the reprojections or the skycubes. These must be specified on the command line. The meridians and parallels are NOT stored as part of the rasterized image.

CALLING SEQUENCE:
pro reproj,inmap,[raster],[proj=proj],[coord=coord],
[psize=psize],[face=face_num],[win=win],[merd=merd],
[para=para],[gcoord=gcoord],[min=min],[max=max],[/noshow]

ARGUMENTS (I = input, O = output, [] = optional):

inmap	I/O	flt arr	Sky Cube/Face (Res 6-9)
[raster]	O	flt arr	Rasterized output array
[proj]	I	string	Projection Type: ('Q','A','S','M')
[coord]	I	string	Coordinate System: ('E','G','Q')
[psize]	I	string	Projection Size ('S','L')
[face]	I	int	face number (face inpt only)
[win]	I	int	Window # (default:0)
[merd]	I	flt arr	Meridian (longitude) array Enter "-1" to suppress
[para]	I	flt arr	Parallel (latitude) array Enter "-1" to suppress
[gcoord]	I	string	Grid coordinate system: ('E','G','Q','N') 'N' suppresses overlays
[min]	I	flt	Image Scale Minimum Default: Input Min
[max]	I	flt	Image Scale Maximum Default: Input Max
[noshow]	I	qualifier	no display proj switch

WARNINGS:
If displaying an image, an X-windows terminal must be used.

EXAMPLE:
1. To reproject a skycube (stored in the IDL array called 'incube') into a small, Aitoff projection in galactic coordinates, store the rasterized image in the IDL array 'ras_1' and display it in window 2 with the default meridians (0,60,120,180) and parallels (0,30,60) and a "forced" minimum of 0:

```
reproj, incube,out_raster,proj='a',coord='g',psize='s',win=2,min=0
```

It is not necessary to designate whether the skycube is in unfolded or sixpacked format.

2. To reproject a skycube (stored in the IDL array called 'incube') and then store the rasterized image in the IDL array 'ras_1' without displaying it:

```
reproj, incube,ras_1,/noshow
```

It is not necessary to designate the meridian and parallel information in this case, indeed it will be ignored if provided.

3. To reproject a skycube (stored in the IDL array called 'incube') and display the rasterized image (in the default window) with the meridians (40,50,60) and no parallels, without storing the raster image:

```
reproj, incube,merd=[40,50,60],para=-1,/nofile
```

In this case the user will be prompted for projection type, coordinate system, projection size, and minimum and maximum for image display scaling.

NAME/ONE LINE DESCRIPTION OF ROUTINE:
ROUND rounds a set of data to nearest absolute integers.

DESCRIPTION:
Rounds data to the nearest absolute integer.

CALLING SEQUENCE:
RESULT = round(indata)

ARGUMENTS (I = input, O = output, [] = optional):

OUTDATA	O	(Fix)	rounded value
INDATA	I		value to round

WARNINGS:
None

NAME:

SCDTT

PURPOSE:

To make short term statistical analysis of Spacecraft data simple and easy.

DESCRIPTION:

SCDTT is an IDL version 2 routine.

INVOCATION:

Once inside the IDL enviroment just type

SCDTT<cr>

The menus are easy follow and the user should be able to execute the entire routine.

OUTPUTS:

The user has several choices for output. The user can create a hardfile to be plotted later of any plot displayed while running the routine. The plot files have a PostScript format. In addition the user is able to save most of the statistical values calculated in an IDL SAVE file.

COMMON BLOCKS:

None.

RESTRICTIONS:

None.

NOTES:

The routine assumes that the user has issued tha SET_PLOT command and the terminal is setup for plots.

NAME/ONE LINE DESCRIPTION OF ROUTINE:

SETUP_PS configures for production of landscape PS plot files.

DESCRIPTION:

SETUP_PS sets the device correctly to produce landscape PS plot files. In addition, it returns the current device type extant on the system when the routine was called, enabling the user to reset the plot device without having specific knowledge of its nature after the desired PS plot is produced.

CALLING SEQUENCE:

SETUP_PS [, OLD_DEVICE] [,PLOTFILE = Plotfile]

ARGUMENTS: (I = input, O = output, [] = optional)

OLD_DEVICE	[O]	str	Name of IDL plot device at time routine was called.
PLOTFILE	[I]	str	Name of plot file to be created. Defaults to 'GENERIC.PS_PLOT'.

WARNINGS:

When user if finished creating the desired PS plot, he/she must then close the PS file and (if desired) reset the plot device using the commands

DEVICE,/CLOSE
SET_PLOT, OLD_DEVICE

EXAMPLES:

1. To create a plot of the vector YY using scale XX and write it to the PS file 'XX_YY.PS', then reset the the plot device:

```
SETUP_PS, Old_Dev, Plotfile = 'XX_YY.PS'
Plot, xx, yy, xstyle = 1, ystyle = 1, title = 'EXAMPLE'
DEVICE,/CLOSE
SET_PLOT, Old_Dev
```

2. To create a plot of the vector YY using scale XX and write it to the PS file 'GENERIC.PS_PLOT', then reset the the plot device:

```
SETUP_PS, Old_Dev
Plot, xx, yy, xstyle = 1, ystyle = 1, title = 'EXAMPLE'
DEVICE,/CLOSE
SET_PLOT, Old_Dev
```

3. To create a plot of the vector YY using scale XX and write it to the PS file 'XX_YY.PS':

```
SETUP_PS, Plotfile = 'XX_YY.PS'
Plot, xx, yy, xstyle = 1, ystyle = 1, title = 'EXAMPLE'
DEVICE,/CLOSE
```

4. To create a plot of the vector YY using scale XX and write it to the PS file 'GENERIC.PS_PLOT':

```
SETUP_PS, Old_Dev
Plot, xx, yy, xstyle = 1, ystyle = 1, title = 'EXAMPLE'
DEVICE,/CLOSE
```

NAME/ONE LINE DESCRIPTION OF ROUTINE:

SET_RDF defines an alternate Record Definition File.

DESCRIPTION:

SET_RDF is an IDL procedure which allows the user to define an alternate Record Definition File different from one in the DAFS database. Subsequent calls to READ_TOD and READ_SKYMAP IDL functions are affected. Users should note that this function defines a either logical name (VMS) or a environment variable (UNIX) which is translated during runtime of READ_TOD, READ_SKYMAP, READ_RMS, and GET_RESOLUTION IDL functions.

CALLING SEQUENCE:

SET_RDF,rdf

ARGUMENTS (I = input, O = output, [] = optional):

RDF	I	string	The Record Definition File specification.

WARNINGS:

None.

EXAMPLE:

The following call defines a Record Definition File to be used in subsequent READ_TOD, READ_SKYMAP, READ_RMS, and GET_RESOLUTION IDL function invocations.

```
SET_RDF,'user_disk:[test]temp.rdf'
```

NAME/ONE LINE DESCRIPTION OF ROUTINE:

SIGMA calculates the standard deviation value of an array.

PURPOSE:

Calculate the standard deviation value of an array, or calculate the standard deviation over one dimension of an array as a function of all the other dimensions.

CALLING SEQUENCE:

```
RESULT = SIGMA(ARRAY)
RESULT = SIGMA(ARRAY,N_PAR)
RESULT = SIGMA(ARRAY,N_PAR,DIMENSION)
```

INPUTS:

ARRAY = Input array. May be any type except string.

OPTIONAL INPUT PARAMETERS:

N_PAR = Number of parameters. Default value is zero. The number of degrees of freedom is N_ELEMENTS(ARRAY) - N_PAR. The value of sigma varies as one over the square root of the number of degrees of freedom.

DIMENSION = Optional dimension to do standard deviation over.

OUTPUTS:

The standard deviation value of the array when called with one parameter.

If DIMENSION is passed, then the result is an array with all the dimensions of the input array except for the dimension specified, each element of which is the standard deviation of the corresponding vector in the input array.

For example, if A is an array with dimensions of (3,4,5), then the command B = SIGMA(A,N,1) is equivalent to

```
B = FLTARR(3,5)
FOR J = 0,4 DO BEGIN
    FOR I = 0,2 DO BEGIN
        B(I,J) = SIGMA( A(I,*,J) , N )
    ENDFOR
ENDFOR
```

RESTRICTIONS:

Dimension specified must be valid for the array passed; otherwise the input array is returned as the output array.

NAME/ONE LINE DESCRIPTION OF ROUTINE:

SINCE_VERSION determines if current IDL release comes after user's.

PURPOSE:

Determine if the current release of IDL (as given in the !VERSION.RELEASE system variable) comes after the user specified release.

CALLING SEQUENCE:

test = SINCE_VERSION(release)

INPUT:

release - scalar string, must be formatted exactly like the !VERSION.RELEASE system variable (e.g. '2.2.1').

OUTPUT:

test - 1 if current release is identical or later the specified 'release'

EXAMPLE:

Use the /FTOXDR keyword to the BYTEORDER procedure if the current release of IDL is 2.2.2 or later

```
IDL> if since_version('2.2.2') then byteorder,a,/FTOXDR
```

NAME/ONE-LINE DESCRIPTION:

SIXPACK packs an unfolded skycube into 3x2 format (no wasted space).

DESCRIPTION:

Data compression routine takes a standard unfolded skycube, i.e.

```
00                    00
00                    00
11223344    OR  44332211
11223344        44332211
55                    55
55                    55

"Left T"          "Right T"
```

and compresses it into the space-saving form:

```
445500
445500
332211
332211
```

It will work for both single "sheets" (2-D arrays) and spectral cubes (3-D). Note that the output corresponds to a "right-T" regardless of the orientation of the input.

