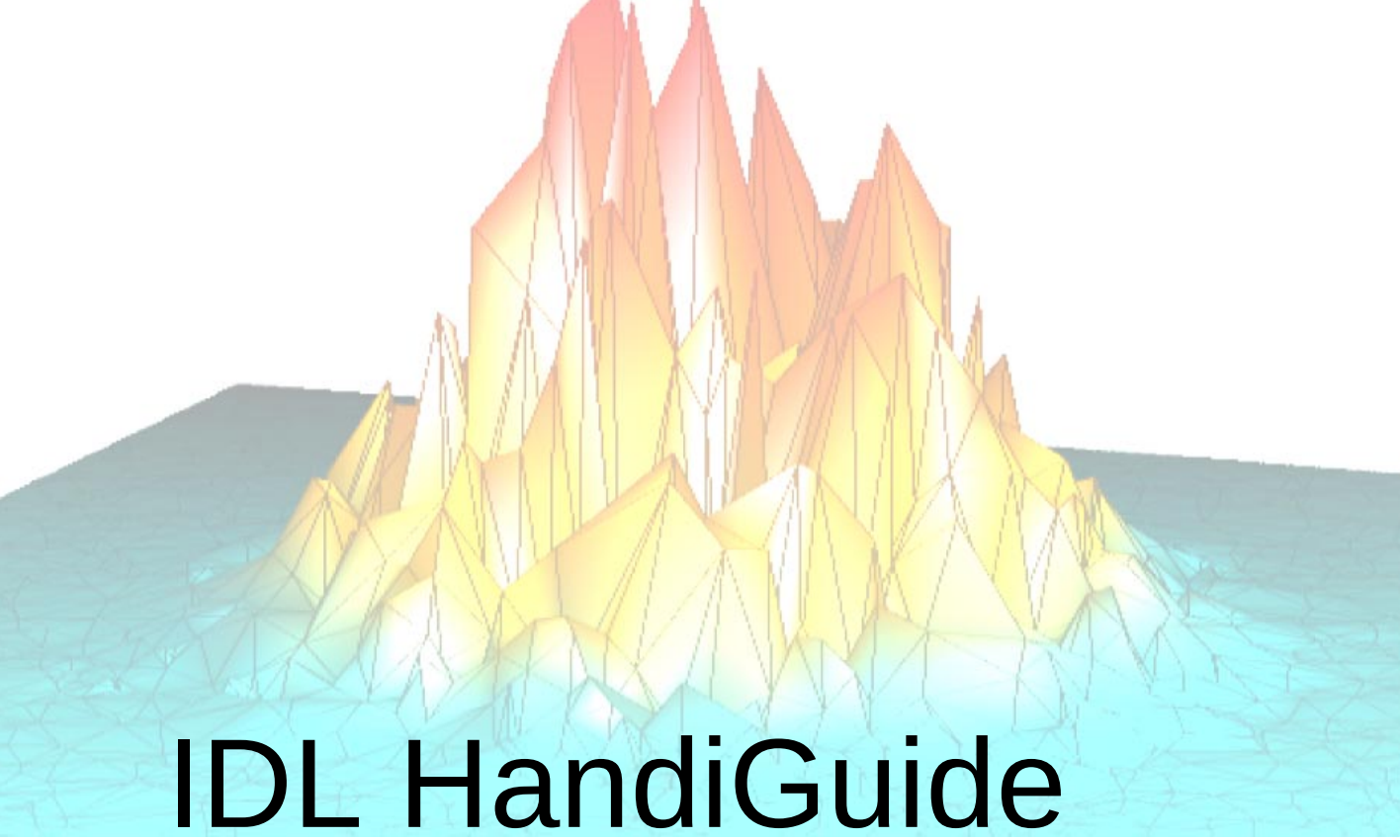

IDL HandiGuide

This quick reference guide alphabetically lists all IDL functions, procedures, statements, and objects, including the syntax of each. The following information is included in this guide:

Restricted Rights Notice

The IDL® software program and the accompanying procedures, functions, and documentation described herein are sold under license agreement. Their use, duplication, and disclosure are subject to the restrictions stated in the license agreement. Research Systems, Inc., reserves the right to make changes to this document at any time and without notice.

Limitation of Warranty

Research Systems, Inc. makes no warranties, either express or implied, as to any matter not expressly set forth in the license agreement, including without limitation the condition of the software, merchantability, or fitness for any particular purpose.

Research Systems, Inc. shall not be liable for any direct, consequential, or other damages suffered by the Licensee or any others resulting from use of the IDL software package or its documentation.

Permission to Reproduce this Manual

If you are a licensed user of this product, Research Systems, Inc. grants you a limited, nontransferable license to reproduce this particular document provided such copies are for your use only and are not sold or distributed to third parties. All such copies must contain the title page and this notice page in their entirety.

Acknowledgments

IDL Syntax

Function: *Result*= FUNCTION(*Argument1* [, *Argument2*] [, KEYWORD1=*value*] [, /KEYWORD2])

Procedure: PROCEDURE, *Argument1* [, *Argument2*] [, KEYWORD1={*value1* | *value2*}] [, /KEYWORD2]

Statement: IF *expression* THEN *statement* [ELSE *statement*]

Elements of Syntax

Element	Description
[ ] (Square brackets)	Indicates that the contents are optional.
[ ] (Italicized square brackets)	Indicates that the square brackets are part of the statement (used to define an array).
Argument	Arguments are shown in italics, and must be specified in the order listed.
KEYWORD	Keywords are all caps, and can be specified in any order. For functions, all arguments and keywords must be contained within parentheses.
/KEYWORD	Indicates a boolean keyword.
Italics	Indicates arguments, expressions, or statements for which you must provide values.
{ } (Braces)	• Indicates that you must choose one of the values they contain • Encloses a list of possible values, separated by vertical lines (|) • Encloses useful information about a keyword • Defines an IDL structure (this is the only case in which the braces are included in the statement).
| (Vertical lines)	Separates multiple values or keywords.
[, $Value_1$, ... , $Value_n$]	Indicates that any number of values can be specified.
[, $Value_1$, ... , $Value_8$]	Indicates the maximum number of values that can be specified.

Square Brackets ([ ])

- Content between square brackets is optional. Pay close attention to the grouping of square brackets. Consider the following examples:

  ROUTINE_NAME, *Value1* [, *Value2*] [, *Value3*] : You must include *Value1*. You do not have to include *Value2* or *Value3*. *Value2* and *Value3* can be specified independently.

  ROUTINE_NAME, *Value1* [, *Value2*, *Value3*] : You must include *Value1*. You do not have to include *Value2* or *Value3*, but you must include both *Value2* and *Value3*, or neither.

  ROUTINE_NAME [, *Value1* [, *Value2*]] : You can specify *Value1* without specifying *Value2*, but if you specify *Value2*, you must also specify *Value1*.

- Do not include square brackets in your statement unless the brackets are italicized. Consider the following syntax:

  Result = KRIG2D(*Z* [, *X*, *Y*] [, BOUNDS=*[xmin, ymin, xmax, ymax]*])

  An example of a valid statement is:

  R = KRIG2D(Z, X, Y, BOUNDS=[0,0,1,1])
- Note that when [, $Value_1$, ... , $Value_n$] is listed, you can specify any number of arguments. When an explicit number is listed, as in [, $Value_1$, ... , $Value_8$], you can specify only as many arguments as are listed.

Braces ({ })

- For certain keywords, a list of the possible values is provided. This list is enclosed in braces, and the choices are separated by a vertical line (|). Do not include the braces in your statement. For example, consider the following syntax:

  LIVE_EXPORT [, QUALITY={0 | 1 | 2}]

  In this example, you must choose either 0, 1, or 2. An example of a valid statement is:

  LIVE_EXPORT, QUALITY=1
- Braces are used to enclose the allowable range for a keyword value. Unless otherwise noted, ranges provided are inclusive. Consider the following syntax:

  Result = CVTTOBM(*Array* [, THRESHOLD=*value*{0 to 255}])

  An example of a valid statement is:

  Result = CVTTOBM(A, THRESHOLD=150)
- Braces are also used to provide useful information about a keyword. For example:

  [, LABEL=*n*{label every *n*th gridline}]

  Do not include the braces or their content in your statement.
- Certain keywords are prefaced by X, Y, or Z. Braces are used for these keywords to indicate that you must choose one of the values it contains. For example, [{X | Y}RANGE=*array*] indicates that you can specify either XRANGE=*array* or YRANGE=*array.*
- Note that in IDL, braces are used to define structures. When defining a structure, you *do* want to include the braces in your statement.

Italics

- Italicized words are arguments, expressions, or statements for which you must provide values. The value you provide can be a numerical value, such as 10, an expression, such as DIST(100), or a named variable. For keywords that expect a string value, the syntax is listed as KEYWORD=*string*. The value you provide can be a string, such as 'Hello' (enclosed in single quotation marks), or a variable that holds a string value.
- The italicized values that must be provided for keywords are listed in the most helpful terms possible. For example, [, XSIZE=*pixels*] indicates that the XSIZE keyword expects a value in pixels, while [, ORIENTATION=*ccw_degrees_from_horiz*] indicates that you must provide a value in degrees, measured counter-clockwise from horizontal.

Specifying Keywords

- Certain keywords are boolean, meaning they can be set to either 0 or 1. These keywords are switches used to turn an option on and off. Usually, setting such keywords equal to 1 causes the option to be turned on. Explicitly setting the keyword to 0 (or not including the keyword) turns the option off. All keywords in this reference that are preceded by a slash can be set by prefacing them by the slash. For example, SURFACE, DIST(10), /SKIRT is a shortcut for SURFACE, DIST(10), SKIRT=1. To turn the option back off, you must set the keyword equal to 0, as in SURFACE, DIST(10), SKIRT=0.

  In rare cases, a keyword's default value is 1. In these cases, the syntax is listed as KEYWORD=0, as in SLIDE_IMAGE [, *Image*] [, CONGRID=0]. In this example, CONGRID is set to 1 by default. If you specify CONGRID=0, you can turn it back on by specifying either /CONGRID or CONGRID=1.
- Some keywords are used to obtain values that can be used upon return from the function or procedure. These keywords are listed as KEYWORD=*variable*. Any valid variable name can be used for these keywords, and the variable does not need to be defined first.
- Some routines have keywords that are mutually exclusive, meaning only one of the keywords can be present in a given statement. These keywords are grouped together, and separated by a vertical line. For example, consider the following syntax:

  PLOT, [*X*,] *Y* [, /DATA | , /DEVICE | , /NORMAL]

  In this example, you can choose either DATA, DEVICE, or NORMAL, but not more than one. An example of a valid statement is:

  PLOT, SIN(A), /DEVICE
- Keywords can be abbreviated to their shortest unique length. For example, the XSTYLE keyword can be abbreviated to XST because there are no other keywords in IDL that begin with XST. You cannot shorten XSTYLE to XS, however, because there are other keywords that begin with XS, such as XSIZE.

Alphabetical Listing

The following alphabetical listing contains all IDL functions, procedures, and statements included in IDL version 5.3.

A

A_CORRELATE - Computes autocorrelation.
Result = A_CORRELATE(*X*, *Lag* [, /COVARIANCE] [, /DOUBLE])

ABS - Returns the absolute value of *X*.
Result = ABS(*X*)

ACOS - Returns the arc-cosine of *X*.
Result = ACOS(*X*)

ADAPT_HIST_EQUAL - Performs adaptive histogram equalization
Result = ADAPT_HIST_EQUAL (*Image* [, CLIP=*value*] [, NREGIONS=*nregions*] [, TOP=*value*])

ALOG - Returns the natural logarithm of *X*.
Result = ALOG(*X*)

ALOG10 - Returns the logarithm to the base 10 of *X*.
Result = ALOG10(*X*)

AMOEBA - Minimizes a function using downhill simplex method.
Result = AMOEBA(*Ftol* [, FUNCTION_NAME=*string*] [, FUNCTION_VALUE=*variable*] [, NCALLS=*value*] [, NMAX=*value*] [, P0=*vector*, SCALE=*vector* | , SIMPLEX=*array*])

ANNOTATE - Starts IDL widget used to interactively annotate images and plots with text and drawings.
ANNOTATE [, COLOR_INDICES=*array*] [, DRAWABLE=*widget_id* | , WINDOW=*index*] [, LOAD_FILE=*filename*] [/TEK_COLORS]

ARG_PRESENT - Returns TRUE if the value of the specified variable will be passed back to the caller.
Result = ARG_PRESENT(*Variable*)

ARROW - Draws line with an arrow head.
ARROW, *X0*, *Y0*, *X1*, *Y1* [, /DATA | , /NORMALIZED] [, HSIZE=*length*] [, COLOR=*index*] [, HTHICK=*value*] [, /SOLID] [, THICK=*value*]

ASCII_TEMPLATE - Presents a GUI that generates a template defining an ASCII file format.
Result = ASCII_TEMPLATE([*Filename*] [, BROWSE_LINES=*lines*] [, CANCEL=*variable*] [, GROUP=*widget_id*])

ASIN - Returns the arc-sine of *X*.
Result = ASIN(*X*)

ASSOC - Associates an array structure with a file.
Result = ASSOC(*Unit*, *Array_Structure* [, *Offset*] [, /PACKED])

ATAN - Returns the arc-tangent of *X*.
Result = ATAN(*X*) or *Result* = ATAN(*Y*, *X*)