CALLING SEQUENCE:

box_out = SIXPACK (t_in, [t_orient])

ARGUMENTS:

t_in	2-D or 3-D data array in "standard" unfolded cube
t_orient	optional character string 'L' or 'R' to describe orientation of input T. Default is 'R'.
box_out	2-D or 3-D array (same as input) compressed unfolded cube, if an error occurs then this is equal to t_in
!err	
!error	0 if no errors, 1 if an error occurs

WARNINGS:

1. Sixpack checks to see that the input array has the proper x/y aspect ratio for an unfolded cube, and quits if it does not.
2. One may run into virtual memory problems for large spectral cubes since the output array is half as large as the original.
3. It is possible to do the compression "in place" by using the same variable as input and output in the calling sequence.

EXAMPLE:

outmap = sixpack(incube,'R')
outmap = sixpack(incube)

NAME/ONE-LINE DESCRIPTION:

SIXUNPACK "uncompresses" the packed skycube created by SIXPACK.

DESCRIPTION:

Data "uncompression" routine takes the space-saving form of the unfolded sky cube, i.e.

```
445500
445500
332211
332211
```

and unpacks it into the "standard" unfolded right-T:

```
      00
      00
44332211
44332211
      55
      55
```

It will work for both single "sheets" (2-D arrays) and spectral cubes (3-D). The empty areas of the array will be filled with zeros or, optionally, a specified bad pixel value.

CALLING SEQUENCE:

t_out = SIXUNPACK (box_in [,badval=value])

ARGUMENTS:

box_in	2-D or 3-D input data array in packed unfolded cube
badval	keyword used to specify bad pixel or fill value
t_out	2-D or 3-D output array (same type as input) in right-T-shaped unfolded cube, if an error occurs then this is equal to box_in
!err	
!error	0 if no errors, 1 if an error occurs

WARNINGS:

1. Sixunpack checks to see that the input array has the proper x/y aspect ratio for an packed cube, and quits if it does not.
2. One may run into virtual memory problems for large spectral cubes since the output array is twice as large as the original.
3. It is possible to do the uncompression "in place" by using the same variable as input and output in the calling sequence.

EXAMPLE:

tdata = SIXUNPACK(pdata, badval=-9999)

The data array will be reformatted from the packed cube into an unfolded cube with the empty portions set to -9999.

NAME/ONE LINE DESCRIPTION OF ROUTINE:

SKY determines the sky level in an image using MMM on 4000 pixels.

PURPOSE:

Determine the sky level in an image by applying the procedure MMM to approximately 4000 uniformly spaced pixels. Adapted from the DAOPHOT routine of the same name.

CALLING SEQUENCE:

```
SKY,IMAGE                          ;Print the sky background and standard deviation
SKY,IMAGE,SKYMODE,SKYSIG           ;Store output values in SKYMODE and SKYSIG
```

INPUTS:

IMAGE - One or two dimensional array

OPTIONAL OUTPUT ARRAYS:

SKYMODE - Scalar, giving the mode of the sky pixel values of the array IMAGE, as determined by the procedure MMM.

SKYSIG - Scalar, giving standard deviation of sky brightness

NAME/ONE LINE DESCRIPTION OF ROUTINE:

SKYCUT extracts sky data along a cross-sectional cut.

DESCRIPTION:

This procedure extracts and displays sky data along a cross sectional cut. Because this program can be run on both X- and non-X windows terminals, and the cut specified in a number of ways on various input formats, the user should consult the EXAMPLES section below for a fuller description of this facility. The first time user should probably read the entire section.

CALLING SEQUENCE:

PRO skycut, input,[nat_cube=nat_cube],[proj=proj],[coord=coord],
[win=win],[w_pos=w_pos],[pwin=pwin],[noplot=noplot],
[sclmin=sclmin],[sclmax=sclmax],[face=face_num],
[color=color],[n_pnts=n_pnts],[cs_plot=cs_plot], [cs_titl=cs_titl]

ARGUMENTS (I = input, O = output, [] = optional):

input	I	flt arr	skycube/face/projection input
[nat_cube]	I	flt arr	Native skycube/face input (unfolded or sixpack)
[proj]	I	string	Projection Type: ('A' - Aitoff, 'S' - Global Sinusoidal, 'M' - Mollweide)
[coord]	I	string	Coordinate System: ('E' - Ecliptic, 'G' - Galactic, 'Q' - Equatorial)
[win]	I	int	Window # of image
[w_pos]	I	int arr	Image offset within window
[pwin]	I	int	Plot window #
[noplot]	I	int	noplot flag for non-X windows terminals
[sclmin]	I	flt	Image Scale Minimum
[sclmax]	I	flt	Image Scale Maximum
[face]	I	int	face number (face inpt only)
[color]	I	int	skycut line color
[n_pnts]	O	int arr	# points in each plot
[cs_plot]	O	flt arr	storage array for plots
[cs-titl]	O	str arr	description array for plots

WARNINGS:

If displaying an image, an X-windows terminal must be used. It is recommended when accessing DIRBE data to use either a single face or a small (512 by 256) projection. Use of an entire DIRBE sky cube or large projection will slow the routine.

EXAMPLES:

1. To extract and store a skycut from a skycube called 'cube' using a non-X window terminal:

```
skycut,cube,n_pnts=npnts,cs_plot=plts,cs_titl=titl
```

The user will be prompted for the longitude and latitude of the end points of the desired cut. SKYCUT then generates a plot with the distance in degrees along the (great circle) cut on the x-axis and the values of the skymap along the cut on the y-axis. This plot is displayed in the graphics window of the terminal. The user is then given the following options: (1) to specify another cross sectional cut, (2) to plot the "complement" of the cut, that is the arc connecting the endpoints with length greater than 180 degress, (3) to store the sky cut, or (4) to exit the program. In the third case, the relevant data is stored in the array specified by the 'cs_plot' keyword which has dimensions: (720,4,16). This data array contains, for each cut, the x and y "vectors" displayed in the plot (the (*,0,*) and (*,1,*) elements, respectively), plus the longitude and latitude values of the arc (the (*,2,*) and (*,3,*) elements). Upon exiting the program, SKYCUT truncates the output arrays to minimize the number of unfilled elements. The actual number of points for a particular cut is stored in an integer vector of dimension 16 specified by the 'n_plts' keyword. In addition the description of the cut, containing the coordinates of the endpoints is stored in the string array specified by the 'cs_titl' keyword. All three storage keywords must be provided on the command line otherwise SKYCUT will not allow plots to be saved for later access.

2. To extract from a single face, 'in_face', on a non-X windows terminal, the user should supply the face number on the command line:

```
skycut,in_face,face=4
```

If the face number is not given on the command line, the user will be prompted for it. Note that in this case, the plots cannot be stored.

3. To extract from a reprojected image, for example, a galactic Aitoff projection called, 'gal_ait', the user must supply the projection type and the coordinate system:

```
skycut,gal_ait,proj='a',coord='g'
```

If either of these are not provided on the command line, the user will be prompted for them. The facility has no way of checking whether these values actually describe the projection provided so care should be taken. In addition, because the reprojection procedure introduces small spatial distortions into the data, the resulting cuts will not be as accurate as those from a skycube, therefore this type of invocation should be avoided if complete data "fidelity" is necessary.

The three invocations above will also work on an X-windows terminal as long as the user gives the window for the skycut plot in the 'pwin' keyword. If not provided, the user will be prompted for it. In the following examples, it is assumed that the user is running SKYCUT from an X-windows terminal.

4. To suppress the display of the cross-sectional plot on a non-X windows terminal, simply supply the /noplot qualifier on the command line.

```
skycut,cube,/noplot
```

If the user provides a displayed image (unfolded skycube, single face reprojection), they may use the cursor to specify the cross section. The user must provide two additional keywords on the command line: (1) 'win' which gives the window number of the displayed image and (2) 'w_pos' which gives the offset within the window of the image. This is a 2 element integer vector. If either of the keywords is missing, the user will be prompted for them.

5. To extract from a skycut from a skycube called, 'cube':

```
skycut,cube,win=0,w_pos=[32,32]
```

The image of 'cube' should already exist with window 0. This can be accomplished with the commands: 'WINDOW,0' and 'TVSCL,cube,32,32'.

6. To specify a cross section, the user should 'point and click' within the image window using the left button of the mouse. The arc will be displayed on the window overlayed on the image. The user can also specify the cut using the longitude and latitude of the arc endpoints.

For single face and reprojection images enter, respectively:

```
skycut,in_face,face=4,win=1,w_pos=[0,0]

skycut,gal_ait,proj='a',coord='g',win=2,w_pos=[16,32]
```

If the user supplies the storage keywords, the cuts can be saved for latter use.

7. If the user has access to both a skycube (or single face) and a reprojected image of it, SKYCUT allows the user to specify the cross section using the reprojected image for better orientation, while extracting the data from the cube (or face) thereby retaining complete accuracy.

The invocation is as follows:

```
skycut,gal_ait,proj='a',coord='g',win=2,w_pos=[16,32],nat_cube=cube
```

The 'nat_cube' keyword contains the name to the skycube from which the galactic Aitoff image, 'gal_ait', was created. The 'win' and 'w_pos' keywords refer to the window containing the reprojection.