AXIS - Draws an axis of the specified type and scale.
AXIS [, *X* [, *Y* [, *Z*]]] [, /SAVE] [, XAXIS={0 | 1} | YAXIS={0 | 1} | ZAXIS={0 | 1 | 2 | 3}] [, /XLOG] [, /YNOZERO] [, /YLOG] [, /ZLOG]
Graphics Keywords: [, CHARSIZE=*value*] [, CHARTHICK=*integer*] [, COLOR=*value*] [, /DATA | , /DEVICE | , /NORMAL] [, FONT=*integer*] [, /NODATA] [, /NOERASE] [, SUBTITLE=*string*] [, /T3D] [, TICKLEN=*value*] [, {X | Y | Z}CHARSIZE=*value*] [, {X | Y | Z}GRIDSTYLE=*integer*{0 to 5}] [, {X | Y | Z}MARGIN=*[left, right]*] [, {X | Y | Z}MINOR=*integer*] [, {X | Y | Z}RANGE=*[min, max]*] [, {X | Y | Z}STYLE=*value*] [, {X | Y | Z}THICK=*value*] [, {X | Y | Z}TICKFORMAT=*string*] [, {X | Y | Z}TICKLEN=*value*] [, {X | Y | Z}TICKNAME=*string_array*] [, {X | Y | Z}TICKS=*integer*] [, {X | Y | Z}TICKV=*array*] [, {X | Y | Z}TICK_GET=*variable*] [, {X | Y | Z}TITLE=*string*] [, ZVALUE=*value*{0 to 1}]

B

BAR_PLOT - Creates a bar graph.
BAR_PLOT, *Values* [, BACKGROUND=*color_index*] [, BARNAMES=*string_array*] [, BAROFFSET=*scalar*] [, BARSPACE=*scalar*] [, BARWIDTH=*value*] [, BASELINES=*vector*] [, BASERANGE=*scalar*{0.0 to 1.0}] [, COLORS=*vector*] [, /OUTLINE] [, /OVERPLOT] [, /ROTATE] [, TITLE=*string*] [, XTITLE=*string*] [, YTITLE=*string*]

BEGIN...END - Defines a block of statements.
BEGIN
statements
END

BESELI - Returns the I Bessel function of order *N* for *X*.
Result = BESELI(*X*, *N*)

BESELJ - Returns the J Bessel function of order *N* for *X*.
Result = BESELJ(*X*, *N*)

BESELY - Returns the Y Bessel function of order *N* for *X*.
Result = BESELY(*X*, *N*)

BETA - Returns the value of the beta function.
Result = BETA(*Z*, *W* [, /DOUBLE])

BILINEAR - Computes array using bilinear interpolation.
Result = BILINEAR(*P*, *IX*, *JY*)

BIN_DATE - Converts ASCII date/time string to binary string.
Result = BIN_DATE(*Ascii_Time*)

BINARY_TEMPLATE - Presents a GUI for interactively generating a template structure for use with READ_BINARY.
Template = BINARY_TEMPLATE ([*Filename*]
[, CANCEL=*variable*] [, GROUP=*widget_id*]
[, N_ROWS=*rows*] [, TEMPLATE=*filename*])

BINDGEN - Returns byte array with each element set to its subscript.
Result = BINDGEN(D_1, ...,D_8)

BINOMIAL - Computes binomial distribution function.
Result = BINOMIAL(*V*, *N*, *P*)

BLAS_AXPY - Updates existing array by adding a multiple of another array.
BLAS_AXPY, *Y*, *A*, *X* [, *D1*, *Loc1* [, *D2*, *Range*]]

BLK_CON - Convolves input signal with impulse-response sequence.
Result = BLK_CON(*Filter*, *Signal*
[, B_LENGTH=*scalar*])

BOX_CURSOR - Emulates the operation of a variable-sized box cursor.
BOX_CURSOR, [*X0*, *Y0*, *NX*, *NY* [, /INIT]
[, /FIXED_SIZE]] [, /MESSAGE]

BREAKPOINT - Sets and clears breakpoints for debugging.
BREAKPOINT [, *File*], *Index* [, AFTER=*integer*]
[, /CLEAR] [, CONDITION='*expression*'] [, /DISABLE]
[, /ENABLE] [, /ONCE] [, /SET]

BROYDEN - Solves nonlinear equations using Broyden's method.
Result = BROYDEN(*X*, *Vecfunc* [, CHECK=*variable*]
[, /DOUBLE] [, ITMAX=*value*] [, STEPMAX=*value*]
[, TOLF=*value*] [, TOLMIN=*value*] [, TOLX=*value*])

BYTARR - Creates a byte vector or array.
Result = BYTARR(D_1, ..., D_8 [, /NOZERO])

BYTE - Converts argument to byte type.
Result = BYTE(*Expression*[, *Offset* [, Dim_1, ..., Dim_8]])

BYTEORDER - Converts integers between host and network byte ordering.
BYTEORDER, $Variable_1$, ..., $Variable_n$ [, /DTOVAX]
[, /DTOXDR] [, /FTOVAX] [, /FTOXDR] [, /HTONL]
[, /HTONS] [, /L64SWAP] [, /LSWAP] [, /NTOHL]
[, /NTOHS] [, /SSWAP] [, /SWAP_IF_BIG_ENDIAN]
[, /SWAP_IF_LITTLE_ENDIAN] [, /VAXTOD]
[, /VAXTOF] [, /XDRTOD] [, /XDRTOF]
VMS keywords: [, /DTOGFLOAT] [, /GFLOATTOD]

BYTSCL - Scales all values of an array into range of bytes.
Result = BYTSCL(*Array* [, MAX=*value*] [, MIN=*value*]
[, /NAN] [, TOP=*value*])

C

C_CORRELATE - Computes cross correlation.
Result = C_CORRELATE(*X*, *Y*, *Lag* [, /COVARIANCE]
[, /DOUBLE])

CALDAT - Converts Julian date to month, day, year.
CALDAT, *Julian*, *Month* [, *Day* [, *Year* [, *Hour* [, *Minute*
[, *Second*]]]]]

CALENDAR - Displays a calendar for a given month or year.
CALENDAR [[, *Month*] , *Year*]

CALL_EXTERNAL - Calls a function in an external sharable object and returns a scalar value.
Result = CALL_EXTERNAL(*Image*, *Entry* [, P_0, ..., P_{N-1}]
[, /ALL_VALUE] [, B_VALUE | , /D_VALUE | ,
/F_VALUE | , /I_VALUE | , /L64_VALUE | , /S_VALUE | ,
/UI_VALUE | , /UL_VALUE | , /UL64_VALUE]
[, /CDECL] [, RETURN_TYPE=*value*] [, /UNLOAD]
[, VALUE=*byte_array*])
VMS keywords: [, DEFAULT=*string*] [, /PORTABLE]
[, /VAX_FLOAT]

CALL_FUNCTION - Calls an IDL function.
Result = CALL_FUNCTION(*Name* [, P_1, ..., P_n])

CALL_METHOD - Calls an IDL object method.
CALL_METHOD, *Name*, *ObjRef*, [, P_1, ..., P_n] or
Result = CALL_METHOD(*Name*, *ObjRef*, [, P_1, ..., P_n])

CALL_PROCEDURE - Calls an IDL procedure.
CALL_PROCEDURE, *Name* [, P_1, ..., P_n]

CASE...ENDCASE - Selects one statement for execution, depending on the value of an expression.
CASE *expression* OF
expression: *statement*
...
expression: *statement*
[ELSE: *statement*]
ENDCASE

CATCH - Intercepts and processes error messages, and continues program execution.
CATCH, *Variable* [, /CANCEL]

CD - Sets and/or changes the current working directory.
CD [, *Directory*] [, CURRENT=*variable*]

CDF_* Routines - See "CDF Routines" on page 38.