Rules for specifying longitude and latitude
----

1. The longitude and latitude for a given point are specified by two numbers separated by spaces or commas.

2. To specify a cross section with endpoints 30 long -25 lat and 260 long 23.5 lat:

```
First point : 30 -25
Second point: 260  23.5
```

3. To specify a cross section along the 42nd parallel from 20 long to 180 long:

```
First point : 20 42.0
Second point: 180
```

4. To specify a cross section along the 42nd parallel from 270 long to 10 long:

   First point : 270 42.0
   Second point: 10

5. Latitude cross-sections go in the direction of increasing longitude.

6. To specify a cross section along the 90th meridian from pole to pole:

   First point : 90
   Second point: <CR>

NAME/ONE LINE DESCRIPTION:

SMOOTHCUBE convolves an unfolded skycube with an arbitrary kernel.

DESCRIPTION:

This smoothing routine operates on unfolded skycubes or single faces, properly accounting in the former case for the way the edges of the six faces match up. It will work with an arbitrary kernel. For unfolded cubes, it requires that the input array be a cube unfolding of the "right-T" type, i.e.

```
      0
4 3 2 1
      5
```

If the input array is square, it assumes it to be a single face. A flag value can be specified so that pixels with bad or missing data are not included in the smoothing process. The routine ignores such pixels and renormalizes the part of the kernel overlying "good" data. When smoothing a single face, all pixels less than a kernel width from the edge are replaced by the flag value.

CALLING SEQUENCE:

outmap = smoothcube (inmap, kernel, [badval=...])

ARGUMENTS (I=input, o=output, []=optional):

inmap	I	2-D arr, any type	Input map
kernel	I	2-D arr, any type	convolution kernel
badval	[I]	keyword flt	Flag value that identifies empty pixels. If not present, take value from the unused "face" array adjacent to face 5. If input is a single face, use 0.
outmap	O	2-D arr	Smoothed map, same type as input

WARNINGS:

1. Input map must be an unfolded cube (3:4 aspect ratio) or a square face.
2. Convolving kernel need not be square but must be odd in both dimensions in order to preserve position info in the map.
3. Mathematically meaningless results may be obtained if the smoothing kernel is not rotationally symmetric!!
4. Finally, it is always safest to specify the badval if known.

EXAMPLE:

Consider a map with e.g., -999 in the unfilled pixels, which one wishes to smooth with the following array:

```
1  2  1
2  5  2
1  2  1
```

The invocation would be:

```
UIDL> kernel = [[1,2,1],[2,5,2],[1,2,1]]
UIDL> outmap = smoothcube (inmap, kernel, badval=-999)
```

...or, IF the map is well-behaved...

```
UIDL> outmap = smoothcube (inmap, kernel)
```

NAME/ONE LINE DESCRIPTION OF ROUTINE:

SORTIT sorts a vector in ascending order of the contents of a second vector.

DESCRIPTION:

Sorts a vector in ascending order of the contents of a second vector. This function is needed because the IDL SORT function does not do a double sort correctly. In the example given below, this routine works as expected, whereas the IDL SORT function returns

XYX = [1, 1, 1, 1, 2, 2, 3, 3, 4, 4]
YYX = [5, 2, 6, 3, 2, 3, 4, 4, 4, 2]

CALLING SEQUENCE:

SortedVector = SortIt (XIn, SortVec)

ARGUMENTS (I = input, O = output, [] = optional):

XIn	I	Numeric vector	Vector to be sorted.
SortVec	I	Numeric vector	Vector specifying the sort order.
XOut	O	Numeric vector	Sorted vector.

WARNINGS:

1. Will only sort in ascending order.
2. For 2-d sort, the order of the commands is important -- don't rearrange the vector you are sorting on until you've finished using it!
3. If XIn and SortVec are different lengths, SortIt will sort the first N elements of XIn based on the first N elements of SortVec, where N = number of elements in the shorter of XIn and SortVec.

EXAMPLE:

If

X = [0,0,0,0,1,1,1,1,2,2,2,2,5,5,6]
I = [1,2,4,5,3,7,6,11,10,9,12,13,15,14,8]

then the command

OUT = SORTIT(X,I)

gives

OUT = [0,0,1,0,0,1,1,6,2,2,1,2,2,5,5]

NAME:

SPEC_DIR expands a partial file specification into a full one

PURPOSE:

Provide a complete file specification by appending a default disk or directory if necessary.

CALLING SEQUENCE:

File_spec = SPEC_DIR(filename,[extension])

INPUT:

filename - character string giving partial specification of a file name. VMS examples include 'UIT$USER2:TEST.DAT', or '[.SUB]TEST'. Unix examples include '/home/idl/lib', or '$IDL_HOME/pro'.

OPTIONAL INPUT:

exten - string giving a default file name extension to be used if filename does not contain one. Do not include the period.

OUTPUT:

File_spec - Complete file specification using default disk or directory when necessary. If the default directory is UIT$USER1:[IDL] then, for the above VMS examples, the output would be

'UIT$USER2:[IDL]TEST.DAT'
'UIT$USER2:[IDL.SUB]TEST'.

NAME/ONE LINE DESCRIPTION OF ROUTINE:

SPFILTER is a general spatial filtering tool.

DESCRIPTION:

IDL function which takes as input a 2-D array and replaces each element with the result of a user-specified function applied to all data within an NPIX x NPIX window centered on that element.

CALLING SEQUENCE:

outmap = SPFILTER(map,npix,filter_func,badval=badval)

ARGUMENTS (I = input, O = output, [] = optional):

map	I	arr	2-D array containing the input map
npix	I	scl	Window size in pixels (full width and height)
filter_func	I	str	String containing name of filter function. The function can be any IDL function which takes as an array as input and returns a scalar. Standard examples include 'MIN', 'MEDIAN','AVG'.
badval	[I]	scl	All data at this value will be ignored (Default=0).
outmap	O	arr	2-D array containing the filtered map

WARNINGS:

This routine uses the EXECUTE function to call the filter function. Since IDL does not allow recursive calls to EXECUTE, the filter function can not contain an EXECUTE call.

EXAMPLES:

1. To apply a 5x5 minimizing filter to the 2-D array called MAP:

   filtered_map = SPFILTER(map,5,'min')

2. To apply a 10x10 averaging filter to the 2-D array called MAP:

   Create within your !path a function of the form:

```
function myavg,data
n = n_elements(data)
return,total(data)/n
end
```

   Then call SPFILTER as:

   filtered_map = SPFILTER(map,10,'myavg')

NAME/ONE LINE DESCRIPTION OF ROUTINE:

STRFORM forms right-justified string of given length from input no.

PURPOSE:

Form right-justified string of desired length from numeric input

CALLING SEQUENCE:

strform, num, str, l_str

INPUT:

num - numeric value
l_str - length of output string

OUTPUT:

str - output string

NAME:

STRN

PURPOSE:

The main and original purpose of this procedure is to convert a number to and unpadded string (i.e. with no blanks around it.) However, it has been expanded to be a multi-purpose formatting tool. You may specify a length for the output string; the returned string is either sut to that length or padded to be that length. You may specify characters to be used in padding and which side to be padded. Finally, you may also specify a format for the number. NOTE that the input "number" need not be a number; it may be a string, or anything. It is converted to string.

CALLING SEQUENCE:

tmp = STRN(number,[LENGTH=, PADTYPE=, PADCHAR=, FORMAT =])

INPUT:

NUMBER	This is the input variable to be operated on. Traditionally, it was a number, but it may be any scalar type.

OPTIONAL INPUT:

LENGTH	This KEYWORD specifies the length of the returned string. If the output would have been longer, it is truncated. If the output would have been shorter, it is padded to the right length.
PADTYPE	This KEYWORD specifies the type of padding to be used, if any. 0=Padded at End, 1=Padded at front, 2=Centered (pad front/end). IF not specified, PADTYPE=1
PADCHAR	This KEYWORD specifies the character to be used when padding. The default is a space (' ').
FORMAT	This keyword allows the FORTRAN type formatting of the input number (e.g. '(f6.2)')

OUTPUT:

tmp The formatted string

USEFUL EXAMPLES:

```
print,'Used ',strn(stars),' stars.'  ==> 'Used 22 stars.'
print,'Attempted ',strn(ret,leng=6,padt=1,padch='0'),' retries.'
          ==> 'Attempted 000043 retries.'
print,strn('M81 Star List',length=80,padtype=2)
          ==> an 80 character line with 'M81 Star List' centered.
print,'Error: ',strn(err,format='(f15.2)')
          ==> 'Error: 3.24'    or ==> 'Error: 323535.22'
```

NAME:

STRNUMBER

PURPOSE:

Function to determine if a string is a valid numeric value.

CALLING SEQUENCE:

result = strnumber(st, [val])

INPUTS:

st - string

OUTPUTS:

1 is returned as the function value if the string st has a valid numeric value, otherwise, 0 is returned.

OPTIONAL OUTPUT:

val - (optional) value of the string. real*8

WARNING:

1. Note that (as of Version 2.2.2) a blank string (e.g. " ") is not a valid numeric value, although an empty string ("") is.

2. As of V2.2.2 there is a bug in the IDL ON_IOERROR procedure that will cause the following statement to hang up IDL

```
IDL> print,'' + string( strnumber('xxx') )
```

NAME:

STRPARSE

DESCRIPTION:

StrParse returns an array of non-blank entities which are extracted from the input string Strings.

NAME/ONE LINE DESCRIPTION OF ROUTINE:
SUBX performs extended word integer subtraction.