CEIL - Returns the closest integer greater than or equal to *X*.
Result = CEIL(*X*)

CHEBYSHEV - Returns the forward or reverse Chebyshev polynomial expansion.
Result = CHEBYSHEV(*D, N*)

CHECK_MATH - Returns and clears accumulated math error status.
Result = CHECK_MATH([, MASK=*bitmask*] [, /NOCLEAR] [, /PRINT])

CHISQR_CVF - Computes cutoff value in a Chi-square distribution.
Result = CHISQR_CVF(*P, Df*)

CHISQR_PDF - Computes Chi-square distribution function.
Result = CHISQR_PDF(*V, Df*)

CHOLDC - Constructs Cholesky decomposition of a matrix.
CHOLDC, *A, P* [, /DOUBLE]

CHOLSOL - Solves set of linear equations (use with CHOLDC).
Result = CHOLSOL(*A, P, B* [, /DOUBLE])

CINDGEN - Returns a complex array with each element set to its subscript.
Result = CINDGEN(D_1, ..., D_8)

CIR_3PNT - Returns radius and center of circle, given 3 points.
CIR_3PNT, *X, Y, R, X0, Y0*

CLOSE - Closes the specified files.
CLOSE[, $Unit_1$, ..., $Unit_n$] [, /ALL] [, /FILE]

CLUST_WTS - Computes the cluster weights of an array for cluster analysis.
Result = CLUST_WTS(*Array* [, /DOUBLE] [, N_CLUSTERS=*value*] [, N_ITERATIONS=*integer*] [, VARIABLE_WTS=*vector*])

CLUSTER - Performs cluster analysis.
Result = CLUSTER(*Array, Weights* [, /DOUBLE] [, N_CLUSTERS=*value*])

COLOR_CONVERT - Converts color triples to and from RGB, HLS, and HSV.
COLOR_CONVERT, I_0, I_1, I_2, O_0, O_1, O_2 {, /HLS_RGB | , /HSV_RGB | , /RGB_HLS | , /RGB_HSV}

COLOR_QUAN - Converts true-color (24-bit) image to pseudo-color (8-bit) image.
Result = COLOR_QUAN(*Image_R, Image_G, Image_B, R, G, B*)
or
Result = COLOR_QUAN(*Image, Dim, R, G, B*)
Keywords: [, COLORS=*integer*{2 to 256}] [, CUBE={2 | 3 | 4 | 5 | 6} | , GET_TRANSLATION=*variable* [, /MAP_ALL]] [, /DITHER] [, ERROR=*variable*] [, TRANSLATION=*vector*]

COMFIT - Fits using gradient-expansion least-squares method.
Result = COMFIT(*X, Y, A* {, /EXPONENTIAL | , /GEOMETRIC | , /GOMPERTZ | , /HYPERBOLIC | , /LOGISTIC | , /LOGSQUARE} [, SIGMA=*variable*] [, WEIGHTS=*vector*] [, YFIT=*variable*])

COMMON - Creates a common block.
COMMON *Block_Name*, $Variable_1$, ..., $Variable_n$

COMPILE_OPT - Gives IDL compiler information that changes the default rules for compiling functions or procedures.
COMPILE_OPT opt_1 [, opt_2, ..., opt_n]
Note: opt_n can be IDL2, DEFINT32, HIDDEN, OBSOLETE, or STRICTARR

COMPLEX - Solves complex linear system using LU decomposition.
Result = COMPLEX(*Real* [, *Imaginary*])
or
Result = COMPLEX(*Expression, Offset*, Dim_1 [, ..., Dim_8])

COMPLEXARR - Creates a complex, single-precision, floating-point vector or array.
Result = COMPLEXARR(D_1, ..., D_8 [, /NOZERO])

COMPLEXROUND - Rounds a complex array.
Result = COMPLEXROUND(*Input*)

COMPUTE_MESH_NORMALS - Computes normal vectors for a set of polygons.
Result=COMPUTE_MESH_NORMALS(*fVerts*[, *iConn*])

COND - Computes the condition number of a square matrix.
Result = COND(*A* [, /DOUBLE])

CONGRID - Resamples an image to any dimensions.
Result = CONGRID(*Array, X, Y, Z* [, CUBIC=*value*{-1 to 0}] [, /INTERP] [, /MINUS_ONE])

CONJ - Returns the complex conjugate of *X*.
Result = CONJ(*X*)

CONSTRAINED_MIN - Minimizes a function using Generalized Reduced Gradient Method.
CONSTRAINED_MIN, *X, Xbnd, Gbnd, Nobj, Gcomp, Inform* [, ESPTOP=*value*] [, LIMSER=*value*] [, /MAXIMIZE] [, NSTOP=*value*] [, REPORT=*filename*] [, TITLE=*string*]

CONTOUR - Draws a contour plot.
CONTOUR, *Z* [, *X, Y*] [, C_ANNOTATION=*vector_of_strings*] [, C_CHARSIZE=*value*] [, C_CHARTHICK=*integer*] [, C_COLORS=*vector*] [, C_LABELS=*vector*{each element 0 or 1}] [, C_LINESTYLE=*vector*] [, /FILL | , /CELL_FILL [, C_ORIENTATION=*degrees*] [, C_SPACING=*value*]] [, C_THICK=*vector*] [, /CLOSED] [, /DOWNHILL] [, /FOLLOW] [, /IRREGULAR] [, LEVELS=*vector* | NLEVELS=*integer*{1 to 60}] [, MAX_VALUE=*value*] [, MIN_VALUE=*value*] [, /OVERPLOT] [{, /PATH_DATA_COORDS, PATH_FILENAME=*string*, PATH_INFO=*variable*, PATH_XY=*variable*} | , TRIANGULATION=*variable*] [, /XLOG] [, /YLOG] [, /ZAXIS] [, /ZLOG]
Graphics Keywords: [, BACKGROUND=*color_index*] [, CHARSIZE=*value*] [, CHARTHICK=*integer*]