PURPOSE:
Performs extended word integer subtraction

CALLING SEQUENCE:
difference = subx(minuend, subtrahend, length, borrow_bit)

INPUT:

minuend -	extended integer (32 * n bits)
subtrahend -	extended integer (32 * n bits)
length -	length of extended integer in longwords
borrow bit -	one if minuend lt subtrahend, zero otherwise

OUTPUT:

difference -	extended integer (32 * n bits)

NAME/ONE-LINE DESCRIPTION:

SUM totals an array over one of its dimensions

DESCRIPTION:

This function performs a summation over either any one dimension of an N-dimensional array, or of the whole array (i.e. summing all elements). When called with an array index 0,1,2... it will sum over the first, second, third, etc., index of the array; when called with no index number, it totals the whole array.

CALLING SEQUENCE:

result = SUM (array, [index])

ARGUMENTS (I=input, O=output, []=optional):

RESULT	O	any data type	Output array or scalar
ARRAY	I	any data type	Input array, any dimensions
[INDEX]	[I]	integer	Dimension over which to sum

WARNINGS:

Note that the numbering of the dimensions differs from that in the TOTAL routine! In SUM, the first dimension of the array has index 0 (not 1!) for consistency with the rest of IDL index notation.

EXAMPLE:

Suppose ARR is an array of dimensions (3,4,5). The call

```
UIDL> OUT = SUM(ARR,0)
```

will yield an array OUT of dimensions (4,5). Similarly,

```
UIDL> OUT = SUM(ARR,2)
```

will cause OUT to have dimensions (3,4). Finally, note that

```
UIDL> OUT = SUM(ARR)
```

will sum all elements of ARR, so that OUT will be a scalar.

NAME/ONE LINE DESCRIPTION OF ROUTINE:

SUNPOS returns the apparent RA and Declination of the sun.

PURPOSE:

To compute the apparent Right Ascension and Declination of the Sun at a given date or a given array of dates. The routine optionally returns the ecliptic longitude of the Sun.

CALLING SEQUENCE:

SUNPOS, Dte, Ra, Dec [,elong]

ARGUMENTS (I = input, O = output, [] = optional):

DTE	I	flt [arr]	Modified Julian Date: i.e. Julian Date - 2400000.5
RA	O	dbl [arr]	Right Ascension. Units: radians
DEC	O	dbl [arr]	Declination. Units: radians.
ELONG	[O]	dbl [arr]	Ecliptic Longitude. Units: degrees.

WARNINGS:

1. The Julian Date must be input as the modified Julian Date, given by JD-2400000.5. False output obtained in any other format.
2. The outputs are not of high accuracy. The Right Ascension is will rarely be in error by more than 0.3 degrees, and the declination will rarely be in error by more than 0.2 degrees.
3. The quantities returned are for the epoch of date, not 1950.0 or 2000.0.

EXAMPLE:

To find the RA and DEC of the sun on JD 2447891.5 (0h UT, Jan 0 1990):

```
MJD=47891.0
SUNPOS, MJD, RA, DEC, ELONG          ; returns RA and DEC in radians
                                     ; Ecl. LONGitude in degrees.
print, ra*180/!pi/15 & dec*180/!pi   ; get RA in hours, DEC in deg.
        -5.3235993     -23.115547
print, elong
        279.28255
```

NAME:

TEN

PURPOSE:

Converts sexigesimal number to decimal. Inverse of SIXTY function.

CALLING SEQUENCES:

```
X=TEN([HOUR_OR_DEG,MIN,SEC])
X=TEN( HOUR_OR_DEG,MIN,SEC )
X=TEN([HOUR_OR_DEG,MIN])
X=TEN( HOUR_OR_DEG,MIN )
X=TEN([HOUR_OR_DEG])          <--  Trivial cases
X=TEN(HOUR_OR_DEG)            <--
```

INPUTS:

HOUR_OR_DEG,MIN,SEC -- Scalars giving sexigesimal quantity in order from largest to smallest.

OUTPUTS:

Function value returned = double real scalar, decimal equivalent of input sexigesimal quantity. A minus sign on any element of the input vector causes all the elements to be taken as < 0.

NAME/ONE LINE DESCRIPTION OF ROUTINE:

TIMECONV is the shell routine for time conversion procedures.

DESCRIPTION:

IDL function that acts as shell around the various time conversion routines. It is the program that the user actually calls.

CALLING SEQUENCE:

```
out_time = timeconv(in_time,ret_stat, infmt=in_fmt, outfmt=out_fmt)
```

ARGUMENTS (I = input, O = output, [] = optional):

in_time	I	[arr/scl]	Input time array
infmt	I	str	Input time format (see table below)
outfmt	I	str	Output time format
out_time	O	[arr/scl]	Output time array
ret_stat	O	int [arr]	Return status array [0 = good, -1 = bad]

Time Formats
============

'Zulu' (Format Designation: 'z')

A 16 character string consisting of the year (1858-9999), the day of the year (001-365/366), the hour (00-23), the minute (00-59), the second (00-59), and the number of milliseconds (000-999). If the string is less than 16 characters long, it is assumed that the first two characters give the year within the 20th century (00-99) unless an 'e' (for extended) is appended to the end in which case it is assumed that the first FOUR characters give the year.

EXAMPLES:

'2090123091530053'	(123rd day of 2090)
'86264134550416'	(264th day of 1986)
'1895002e'	(2nd day of 1895)

'Julian' (Format Designation: 'j')

A (non-negative) real*8 floating point variable giving the number of days (with fractional part) from 00hr 17-Nov-1858. (This is actually the "reduced" Julian date; "true" Julian is computed from 12hr 01-Jan-4713 BC.)

EXAMPLE:

40235.25

'Vax' (Format Designation: 'v')

A character string consisting of the day of the month, the month (using the standard 3 character abrieviation), the (full) year, and the hour-minute-second information as in 'Zulu'. This last portion is optional in which case '00:00:00.000' is assumed. The month may also be specified in lower case.

EXAMPLES:

'03-MAR-1989:14:07:23.952'
'10-oct-1993'

'Sec' (Format Designation: 's')

A (non-negative) real*8 floating point variable giving the number of seconds (with fractional part) since 0hr 01-JAN-1968. It must be lessthan 2.14748365e9, corresponding to: 19-JAN-2036:03:14:00.000.

EXAMPLE:

2.035456675e+8

'ADT' (Format Designation: 'a')

VMS system time, giving the number of 100 nanosecond intervals since 00:00 hours, 17-NOV-1858. It is stored in a two-dimensional longword (integer*4) array. Only times after 01-JAN-1985 00:00hr should be used.

EXAMPLE:

[46556L,9642143L] (09-FEB-1990:08:28:04.84)

WARNINGS:

The user should check the return status array to see whether element in output array was processed without an error.

EXAMPLE:

1. To convert from Julian to zulu time:

out_time = timeconv([34500.35,36783.9],r_stat,infmt='j',outfmt='z')

out_time will be a string array with 2 elements

2. To convert from adt to 'sec' time:

in_time = [[345345l,9350000l],[27l,9372260l],[-74634l,9367600l]]

out_time = timeconv(in_time,ret_stat,infmt='a',outfmt='s')

out_time will be a double array with 3 elements.

NAME/ONE LINE DESCRIPTION:

UGETDATA copies data from the UIMAGE environment to a UIDL variable.

DESCRIPTION:

This routine allows a UIDL user to transfer data from the UIMAGE environment into a main-level variable.

CALLING SEQUENCE:

UGETDATA,data

ARGUMENTS (I=input, O=output, []=optional)

data	O	2-D or 3-D arr flt	A variable which will receive an array of data.

EXAMPLE:

UGETDATA,data

NAME/ONE LINE DESCRIPTION:

UIMAGE is a package that allows one to manipulate COBE data.

DESCRIPTION:

UIMAGE is an image analysis program that works with the COBE initial products and project data sets from all three instruments (FIRAS, DIRBE, and DMR), as well as some other approved astronomical data bases. The program is invocable from both X-window and non-X-window terminals, although the functionality is somewhat different between the two.

The UIMAGE package has been provided so that Guest Investigators can have a means to perform numerous operations on data, whether or not they know IDL. A menu-driven environment is incorporated into the program, to facilitate ease of use.

Data may be inputted into UIMAGE from a disk file, or else via main-level UIDL variables. The types of disk files from which input may be acquired are COBETRIEVE-type files, COBE-specific FITS files, or COBE-specific IDL savesets. Main-level UIDL variables may be transferred into the UIMAGE environment by using the UPUTDATA routine while at the UIDL prompt.

The forms of data which can be inputted into UIMAGE are sky maps or individual sky map faces in resolution 6, 7, 8, or 9, in the sky-cube projection, and in either the ecliptic, equatorial, or galactic coordinate system.

Data may be outputted from UIMAGE into a disk file or else (indirectly) into main-level UIDL variables. The types of output disk files are COBE-specific FITS files or COBE-specific IDL savesets. Data may be transferred into main-level UIDL variables by calling the UGETDATA routine from the UIDL prompt.

CALLING SEQUENCE:

UIMAGE

ARGUMENTS (I=input, O=output, []=optional):

None.

EXAMPLE:

UIMAGE

WARNINGS:

Trying to input data that is not pixelized in the Quadrilateralized sky-cube format will cause the data input routines to crash!

NAME/ONE LINE DESCRIPTION OF ROUTINE:

UNITSTR returns a string with units corresponding to the units code.

DESCRIPTION:

This function returns the string corresponding to the units code. These codes are used in the ADB header files.

The returned string will be empty if the units code is out of range.
Units code 0 corresponds to mixed units ('mixed').
Units code 1 corresponds to no units ('none').

See the file CSDR$SOURCE:[XTR]XTR_UNITS.TXT for the definitive list.

CALLING SEQUENCE:

units_string = UnitStr(units_code)

ARGUMENTS:

units_code	Input	int	code for units, see XTR_UNITS.TXT

RETURNS:

units_string	Output	string	units empty string if code is out of range

WARNINGS:

!err / !error are not set even if the units code is out of range
check the string, if it is empty then the units code is out of range

EXAMPLE:

```
UIDL> print, UnitStr( 51 )
W/m**2/Hz/sr
UIDL>
```

NAME\ONE LINE DESCRIPTION OF ROUTINE:

UPKWHERE unpacks a 'where' vector into cube,level,col,row.

DESCRIPTION:

Takes the result of any 'where' type operation and unpacks it into the equivalent 'cube','level','col','row' format.

CALLING SEQUENCE:

UPKWHERE,Inarray,Where_vector,vec1,vec2 [,vec3,vec4]

ARGUMENTS (I = input, O = output, [] = optional):

Inarray	I	arr Array which was operated on by the where function to give the 'Where_vector'
Where_vector	I	arr The result of the 'where' operation
Vec1,Vec2,..	O	arr Depends on the dimensions of 'Inarray'. Two to four vectors will be returned. If inarray is 2-D,then vec1 gives the coloumn indices and vec2 the rows, if 3-D, vec1 is the level, vec2 the col and vec3 the rows... etc.

WARNINGS:

This procedure can only handle up to four dimensions.

EXAMPLE:

```
x = intarr(10,10) & x(2:4,2)=1 & x(5,6:8)=1
select = where(x gt 0)
print,select
        22      23      24      65      75      85
upkwhere,x,select,i,j
print,i
        2       3       4       5       5       5
print,j
        2       2       2       6       7       8
```

NAME/ONE LINE DESCRIPTION:

UPUTDATA sets up a UIMAGE data-object.

DESCRIPTION:

This routine allows a UIDL user to transfer data from a main-level UIDL variable into the UIMAGE environment. Ancilliary information about the data may also be passed. One may also use this routine to read in a UIMAGE-type data-object from an IDL saveset.

CALLING SEQUENCE:

uputdata,data,[badpixval=...],[bandno=...],[bandwidth=...],
[coordinate_system=...],[faceno=...],[frequency=...],
[instrume=...],[orient=...],[projection=...],[title=...], [weights=...],[units=...],[/noprompt]

ARGUMENTS (I=input, O=output, []=optional)

data	[I]	2-D arr	flt	A 2-D or 3-D array of data, or the name of an IDL saveset. If not specified, then the user is prompted for file name
badpixval	[I]	keyword	flt	Bad pixel value.
bandno	[I]	keyword	int	Band number.
bandwidth	[I]	keyword	int	Band width.
coordinat	[I]	keyword	str	Coordinate system (usually 'ECLIPTIC', but could also be 'EQUATORIAL' or 'GALACTIC').
datavar	[I]	keyword	str	Name of variable in a saveset that contains the data.
error	[O]	keyword	int	0 if no errors, 1 otherwise.
faceno	[I]	keyword	int	Face number (for faces only).
frequency	[I]	keyword	flt	Frequency.
frequnits	[I]	keyword	str	Frequency units.
freqvar	[I]	keyword	str	Name of variable in a saveset that contains the frequencies for 3-D data.
instrume	[I]	keyword	str	Instrument name (either 'DIRBE', 'DMR', or 'FIRAS').
noprompt	[I]	keyword	int	Flag, if set then don't prompt for missing keywords.
orient	[I]	keyword	str	Orientation ('L' or 'R').
projection	[I]	keyword	str	Projection (should be 'SKY-CUBE' at this level).
title	[I]	keyword	str	A descriptive title (preferably less than 40 characters).
weights	[I]	2-D arr	flt	A 2-D or 3-D array of weights
units	[I]	keyword	str	A string which contains both the name of the physical quantity and its units.

EXAMPLE:

UPUTDATA,data,title='DMR: 53A
Temperature',instr='DMR',proj='SKY-CUBE',coor='GALACTIC',ori='R',bad=0.,/noprompt

NAME/ONE LINE DESCRIPTION OF ROUTINE:

USRPROJ generates projections from user-generated lookup tables.

DESCRIPTION:

This IDL procedure creates reprojections of skycubes using user-defined lookup tables created by the PROJBLD facility. The user supplies the input skycube (in either unfolded or sixpack format, and the the three projection lookup tables built previously by PROJBLD. (In contrast, the REPROJ facility restores the needed lookup files, transparently, from the IDL data directory.) If default coordinate grid overlays are desired, the user must provide the projection type. The default grid is simply the standard (symmetric) grid drawn by REPROJ when the projection and grid coordinate systems are the same. If the user wants a non-standard grid, for example, a galactic grid over a projection centered on Orion, they must also provide the reference coordinate system and Euler angle vector used to define the projection in PROJBLD and the grid coordinate system ('E','G','Q'). If the Euler angle vector is not supplied on the command line, the default grid is used.

CALLING SEQUENCE:

```
pro usrproj,input,[proj_img],ilut=ilut,jlut=jlut,mask=mask,
[proj=proj_type],[coord=coord_type],[gcoord=gcoord],
[euler=euler],[face=face_num],[win=win],[merd=merd],
[para=para],[min=min],[max=max],[/noshow]
```

ARGUMENTS (I = input, O = output, [] = optional):

input	I	flt arr	Sky Cube/Face (Res 6-9) (Unfolded or Sixpack)
[proj_img]	O	flt arr	Rasterized output array
ilut	I	int arr	I*2 array containing skycube column numbers for reproj. Dimensions: (4*n,2*n)
jlut	I	int arr	I*2 array containing skycube row numbers for reproj. Dimensions: (4*n,2*n)
mask	I	byte arr	Face number / Boundary mask
[proj]	I	string	Projection Type: ('A','S','M')
[coord]	I	string	Reference Coordinate System: ('E','G','Q')
[gcoord]	I	string	Grid coordinate system ('E','G','Q')
[euler]	I	flt arr	Euler angle vector (3 elem)
[face]	I	int	face number (face inpt only)
[win]	I	int	Window # (default:0)
[merd]	I	flt arr	Meridian (longitude) array. Default: [0,60,120,180]
[para]	I	flt arr	Parallel (latitude) array. Default: [0,30,60]
[min]	I	flt	Image Scale Minimum
[max]	I	flt	Image Scale Maximum
[noshow]	I	qualifier	no display proj switch

WARNINGS:

If displaying an image, an X-windows terminal must be used.

EXAMPLE:

1. To generate and display an "anti-ecliptic" Mollweide projection of a skycube called 'cube' using the default coordinate overlay and default meridian and parallel values and return the image in the 'outimg' array:

```
usrproj,cube,outimg,proj='m',ilut=ae_i,jlut=ae_j,mask=ae_mask
```

2. To generate and display in window 2, a "CMB dipole" Global Sinusoidal projection with a ecliptic coordinate overlay with meridians at 0,45,90,135,180 degrees and no parallels:

```
usrproj,cube,outimg,proj='s',coord='g',gcoord='e',
euler=[267.7,43,0],merd=[0,45,90,135,180],para=-1,ilut=dp_i,
jlut=dp_j,mask=dp_mask,win=2
```

3. To generate a "Leo-centered" Aitoff projection from a skycube called 'incube' and store the rasterized image in 'leo_proj' without displaying it:

```
usrproj, incube,leo_proj,proj='a',/noshow
```

NAME/ONE LINE DESCRIPTION:

VTGREY displays grey-scale images on graphics terminals (e.g. VT240).

DESCRIPTION:

This routine will put a grey-scale image on a graphics screen. It is an alternative to contour plots for users of monochrome terminals.

CALLING SEQUENCE:

vtgrey,image,[title=...],[tty=....],[termx=...],[termy=...], [position=...],[norescale=...]

ARGUMENTS (I=input, o=output, []=optional):

image	I	2-D arr flt/int	Input image.
title	[I]	keyword str	Title string.
tty	[I]	keyword str	string specifying the terminal type, and thus setting termx and termy. Valid types are (case-insensitive): 'VT240' - Regis graphics (default) 'Microterm' - Regis graphics (same as VT240) 'GraphOn' - TEK graphics 'PC' - NCSA Telnet TEK Graphics 'Mac' - VersaTerm PRO - TEK on Mac Plus, SE, etc. 'LC' - VersaTerm TEK Graphics on Mac LC
termx	[I]	keyword int	Terminal X-resolution. (overrides tty)
termy	[I]	keyword int	Terminal Y-resolution. (overrides tty)
position	[I]	1-D arr int	A 1-D array with four elements which tell where the plot should be placed in an X window.
norescale	[I]	keyword	setting this keyword inhibits the default rescaling of IMAGE to fill as much of the screen as possible.

WARNINGS:

1. Can be very slow for images where most pixels are bright.
2. termx and termy may need to be set to produce acceptable output on non-standard terminals.

EXAMPLES:

1. For a VT240 or Microterm in Regis graphics mode use the default.

```
IDL> vtgrey,band1a
```

will plot a greyscale map of band1a on your terminal

2. For a Graphon in TEK graphics mode,

```
IDL> vtgrey,band1a,term='graphon'
```

will plot a greyscale map of band1a on your terminal

3. For weird terminals or emulators the default or one of the defined terminal types may work well. If not, (meaning you get an awful plaid pattern for vtgrey,findgen(200,200)) then you can play with termx and termy to find the right numbers for your terminal. E.g.

```
IDL> greyterm,band1a,211,198
```

might be what you need to get the proper spacing of pixels in the greyscale plot.