[, CLIP=*[X_0, Y_0, X_1, Y_1]*] [, COLOR=*value*] [, /DATA | , /DEVICE | , /NORMAL] [, FONT=*integer*] [, /NOCLIP] [, /NODATA] [, /NOERASE] [, POSITION=*[X_0, Y_0, X_1, Y_1]*] [, SUBTITLE=*string*] [, /T3D] [, THICK=*value*] [, TICKLEN=*value*] [, TITLE=*string*] [, {X | Y | Z}CHARSIZE=*value*] [, {X | Y | Z}GRIDSTYLE=*integer*{0 to 5}] [, {X | Y | Z}MARGIN=*[left, right]*] [, {X | Y | Z}MINOR=*integer*] [, {X | Y | Z}RANGE=*[min, max]*] [, {X | Y | Z}STYLE=*value*] [, {X | Y | Z}THICK=*value*] [, {X | Y | Z}TICKFORMAT=*string*] [, {X | Y | Z}TICKLEN=*value*] [, {X | Y | Z}TICKNAME=*string_array*] [, {X | Y | Z}TICKS=*integer*] [, {X | Y | Z}TICKV=*array*] [, {X | Y | Z}TICK_GET=*variable*] [, {X | Y | Z}TITLE=*string*] [, ZVALUE=*value*{0 to 1}]

CONVERT_COORD - Transforms coordinates to and from the coordinate systems supported by IDL.

Result = CONVERT_COORD(*X* [, *Y* [, *Z*]] [, /DATA | , /DEVICE | , /NORMAL] [, /T3D] [, /TO_DATA | , /TO_DEVICE | , /TO_NORMAL])

CONVOL - Convolves two vectors or arrays.

Result = CONVOL(*Array, Kernel* [, *Scale_Factor*] [, /CENTER] [, /EDGE_WRAP] [, /EDGE_TRUNCATE])

COORD2TO3 - Returns 3D data coordinates given normalized screen coordinates.

Result = COORD2TO3(*Mx, My, Dim, D0* [, *PTI*])

CORRELATE - Computes the linear Pearson correlation.

Result = CORRELATE(*X* [, *Y*] [, /COVARIANCE] [, /DOUBLE])

COS - Returns the cosine of X.

Result = COS(*X*)

COSH - Returns the hyperbolic cosine of *X*.

Result = COSH(*X*)

CRAMER - Solves system of linear equations using Cramer's rule.

Result = CRAMER(*A, B* [, /DOUBLE] [, ZERO=*value*])

CREATE_STRUCT - Creates and concatenates structures.

Result = CREATE_STRUCT([*Tag_1*, *$Value_1$*, ..., *Tag_n*, *$Value_n$*])

or

Result = CREATE_STRUCT(NAME=*string, [Tag_1, ..., Tag_n], $Value_1$, ..., $Value_n$*)

CREATE_VIEW - Sets up 3D transformations.

CREATE_VIEW [, AX=*value*] [, AY=*value*] [, AZ=*value*] [, PERSP=*value*] [, /RADIANS] [, WINX=*pixels*] [, WINY=*pixels*] [, XMAX=*scalar*] [, XMIN=*scalar*] [, YMAX=*scalar*] [, YMIN=*scalar*] [, ZFAC=*value*] [, ZMAX=*scalar*] [, ZMIN=*scalar*] [, ZOOM=*scalar or 3-element vector*]

CROSSP - Computes vector cross product.

Result = CROSSP(*V1*, *V2*)

CRVLENGTH - Computes the length of a curve.

Result = CRVLENGTH(*X, Y* [, /DOUBLE])

CT_LUMINANCE - Calculates the luminance of colors.

Result = CT_LUMINANCE([*R, G, B*] [, BRIGHT=*variable*] [, DARK=*variable*] [, /READ_TABLES])

CTI_TEST - Performs chi-square goodness-of-fit test.

Result = CTI_TEST(*Obfreq* [, COEFF=*variable*] [, /CORRECTED] [, CRAMV=*variable*] [, DF=*variable*] [, EXFREQ=*variable*] [, RESIDUAL=*variable*])

CURSOR - Reads position of the interactive graphics cursor.

CURSOR, *X, Y* [, *Wait* | [, /CHANGE | , /DOWN | , /UP | , /WAIT]] [, /DATA | , /DEVICE, | , /NORMAL]

CURVEFIT - Computes a non-linear least squares fit.

Result = CURVEFIT(*X, Y, Weights, A* [, *Sigma*] [, CHISQ=*variable*] [, FUNCTION_NAME=*string*] [, ITER=*variable*] [, ITMAX=*value*] [, /NODERIVATIVE] [, TOL=*value*])

CV_COORD - Converts 2D and 3D coordinates between coordinate systems.

Result = CV_COORD([, /DEGREES] [, FROM_CYLIN=*cyl_coords* | , FROM_POLAR=*pol_coords* | , FROM_RECT=*rect_coords* | , FROM_SPHERE=*sph_coords*] [, /TO_CYLIN | , /TO_POLAR | , /TO_RECT | , /TO_SPHERE])

CVTTOBM - Creates a bitmap byte array for a button label.

Result = CVTTOBM(*Array* [, THRESHOLD=*value*{0 to 255}])

CW_ANIMATE - Creates a compound widget for animation.

Result = CW_ANIMATE(*Parent*, *Sizex*, *Sizey*, *Nframes* [, /NO_KILL] [, OPEN_FUNC=*string*] [, PIXMAPS=*vector*] [, /TRACK] [, UNAME=*string*] [, UVALUE=*value*])

CW_ANIMATE_GETP - Gets pixmap window IDs used by CW_ANIMATE.

CW_ANIMATE_GETP, *Widget*, *Pixmaps* [, /KILL_ANYWAY]

CW_ANIMATE_LOAD - Loads images into CW_ANIMATE.