NAME/ONE LINE DESCRIPTION OF ROUTINE:

WHERENAN finds pos'ns of all vals. in array corr. to IEEE NaN vals.

PURPOSE:

Find the positions of all values within an array that correspond to the IEEE NaN (not-a-number) special values.

CALLING SEQUENCE:

Result = WHERENAN(ARRAY [, COUNT])

INPUT PARAMETERS:

ARRAY= Array to test against the IEEE NaN special values. Must be of either floating point, double-precision, or complex type.

OUTPUTS:

The result of the function is the indices of all values of ARRAY corresponding to the IEEE NaN specification, similar to the IDL WHERE function.

OPTIONAL OUTPUT PARAMETERS:

COUNT = Number of values found corresponding to IEEE NaN.

OPTIONAL KEYWORD PARAMETERS:

None.

NAME/ONE-LINE DESCRIPTION:

WINLABEL uses a GUI to position labels in a window with the cursor.

DESCRIPTION

This routine uses a GUI to allow the user to specify the text, size, thickness, orientation, and justification of a string. It also allows the user to see the string in a test window before writing it into the window of interest, and to erase a string that has just been placed into the target window. The string is positioned in the window using the cursor. As many strings as desired can be placed before exiting the routine.

The user must specify the desired target window number on the command line. The GUI that appears consists of the following:

Text box: User clicks on here, then enters text. TEXT MUST END WITH A CARRIAGE RETURN!

Sliders: Select the text size and thickness in units of 0.1 x IDL's default (i.e. default=10)

Select the orientation in degrees CCW from horizontal

Select the color, where the default 255 is brightest

Menu: Select text justification with respect to cursor position. Default = 'Center'

Buttons:

Test: Causes a horizontal, centered version of the string to appear in a test window. If you don't like it, change the settings or text and click 'Test' again.

OK: Will activate the cursor. Move it to the desired location in the window of interest, then click the left mouse button. The text will appear with the appropriate orientation, aligned to the left, center, or right of the cursor as selected.

Erase: Will overwrite the most recently-placed string with the background color.

Done: Exits WINLABEL, causing the widget box and test window to disappear

CALLING SEQUENCE:

WINLABEL, window_number

ARGUMENTS (I = input):

window_number	I	integer	Target window number

WARNINGS:

1. A valid window number must be supplied on the command line.
2. The text string MUST end in a <CR>, or no text will appear.
3. Buffering idiosyncrasies when running under MS Windows on PCs cause text output and erasing not to be completed until the next mouse event happens. Don't panic: all buffers are flushed and output completed when you leave the routine.

EXAMPLE:

WINLABEL, 3

NAME/ONE LINE DESCRIPTION OF ROUTINE:

WRITEFITS writes an IDL array into a FITS format disk file.

DESCRIPTION:

Procedure to write an IDL data array into a FITS formatted file onto disk. The corresponding FITS header may either be provided by the user, or generated by WRITEFITS.

CALLING SEQUENCE:

writefits, filename, data [,header] [,NaNvalue =]

ARGUMENTS (I = input, O = output, [] = optional):

FILENAME	I	str	String containing the name of the file to be written.
DATA	I	arr	Image array to be written to FITS file.
HEADER	[I]	str arr	String array containing the header for the FITS file. If variable HEADER is not given, the program will generate a minimal header.
NaNvalue	[I]	key	Value in the data array to be set to the IEEE NaN (not-a-number) condition. This is the FITS representation of undefined values.

WARNINGS:

1. It recommended that BSCALE and BZERO not be used (or set equal to 1. and 0) with REAL*4 or REAL*8 data.
2. Does not handle groups or extensions. WRITEFITS will remove any group parameters from the FITS header.

EXAMPLE:

Write a randomn 50 x 50 array as a FITS file creating a minimal header:

```
IDL> im = randomn(seed, 50, 50)          ;Create array
IDL> writefits, 'test', im               ;Write to a FITS file "test"
```

To write IDL float array DATA into a disk FITS file named STUFF.DAT, letting WRITEFITS generate the FITS header:

```
writefits,'stuff.dat',data
```

To write IDL byte array JUNK into a FITS file named Q.FITS, including the user specified header information in IDL string array variable HEAD:

```
writefits,'q.fits',junk,head
```

NAME/ONE LINE DESCRIPTION OF ROUTINE:

XY2PIX calculates pixel numbers from input x-y raster coordinates.

DESCRIPTION:

This IDL function creates a pixel list given x and y raster coordinate arrays and the cube resolution. The function assumes the coordinate arrays are from right oriented skycubes.

CALLING SEQUENCE:

pixel = xy2pix(x_in,y_in,res=res,[face=face],[/sixpack])

ARGUMENTS (I = input, O = output, [] = optional):

x_in	I	int arr	x-coordinate array
y_in	I	int arr	y-coordinate array
pixel	O	long arr	Output pixel list
res	I	int	Cube resolution
face	[I]	int	Face # for single face entry
sixpack	[I]	qualifier	Sixpack input format switch

WARNINGS:

Only coordinate arrays of Right-T cubes are properly processed. The pixel number is set to -1 if the coordinate is outside of the T.

EXAMPLE:

1. To create a FIRAS (res 6) pixel list from x & y raster (skycube/sixpack) coordinate arrays:

   firas_pixel = xy2pix(x_cube,y_cube,res=6) (unfolded skycube)
   firas_pixel = xy2pix(x_cube,y_cube,res=6,/sixpack) (sixpack)

2. To create a DIRBE (res 9) pixel list from x & y raster (face) coordinate arrays:

   dirbe_pixel = xy2pix(x_face,y_face,res=9,face=3)

NAME/ONE LINE DESCRIPTION OF ROUTINE:

ZOOMW zooms window and alters color tables on windowing devices

DESCRIPTION:

Displays part of an image (or graphics) from the current window expanded in another window. This window is retained for future use after the procedure finishes. The cursor is used to mark the center of the zoom, and make selections from menus to alter the color table. Instructions are provided in the session text window.

CALLING SEQUENCE:

ZOOMW [,cur_w] [,FACT=fact] [,XSIZE=xsize] [,YSIZE=ysize] [,/CONTINUOUS]

ARGUMENTS (I = input, O = output, [] = optional):

cur_w	[I]	int	window to zoom
fact	[I]	flt	zoom expansion factor, default = 4
xsize	[I]	int	X size of new window, default = 500
ysize	[I]	int	Y size of new window, default = 500
/continuous	[I]	key	If specified, the zoom window tracks the mouse.

WARNINGS:

1. The display window to be zoomed should exist.
2. Only works with color windowing systems.
3. A window is created and retained after procedure finishes.
4. If the option to save color tables as disk files is used, then the directory to which the files are written must have appropriate write protection and sufficient free space (>5 blocks/file should do).

EXAMPLE:

After displaying an image in a window using TV or TVSCL, suppose you want to get a better look at fine detail in the image. Type:

```
zoomw
```

A crosshair cursor will appear in the window. Center the cursor at the region of interest, and click the left mouse button (MB1) to create an enlarged view in a new window. Move the cursor and click again to zoom a different region. Clicking the center mouse button (MB2) will present a menu which allows you to reset the size of the zoom window and the magnification factor. This menu also leads to another menu providing access to LOADCT, ADJCT, and other procedures for altering, saving, and restoring color tables. Clicking the right mouse button (MB3) will exit from the procedure.

The more complete command:

```
zoomw,2,fact=5,xsize=200,ysize=250,/continuous
```

first switches the active display window to window 2 (i.e. it performs WSET,2), and then provides a new 200x250 pixel window in which a portion of window 2 is displayed at 5x magnification. The size and magnification can easily be altered (click the middle mouse button, MB2), so it is seldom necessary to set these in the command line. The /continuous keyword causes the magnified region to track the movements of the cursor without repeated clicks of the left mouse button (MB1). This only works well on fast computers which can recalculate the zoom window display as fast as the cursor moves. The COBECL computers do not normal qualify as fast.

NAME:

ZPARCHECK verifies data type of parameters in a procedure call.

PURPOSE:

Routine to check user parameters to a procedure and prints a message/returns if the parameters don't match

CALLING SEQUENCE:

zparcheck,progname,parameter,parnum,types,dimens,message

ARGUMENTS:

progname	I	str	name of calling procedure
parameter	I	any	variable parameter passed to the routine
parnum	I	int	parameter number for error message output
types	I	int vec or sclr	valid data types for parameter 1 - byte 2 - integer 3 - int*4 4 - real*4 5 - real*8 6 - complex 7 - string 8 - structure
ndimens	I	int sclr or vec	number of dimensions of allowed dimensions.
message	[I]	str	string message to be printed if an error is found).

WARNINGS:

Do not misuse government software.

EXAMPLES:

1. In this example, the routine simply returns since the number '10' is an integer scaler

```
UIDL> zparcheck,'DOGANDPONY',10,7,2,0,'Error you dummy'
UIDL>
```

Here we get an error since 10 is not a long scaler

```
UIDL> zparcheck,'DOGANDPONY',10,7,3,0,'Error you dummy'
```

Parameter 7 (Error you dummy) of routine DOGANDPONY is an invalid data type
Valid dimensions are: scaler
Valid types are: longword

INSTRUMENT-SPECIFIC UNCONFIGURED SOFTWARE

The following pages list some unconfigured software created by COBE scientists. Users should realize that this software is not guaranteed in any way. The documentation provided in the routines varies greatly and for that reason is not listed in this document.