CW_ANIMATE_LOAD, *Widget* [, /CYCLE] [, FRAME=*value*{0 to *NFRAMES*}] [, IMAGE=*value*] [, /ORDER] [, WINDOW=*[window_num* [, *X0*, *Y0*, *Sx*, *Sy]]*] [, XOFFSET=*pixels*] [, YOFFSET=*pixels*]

CW_ANIMATE_RUN - Displays images loaded into CW_ANIMATE.
CW_ANIMATE_RUN, *Widget* [, *Rate*{0 to 100}] [, NFRAMES=*value*] [, /STOP]

CW_ARCBALL - Creates compound widget for intuitively specifying 3D orientations.
Result = CW_ARCBALL(*Parent* [, COLORS=*array*] [, /FRAME] [, LABEL=*string*] [, RETAIN={0 | 1 | 2}] [, SIZE=*pixels*] [, /UPDATE] [, UNAME=*string*] [, UVALUE=*value*] [, VALUE=*array*])

CW_BGROUP - Creates button group for use as a menu.
Result = CW_BGROUP(*Parent*, *Names* [, BUTTON_UVALUE=*array*] [, COLUMN=*value*] [, EVENT_FUNCT=*string*] [{, /EXCLUSIVE | , /NONEXCLUSIVE} | [, SPACE=*pixels*] [, XPAD=*pixels*] [, YPAD=*pixels*]] [, FONT=*font*] [, FRAME=*width*] [, IDS=*variable*] [, /LABEL_LEFT | , /LABEL_TOP] [, /MAP] [, /NO_RELEASE] [, /RETURN_ID | , /RETURN_INDEX | , /RETURN_NAME] [, ROW=*value*] [, /SCROLL] [, X_SCROLL_SIZE=*width*] [, Y_SCROLL_SIZE=*height*]] [, SET_VALUE=*value*] [, UNAME=*string*] [, UVALUE=*value*] [, XOFFSET=*value*] [, XSIZE=*width*] [, YOFFSET=*value*] [, YSIZE=*value*])

CW_CLR_INDEX - Creates compound widget for the selection of a color index.
Result = CW_CLR_INDEX(*Parent* [, COLOR_VALUES=*vector* | [, NCOLORS=*value*] [, START_COLOR=*value*]] [, EVENT_FUNC='*function_name*'] [, /FRAME] [, LABEL=*string*] [, UNAME=*string*] [, UVALUE=*value*] [, XSIZE=*pixels*] [, YSIZE=*pixels*])

CW_COLORSEL - Creates compound widget that displays all colors in current colormap.
Result = CW_COLORSEL(*Parent* [, /FRAME] [, UNAME=*string*] [, UVALUE=*value*] [, XOFFSET=*value*] [, YOFFSET=*value*])

CW_DEFROI - Creates compound widget used to define region of interest.
Result = CW_DEFROI(*Draw* [, IMAGE_SIZE=*vector*] [, OFFSET=*vector*] [, /ORDER] [, /RESTORE] [, ZOOM=*vector*])

CW_DICE - Creates compound widget for a 6-sided die.
Result = CW_DICE(*Parent* [, TUMBLE_CNT=*value*] [, TUMBLE_PERIOD=*seconds*] [, UNAME=*string*] [, UVALUE=*value*])

CW_FIELD - Creates a widget data entry field.
Result = CW_FIELD(*Parent* [, /ALL_EVENTS] [, /COLUMN] [, FIELDFONT=*font*] [, /FLOATING | , /INTEGER | , /LONG | , /STRING] [, FONT=*string*] [, FRAME=*pixels*] [, /NOEDIT] [, /RETURN_EVENTS] [, /ROW] [, TITLE=*string*] [, UNAME=*string*] [, UVALUE=*value*] [, VALUE=*value*] [, XSIZE=*characters*] [, YSIZE=*lines*])

CW_FILESEL - Creates compound widget for file selection.
Result = CW_FILESEL (*Parent* [, /FILENAME] [, FILTER=*string array*] [, /FIX_FILTER] [, /FRAME] [, /IMAGE_FILTER] [, /MULTIPLE] [, PATH=*string*] [, UNAME=*string*] [, UVALUE=*value*])

CW_FORM - Creates compound widget for creating forms.
Result = CW_FORM([*Parent*,] *Desc* [, /COLUMN] [, IDS=*variable*] [, /NO_RELEASE] [, TITLE=*string*] [, UNAME=*string*] [, UVALUE=*value*])
Note: *Desc* is a string array. Each element of string array contains 2 or more comma-delimited fields. Each string has the following format: *['Depth*, *Item*, *Initial_Value*, *Keywords']*

CW_FSLIDER - Creates slider that selects floating-point values.
Result = CW_FSLIDER(*Parent* [, /DRAG] [, /EDIT] [, FORMAT=*string*] [, /FRAME] [, MAXIMUM=*value*] [, MINIMUM=*value*] [, SCROLL=*units*] [, /SUPRESS_VALUE] [, TITLE=*string*] [, UNAME=*string*] [, UVALUE=*value*] [, VALUE=*initial_value*] [, XSIZE=*length* | {, /VERTICAL [, YSIZE=*height*]}])

CW_LIGHT_EDITOR - Creates compound widget to edit properties of existing IDLgrLight objects in a view.
Result = CW_LIGHT_EDITOR (*Parent* [, /DIRECTION_DISABLED] [, /DRAG_EVENTS] [, FRAME=*width*] [, /HIDE_DISABLED] [, LIGHT=*objref(s)*] [, /LOCATION_DISABLED] [, /TYPE_DISABLED] [, UVALUE=*value*] [, XSIZE=*pixels*] [, YSIZE=*pixels*] [, XRANGE=*vector*] [, YRANGE=*vector*] [, ZRANGE=*vector*])

CW_LIGHT_EDITOR_GET - Gets the CW_LIGHT_EDITOR properties.
CW_ LIGHT_EDITOR_GET, *WidgetID* [, DIRECTION_DISABLED=*variable*] [, DRAG_EVENTS=*variable*] [, HIDE_DISABLED=*variable*] [, LIGHT=*variable*] [, LOCATION_DISABLED=*variable*] [, TYPE_DISABLED=*variable*] [, XSIZE=*variable*] [, YSIZE=*variable*] [, XRANGE=*variable*] [, YRANGE=*variable*][, ZRANGE=*variable*]

CW_LIGHT_EDITOR_SET - Sets the CW_LIGHT_EDITOR properties.
CW_ LIGHT_EDITOR_SET, *WidgetID* [, /DIRECTION_DISABLED] [, /DRAG_EVENTS] [, /HIDE_DISABLED] [, LIGHT=*objref(s)*] [, /LOCATION_DISABLED] [, /TYPE_DISABLED] [, XSIZE=*pixels*] [, YSIZE=*pixels*] [, XRANGE=*vector*] [, YRANGE=*vector*] [, ZRANGE=*vector*]

CW_ORIENT - Creates compound widget used to interactively adjust the 3D drawing transformation.
Result = CW_ORIENT(*Parent* [, AX=*degrees*] [, AZ=*degrees*] [, /FRAME] [, TITLE=*string*] [, UNAME=*string*] [, UVALUE=*value*] [, XSIZE=*width*] [, YSIZE=*height*])

CW_PALETTE_EDITOR - Creates compound widget to display and edit color palettes.
Result = CW_PALETTE_EDITOR (*Parent* [, DATA=*array*] [, FRAME=*width*] [, HISTOGRAM=*vector*] [, /HORIZONTAL] [, SELECTION=*[start, end]*] [, UNAME=*string*] [, UVALUE=*value*] [, XSIZE=*width*] [, YSIZE=*height*])

CW_PALETTE_EDITOR_GET - Gets the CW_PALETTE_EDITOR properties.
CW_PALETTE_EDITOR_GET, *WidgetID* [, ALPHA=*variable*] [, HISTOGRAM=*variable*])

CW_PALETTE_EDITOR_SET - Sets the CW_PALETTE_EDITOR properties.
CW_PALETTE_EDITOR_SET, *WidgetID* [, ALPHA=*byte_vector*] [, HISTOGRAM=*byte_vector*])

CW_PDMENU - Creates widget pulldown menus.
Result = CW_PDMENU(*Parent*, *Desc* [, DELIMITER=*string*] [, FONT=*value*] [, /MBAR [, /HELP]] [, IDS=*variable*] [, /RETURN_ID | , /RETURN_INDEX | , /RETURN_NAME | , /RETURN_FULL_NAME] [, UNAME=*string*] [, UVALUE=*value*] [, XOFFSET=*value*] [, YOFFSET=*value*])

CW_RGBSLIDER - Creates compound widget with sliders for adjusting RGB color values.
Result = CW_RGBSLIDER(*Parent* [, /CMY | , /HSV | , /HLS | , /RGB] [, /COLOR_INDEX] [, /DRAG] [, /FRAME] [, LENGTH=*value*] [, UNAME=*string*] [, UVALUE=*value*] [, /VERTICAL])

CW_TMPL - Template for compound widgets that use XMANAGER.
Result = CW_TMPL(*Parent* [, UNAME=*string*] [, UVALUE=*value*])

CW_ZOOM - Creates widget for displaying zoomed images.
Result = CW_ZOOM(*Parent* [, /FRAME] [, MAX=*scale*] [, MIN=*scale*] [, RETAIN={0 | 1 | 2}] [, SAMPLE=*value*] [, SCALE=*value*] [, /TRACK] [, UNAME=*string*] [, UVALUE=*value*] [, XSIZE=*width*] [, X_SCROLL_SIZE=*width*] [, X_ZSIZE=*zoom_width*] [, YSIZE=*height*] [, Y_SCROLL_SIZE=*height*] [, Y_ZSIZE=*zoom_height*])

D

DBLARR - Creates a double-precision array.
Result = DBLARR(D_1, ..., D_8 [, /NOZERO])

DCINDGEN - Returns a double-precision, complex array with each element set to its subscript.
Result = DCINDGEN(D_1, ..., D_8)

DCOMPLEX - Converts argument to double-precision complex type.
Result = DCOMPLEX(*Real* [, *Imaginary*])
or
Result = DCOMPLEX(*Expression*, *Offset*, Dim_1 [, ..., Dim_8])

DCOMPLEXARR - Creates a complex, double-precision vector or array.
Result = DCOMPLEXARR(D_1, ..., D_8 [, /NOZERO])

DEFINE_KEY - Programs keyboard function keys.
DEFINE_KEY, *Key* [, *Value*] [, /MATCH_PREVIOUS] [, /NOECHO] [, /TERMINATE]
UNIX Keywords: [, /BACK_CHARACTER] [, /BACK_WORD] [, /CONTROL | , /ESCAPE] [, /DELETE_CHARACTER] [, /DELETE_CURRENT] [, /DELETE_EOL] [, /DELETE_LINE] [, /DELETE_WORD] [, /END_OF_LINE] [, /END_OF_FILE] [, /ENTER_LINE] [, /FORWARD_CHARACTER] [, /FORWARD_WORD] [, /INSERT_OVERSTRIKE_TOGGLE] [, /NEXT_LINE] [, /PREVIOUS_LINE] [, /RECALL] [, /REDRAW] [, /START_OF_LINE]

DEFROI - Defines an irregular region of interest of an image.
Result = DEFROI(*Sx*, *Sy* [, *Xverts*, *Yverts*] [, /NOREGION] [, /NOFILL] [, /RESTORE] [, X0=*device_coord*, Y0=*device_coord*] [, ZOOM=*factor*])

DEFSYSV - Creates a new system variable.
DEFSYSV,*Name*,*Value* [, *Read_Only*] [, EXISTS=*variable*]

DELETE_SYMBOL (VMS Only) - Deletes a DCL interpreter symbol for the current process.
DELETE_SYMBOL, *Name* [, TYPE={1 | 2}]

DELLOG (VMS Only) - Deletes a VMS logical name.
DELLOG, *Lognam* [, TABLE=*string*]

DELVAR - Deletes variables from the main IDL program level.
DELVAR, V_1, ..., V_n

DERIV - Performs differentiation using 3-point, Lagrangian interpolation and returns the derivative.
Result = DERIV([*X*,] *Y*)

DERIVSIG - Computes standard deviation of derivative found by DERIV.
Result = DERIVSIG([*X*, *Y*, Sig_x,] Sig_y)

DETERM - Computes the determinant of a square matrix.
Result = DETERM(*A* [, /CHECK] [, /DOUBLE] [, ZERO=*value*])

DEVICE - Sets to plot in device coordinates.

Note: Each keyword to DEVICE is followed by the device(s) to which it applies.