See the ‘‘General Information’’ section of this document to determine how to include a directory in the IDL/UIDL path or to find out how to copy/view a program from VMS.

DIRBE UNCONFIGURED SOFTWARE

The following list is a summary of IDL procedures not currently in UIDL which DIRBE personnel have found useful in their analysis work. Most of these can be used on any data array, not just DIRBE data.

DISCLAIMER: In many cases, the procedures have not been written for general release. Some cleanup and documentation work would have to be done prior to incorporation into UIDL.

ROUTINE NAME, DIRECTORY, and AUTHOR	FUNCTION	CALLING SEQUENCE
BIWEIGHT_MEAN DBSCI1:[DBSWG.TOOLS.DBIDL] H.Freudenreich	Calculates the mean and sigma using bi-square weights	mean = BIWEIGHT_MEAN (vector [,sigma] [,weights])
DBC DBSCI2:[DBSWG_PRIV_ARCHIVE .ABS_CAL] C.Lisse	Corrects DIRBE observations of point sources for known beam profiles	result = DBC (data, bandnum, x, y, ptsdone)
DCC DBSCI2:[DBSWG_PRIV_ARCHIVE .ABS_CAL] C.Lisse	Calculates color correction coefficients for the DIRBE broadband photometry	result = DCC (bandnum, functype, temp_or_index)
DEG2XY DBSCI1:[DBSWG.TOOLS.DBIDL] M.Mitra	Convert sky coords on a projected maps to xy position	xy = DEG2XY (lonlat, array, projtype=projtype, inco=in_co, outco=out_co, infmt=in_fmt)
DIRBE_STATS DBSCI2:[BERRIMAN.STATS] G.Berriman	Generates histogram of data, plus mode, median, mean, resistant mean. Overlays Gaussian distribution and estimates confusion tail.	DIRBE_STATS, time, phot, elong, pixel_no, header
FFT_SPLINE DBSCI1:[DBSWG.TOOLS.DBIDL] B.Franz	Interpolates unevenly sample functions to even grid spacing	FFT_SPLINE, xin, yin, delx, power, freq, xout, yout
HLINE DBSCI1:[DBSWG.TOOLS.DBIDL] B.Franz	Overplots a horizontal line at specified Y value.	HLINE, y, [ltype]

IMVIM DBSCI1:[DBSWG.TOOLS.DBIDL] R.Arendt	Image vs. image correlation plot and stats. Uses routine LSQSIG.	IMVIM, x, y, [minx], [miny], [maxx], [maxy], [noplot=noplot]
KSTWO DBSCI2:[BERRIMAN.EXTRACT] G.Berriman	Kolmogrov-Smirnov test. Not available in STATLIB.	KSTWO, data1, data2, D, PROB
LINEAR_MEAN DBSCI1:[DBSWG.TOOLS.DBIDL] B.Franz	Fits outlier resistant line, value at a specified X.	mean = LINEAR_MEAN (xin, yin, xmean, sigma, cc)
LSQSIG DBSCI1:[DBSWG.TOOLS.DBIDL] R.Arendt	Runs least squares fit with sigmas.	LSQSIG, x, y, slope, constant, rmserr, sigslope, sigconst
ORIG_DATE DBSCI1:[DBSWG.TOOLS.DBIDL] B.Franz	Annotate a plot window with a date and name.	ORIG_DATE, name
OSLICE DBSCI1:[DBSWG.TOOLS.DBIDL] R.Arendt	Overplot of the same straight line slice from 2 different maps	OSLICE, arr, x1, y1, x2, y2, [smooth=smth], [median=median]
PLN03 DBSCI1:[DBSWG.TOOLS.DBIDL] B.Franz	User friendly form of LN03	PLN03, iflag, que
PLOTCOLOR DBSCI1:[DBSWG.TOOLS.DBIDL] R.Arendt	Produces multicolor lineplots. Nine rainbow color table levels is accessed via color=0-9.	PLOTCOLOR
POINT_REMOVER DBSCI2:[DRERR.V2] H.Freudenreich	Point source filter on maps (very slow execution time)	outarray = POINT_REMOVER (map [,width] [,sigma_mult] [,itmax])
PRINT_WIN DBSCI1:[DBSWG.TOOLS.DBIDL] R.Arendt	Produces a hardcopy of a window (works with PS_IMAGE)	PRINT_WIN [, window] [,queue = queue]
PROJCOOR DBSCI1:[DBSWG.TOOLS.DBIDL] J.Weiland	Get sky coords for xy position from projected maps	coord = PROJCOOR (xy, array, projtype=projtype, inco=in_co, outco=out_co, outfmt=out_fmt)

RDPIX2 DBSCI1:[DBSWG.TOOLS.DBIDL] R.Arendt	Performs RDPIX function	RDPIX2, Image, [X0], [Y0], [zoom=zoom]
RDPIXZ DBSCI1:[DBSWG.TOOLS.DBIDL] R.Arendt	Performs RDPIX function on a ZOOMW window	RDPIXZ, image
RESIS_MEAN DBSCI1:[DBSWG.TOOLS.DBIDL] H.Freudenreich B.Franz	Calculates outlier-resistant mean and sigma	RESIS_MEAN, vector, sigma_cut, mean, sigma, num_rejected
ROBUST_LINEFIT DBSCI1:[DBSWG.TOOLS.DBIDL] H.Freudenreich	Calculates outlier-resistant fit to straight line	coeff = ROBUST_LINEFIT (xin, yin, [yfit], [ss], [chi])
ROBUST_POLY_FIT DBSCI1:[DBSWG.TOOLS.DBIDL] H.Freudenreich	Calculates outlier-resistant polynomial fit	coeff = ROBUST_POLY_FIT (x, y, ndeg, [yfit], [chi])
SLICE DBSCI1:[DBSWG.TOOLS.DBIDL] R.Arendt	Straight line slices thru map	SLICE, arr, x1, y1, x2, y2, [smooth=smth], [median=median], [draw=draw]
SPEC_INT DBSCI2:[DBSWB.PRIV_ARCHIVE.OGLOW] M.Mitra	Runs interactively. Plots DIRBE 10 band spectrum for a specified pixel or patch (user must have already obtained band 1-10 data), calculates average background, performs background subtraction, and applies a mark 1.7 calibration.	.run SPEC_INT
SYSRESP DBSCI2:[DBSWG_PRIV_ARCHIVE.ABS_CAL] C.Lisse	Plots system responses for DIRBE broadband photometry.	.run SYSRESP

TVBOX DBSCI1:[DBSWG.TOOLS.DBIDL] J.Weiland	Interactive, cursor controlled generation of a graphics box of arbitrary size on image. Need routines: box_create, box_draw, box_erase, and box_save.	TVBOX, x0, y0, x1, y1, color=color
TVELLIPSE DBSCI1:[DBSWG.TOOLS.DBIDL] M.Gleason	Draws ellipse in display window.	TVELLIPSE, rmax, rmin, xc, yc, [pos_ang], [color], [THICK=thick]
UNMASK DBSCI1:[DBSWG.TOOLS.DBIDL] B.Franz	Converts bit mask(s) to array(s) of 0/1	maskarr = UNMASK (inmask)
UNPACK_WHERE DBSCI1:[DBSWG.TOOLS.DBIDL] W.Daffer	Transforms a WHERE vector result to a 3-D array	UNPACK_WHERE, inarray, where_vector, vec1, vec2, vec3, vec4
VLINE DBSCI1:[DBSWG.TOOLS.DBIDL] B.Franz	Overplots a vertical line at a specified X value.	VLINE, x, [ltype]
WRITE256 DBSCI1:[DBSWG.TOOLS.DBIDL] R.Arendt	Converts an IDL array to a BMP formatted file for the PC. Converts to a 256 color bitmap.	WRITE256, image, [red], [green], [blue], [file = file]
WRITEBMP DBSCI1:[DBSWG.TOOLS.DBIDL] R.Arendt	Convert an IDL array to a BMP formatted file for the PC. Converts to a 16-color bitmap.	WRITEBMP, image, [red], [green], [blue], [file = file]
WTD_MEAN DBSCI1:[DBSWG.TOOLS.DBIDL] B.Franz	Calculates a weighted mean	mean = WTD_MEAN (data,wts)

DMR UNCONFIGURED SOFTWARE

The following is a preliminary list of FORTRAN programs and subroutines that have been written by DMR scientists and programmers. These routines are in various stages of development and configurability. No warrantees, either expressed or implied, are made as to the reliability and performance of unconfigured code.

NOTE: Unless otherwise mentioned, all routines that operate on sky maps assume the maps are pixelized in quad-cube, pixel list format and that each pixel record has a standard DSK record structure (see CSDR$SOURCE:[DRDL]DSK_SKYMAP.RDL for a listing of this record structure).

ROUTINE NAME, DIRECTORY, and AUTHOR	FUNCTION
CORRELATION_3_PT_V2 DMRUSER:[KOGUT.ROUTINES] A.Kogut	Computes the 'reduced' 3-point correlation function of a user-specified map. The 3-point correlation function is defined as: C3 = < T(n1) T(n2) T(n3) > The 'reduced' 3-point function is defined to be C3 with n1 = n2, which may be expressed as a function of a single angle: C3'(theta) ~ 0.5 < T(n1) T^2(n2) > + 0.5 < T^2(n1) T(n2) > where theta is the angle between n1 and n2. The program prompts the user for a variety of input including the galactic cut angle and type of pixel weighting.
CROSS_CORRELATION DMRVAL:[XRDMR.HINSHAW.TOOLS] G.Hinshaw	Computes the auto- or cross-correlation function of any two sky maps of arbitrary resolution. The program has the capability to fit and subtract multipole expansions of any order from each of the input maps. The program prompts the user for all input including the galactic cut angle and type of pixel weighting to apply.
CROSS_COVARIANCE DMRVAL:[XRDMR.HINSHAW.TOOLS] G.Hinshaw	Computes correlation functions and a full covariance matrix. An upgrade of CROSS_CORRELATION.

DAI7_DKS
CSDR$SOURCE:[DAI7]
L.Tenorio (LBL)

Reads two sky maps and computes the KolmogorovSmirnov statistic and the significance level for the null hypothesis that the the data from the two maps are drawn from the same distribution. The code makes a galactic cut of 10 degrees and it assumes that the input maps are in galactic coordinates.

DAI7_GENUS
CSDR$SOURCE:[DAI7]
C.Park (Princeton)
L.Tenorio (LBL)

Calculates topological quantities for a given 180 x 89 temperature array. The quantities computed are the total curvature and the fractional area of excursions for 13 thresholds.

DAI_BLM_MAIN
CSDR$SOURCE:[DAI7]
C.Lineweaver (UCB)

Creates spherical harmonic expansion coefficients, Blm's, from user input, DMR data, or Monte Carlo models. The program also computes power spectra and correlation functions. If Monte Carlo simulations have been done, the program also finds the 95% and 68% confidence levels and medians for the power spectra and correlation functions. For further information, read the file DAI_BLM.UDF.

DAIRMS_MAIN
CSDR$SOURCE:[DAIRMS]
A.Kogut

Computes the variance of the sky temperature for a variety of galactic cut angles and smoothing scales.

DCOR_MAIN
CSDR$SOURCE:[DAICOR]
G.Smoot, UC Berkeley
C.Lineweaver, UC Berkeley

Computes the auto- and cross-correlation functions of two quad-cube pixelized sky maps.

DSRC4_MAIN
CSDR$SOURCE:[DAISRC]
R.Kummerer

Searches for sky sources by fitting Gaussian approximation of the DMR beam to selected skymaps.

FAST_HZ_MAP
DMRUSER:[KOGUT.ROUTINES].
A.Kogut

Generates and writes a sky map with approximately scale-invariant structure (Harrison-Zel'dovich spectrum) using the UPX pixel routines. The basic idea is to pixelize the sky at resolution 6, 5, 4, 3, 2 with the pixel values at each resolution drawn from the same parent population of random Gaussianly-distributed variables. To get an output map at DMR resolution (res 6), one then 1) gets the sky position for each resolution-6 pixel, 2) gets the pixel number of that position at each resolution, and 3) sums the random numbers for each pixel resolution. Large-scale correlations are thus introduced by the lowresolution pixels, which cover large areas of the sky. The resultant sky map is be approximately scale-invariant. The program prompts the user for a variety of input.

MONTE_CARLO_2PT
DMRUSER:[KOGUT.ROUTINES]
A.Kogut

This program computes the cross-correlation functions for a set of internally simulated sky maps with a given data map and compiles the results in statistical form. This routine can provide the core for a variety of Monte Carlo simulations of correlation functions.

MONTE_CARLO_3PT
DMRUSER:[KOGUT.ROUTINES].
A.Kogut

This program computes the reduced 3-point crosscorrelation functions for a set of internally simulated sky maps with a given data map and compiles the results in statistical form. This routine can provide the core for a variety of Monte Carlo simulations of 3-point correlation functions.

MONTE_CARLO_HZQ
CADCM2:[DMRSCI.HINSHAW.QUADRUPOLE]
G.Hinshaw

This program generates simulated Harrison-Zeldovich maps and fits multipole expansions to them for the purposes of studying a variety of systematic effects associated with spherical harmonic expansions derived from data with non-uniform sky coverage.

SIMULATE_HZ_MAP
DMRVAL:[XRDMR.HINSHAW.TOOLS]
G.Hinshaw

Produces a simulated sky map with a Harrison-Zeldovich spectrum of spherical harmonic components. The program prompts the user for the overall map normalization (Qrms-ps) and the minimum and maximum multipole orders to include in the simulation. The program automatically includes cosmic variance about a perfect n=1 spectrum.

SMOOTH_MAP_FAST
DMRUSER:[KOGUT.ROUTINES].
A.Kogut

Smooths a sky map (up to resolution 7) with a gaussian kernal of user-specified FWHM. The program prompts the user for the In order to facilitate the processing of high resolution maps in a reasonable amount of time, this program only convolves the kernal over pixels within

~10-20 degrees of the parent pixel. It is therefore unsuitable for smoothing with kernals whose FWHM is greater than ~10 degrees.

SMOOTH_SKYMAP
DMRVAL:[XRDMR.HINSHAW.TOOLS]
G.Hinshaw

Smooths an arbitrary resolution sky map with a gaussian kernal of user-specified FWHM. In order to facilitate the processing of high resolution maps in a reasonable amount of time, this program only convolves the kernal over pixels within ~10-20 degrees of the parent pixel. It is therefore unsuitable for smoothing with kernals whose FWHM is greater than ~10 degrees.

SUBTRACT_MULTIPOLE
DMRVAL:[XRDMR.HINSHAW.TOOLS]
G.Hinshaw

Fits and subtracts a spherical harmonic expansion of user-specified order from a given input map. The program prompts the user for a galactic cut angle to apply and can operate on maps of any resolution.

FIRAS UNCONFIGURED SOFTWARE

The following list is a summary of IDL procedures not currently in UIDL which FIRAS personnel have found useful in their analysis work. Most of these can be used on any data array, not just FIRAS data.

DISCLAIMER: In many cases, the procedures have not been written for general release. Some cleanup and documentation work would have to be done prior to incorporation into UIDL.

ROUTINE, DIRECTORY LOCATION, and AUTHOR	FUNCTION	CALLING SEQUENCE
ARRAY_DUPS FIRUSER:[FIRSWG.XRLPR.IDL] L.Rosen	Search an array for entries with the same value.	arr=ARRAY_DUPS(array)
CUBE FIRUSER:[FIRSWG.XRLPR.CUBE] L.Rosen	Shows a requested pixel location on an unfolded sky cube.	CUBE, pltdev
CONV_DALE FIRUSER:[FIRSWG.XRLPR.XCAL] L.Rosen	Reads Dale Fixsen's times for coadds from file containing them in 5 columns, and convert them to GMT. Dale's times are in years since 893251118.	.run CONV_DALE
CONV_GMT_SEC FIRUSER:[FIRSWG.XRLPR.IDL] L.Rosen	Converts GMT to seconds since 86001.	sec=CONV_GMT_SEC (gmt)
ENG_TEMP_LIST FIRUSER:[FIRSWG.XRLPR.IDL] K.Jensen	Mimics FUT_Temp_List.	ENG_TEMP_LIST, data_array, index, grtrawwt, grttrans, combsw, temp
GETTIM FIRUSER:[FIRSWG.XRLPR.XCAL] L.Rosen	Gets GMT times from Dale Fixsen's coadds.	.run GETTIM
GLIT_RAT_HISTO FIRUSER:[FIRSWG.XRLPR.GLITCH] L.Rosen	Creates histograms of the number of (IFGs) vs. glitch rate.	GLIT_RAT_HISTO, chan, jstart, jstop, plot_dev

GLITCH_HISTO FIRUSER:[FIRSWG.XRLPR.GLITCH] L.Rosen	Creates histograms of the number of (IFGs) vs. glitch count.	GLITCH_HISTO, chan, jstart, jstop, plot_dev
GMT_DEFAULT FIRUSER:[FIRSWG.XRLPR.IDL] L.Rosen	Pads GMT strings to start of day and to end of day. Fast.	GMT_DEFAULT, jstart, jstop
GRATE2 FIRUSER:[FIRSWG.XRLPR.GLITCH] L.Rosen	Newer version of GLIT_RAT_HISTO.	GRATE, chan, jstart, jstop, plot_dev
LINFIT FIRUSER:[FIRSWG.XRLPR.IDL] L.Rosen	Get the slope and y intercept from least squares fit.	LINFIT, x, y, slope, intrcpt
MTMTIDL FIRUSER:[FIRSWG.XRLPR.MTM] L.Rosen	Calculates timings of MTM modes from NFS_ANC records.	MTMTIDL, arc, fext, jstart, jstop
PUT_TIME_STAMP FIRUSER:[FIRSWG.XRLPR.IDL] T.Hewagama	Puts user ID and time stamp on the plot.	.run PUT_TIME_STAMP
REDCENT FIRUSER:[FIRSWG.XRLPR.IDL] L.Rosen	Reduces data set by replacing contiguous points within a specified percent change, by one point at the average.	REDCENT, x, y, percent
REDUCE FIRUSER:[FIRSWG.XRLPR.IDL] L.Rosen	Reduces data set by replacing contiguous points with same value by one point at the value.	REDUCE, x, y
SAA_CHECK FIRUSER:[FIRSWG.XRLPR.CUBE] L.Rosen	Plots FDQ_SDF data that is within the SAA, on terrestrial longitude & latitude map.	SAA_CHECK, jstart, jstop, chan, pltdev
STR_TO_REAL FIRUSER:[FIRSWG.XRLPR.IDL] L.Rosen	Converts a string of a real number into a real number.	real=STR_TO_REAL(snum)

XYPIX

FIRUSER:[FIRSWG.XRLPR.MOON] L.Rosen	Generates table of x,y coordinates of pixels on an opened FIRAS skycube.	.run XYPIX