DEVICE [, /AVANTGARDE | , /BKMAN | , /COURIER | , /HELVETICA | , /ISOLATIN1 | , /PALATINO | , /SCHOOLBOOK | , /SYMBOL | , TIMES | , ZAPFCHANCERY | , ZAPFDINGBATS {PS}]
[, /AVERAGE_LINES{REGIS}]
[, /BINARY | [, /NCAR | , TEXT {CGM}]
[, BITS_PER_PIXEL={1 | 2 | 4 | 8}{PS}]
[, /BOLD{PS}] [, /BOOK{PS}]
[, /BYPASS_TRANSLATION{MAC, WIN, X}]
[, /CLOSE{Z}] [, /CLOSE_DOCUMENT{PRINTER}]
[, /CLOSE_FILE{CGM, HP, LJ, PCL, PS, REGIS, TEK}]
[, /COLOR{PCL, PS}] [, COLORS=*value*{CGM, TEK}]
[, COPY=*[Xsource, Ysource, cols, rows, Xdest, Ydest*
[, *Window_index*]*]*{MAC, WIN, X}]
[, /CURSOR_CROSSHAIR{WIN, X}]
[, CURSOR_IMAGE=*value*{16-element short int vector}{X}] [, CURSOR_MASK=*value*{X}]
[, /CURSOR_ORIGINAL{MAC, WIN, X}]
[, CURSOR_STANDARD=*value*{MAC: crosshair=1}{WIN: arrow=32512, I-beam=32513, hourglass=32514, black cross=32515, up arrow=32516, size(NT)=32640, icon(NT)=32641, size NW-SE=32642, size NE-SW=32643, size E-W=32644, size N-S=32645}{X: one of the values in file cursorfonts.h}]
[, CURSOR_XY=*[x,y]*{(X)] [, /DECOMPOSED{MAC, WIN, X}] [, DEPTH=*value*{significant bits per pixel}{LJ}] [, /DIRECT_COLOR{X}] [, EJECT={0 | 1 | 2}{HP}] [, ENCAPSULATED={0 | 1}{PS}]
[, ENCODING={1 (binary) | 2 (text) | 3 (NCAR binary)}{CGM}] [, FILENAME=*filename*{CGM, HP, LJ, PCL, PS, REGIS, TEK}] [, /FLOYD{LJ, MAC, PCL, X}]
[, FONT_INDEX=*integer*{PS}]
[, FONT_SIZE=*points*{PS}]
[, GET_CURRENT_FONT=*variable*{MAC, PRINTER, WIN, X}] [, GET_DECOMPOSED=*variable*{MAC, WIN, X}] [, GET_FONTNAMES=*variable*{MAC, PRINTER, WIN, X}]
[, GET_FONTNUM=*variable*{MAC, PRINTER, WIN, X}] [, GET_GRAPHICS_FUNCTION=*variable*{MAC, WIN, X, Z}] [, GET_SCREEN_SIZE=*variable*{MAC, WIN, X}] [, GET_VISUAL_DEPTH=*variable*{MAC, WIN, X}] [, GET_VISUAL_NAME=*variable*{MAC, WIN, X}]
[, GET_WINDOW_POSITION=*variable*{MAC, WIN, X}] [, GET_WRITE_MASK=*variable*{X, Z}]
[, GIN_CHARS=*number_of_characters*{TEK}]
[, GLYPH_CACHE=*number_of_glyphs*{MAC, PRINTER, PS, WIN, Z}] [, /INCHES{HP, LJ, PCL, PRINTER, PS}]
[, /INDEX_COLOR{PRINTER}] [, /ITALIC{PS}]
[, /LANDSCAPE | , /PORTRAIT{HP, LJ, PCL, PRINTER, PS}] [, /DEMI | , /LIGHT | , /MEDIUM | , /NARROW | , /OBLIQUE{PS}] [, OPTIMIZE={0 | 1 | 2}{PCL}]
[, /ORDERED{LJ, MAC, PCL, X}] [, OUTPUT=*scalar string*{HP, PS}] [, /PIXELS{LJ, PCL}]
[, PLOT_TO=*logical unit num*{REGIS, TEK}]
[, /PLOTTER_ON_OFF{HP}] [, /POLYFILL{HP}]
[, /PREVIEW{PS}] [, PRINT_FILE=*filename*{WIN}]
[, /PSEUDO_COLOR{MAC, X}]
[, RESET_STRING=*string*{TEK}]
[, RESOLUTION=*value*{LJ, PCL}]
[, RETAIN={0 | 1 | 2}{MAC, WIN, X}]
[, SCALE_FACTOR=*value*{PRINTER, PS}]
[, SET_CHARACTER_SIZE=*[font size, line spacing]*{CGM, HP, LJ, MAC, PCL, PS, REGIS, TEK, WIN, X, Z}] [, SET_COLORMAP=*value*{14739-element byte vector}{PCL}] [, SET_COLORS=*value*{2 to 256}{Z}] [, SET_FONT=*scalar string*{MAC, PRINTER, PS, WIN, Z}] [, SET_GRAPHICS_FUNCTION=*code*{0 to 15}{MAC, WIN, X, Z}]
[, SET_RESOLUTION=*[width, height]*{Z}]
[, SET_STRING=*string*{TEK}]
[, SET_TRANSLATION=*variable*{X}]
[, SET_WRITE_MASK=*value*{0 to 2^n-1 for *n*-bit system}{X, Z}] [, STATIC_COLOR=*value*{bits per pixel}{X}] [, STATIC_GRAY=*value*{bits per pixel}{X}]
[, /TEK4014{TEK}] [, TEK4100{TEK}]
[, THRESHOLD=*value*{LJ, MAC, PCL, X}]
[, TRANSLATION=*variable*{MAC, WIN, X}]
[, TRUE_COLOR=*value*{bits per pixel}{MAC, PRINTER, X}] [, /TTY{REGIS, TEK}]
[, /VT240 | , /VT241 | , /VT340 | , /VT341 {REGIS}]
[, WINDOW_STATE=*variable*{WIN, X}]
[, XOFFSET=*value*{HP, LJ, PCL, PRINTER, PS}]
[, XON_XOFF={0 | 1 (default)}{HP}]
[, XSIZE=*width*{HP, LJ, PCL, PRINTER, PS}]
[, YOFFSET=*value*{HP, LJ, PCL, PRINTER, PS}]
[, YSIZE=*height*{HP, LJ, PCL, PRINTER, PS}]
[, Z_BUFFERING={0 | 1 (default)}{Z}]

DFPMIN - Minimizes a function using Davidon-Fletcher-Powell method.

DFPMIN, *X*, *Gtol*, *Fmin*, *Func*, *Dfunc* [, /DOUBLE]
[, EPS=*value*] [, ITER=*variable*] [, ITMAX=*value*]
[, STEPMAX=*value*] [, TOLX=*value*]

DIALOG_MESSAGE - Creates modal message dialog.

Result = DIALOG_MESSAGE(*Message_Text*
[, /CANCEL] [, /DEFAULT_CANCEL | , /DEFAULT_NO] [, DIALOG_PARENT=*widget_id*]
[, DISPLAY_NAME=*string*] [, /ERROR | , /INFORMATION | , /QUESTION]
[, RESOURCE_NAME=*string*] [, TITLE=*string*])

DIALOG_PICKFILE - Creates native file-selection dialog.
Result = DIALOG_PICKFILE([, /DIRECTORY]
[, DIALOG_PARENT=*widget_id*]
[, DISPLAY_NAME=*string*] [, FILE=*string*]
[, FILTER=*string*] [, /FIX_FILTER]
[, GET_PATH=*variable*] [, GROUP=*widget_id*]
[, /MULTIPLE_FILES] [, /MUST_EXIST]
[, /NOCONFIRM] [, PATH=*string*] [, /READ | , /WRITE]
[, /RESOURCE_NAME] [, TITLE=*string*])

DIALOG_PRINTERSETUP - Opens native dialog used to set properties for a printer.
Result = DIALOG_PRINTERSETUP([*PrintDestination*]
[, DIALOG_PARENT=*widget_id*]
[, DISPLAY_NAME=*string*]
[, RESOURCE_NAME=*string*] [, TITLE=*string*])

DIALOG_PRINTJOB - Opens native dialog used to set parameters for a print job.
Result = DIALOG_PRINTJOB([*PrintDestination*]
[, DIALOG_PARENT=*widget_id*]
[, DISPLAY_NAME=*string*]
[, RESOURCE_NAME=*string*] [, TITLE=*string*])

DIALOG_READ_IMAGE - Presents GUI for reading image files.
Result = DIALOG_READ_IMAGE ([*Filename*]
[, DIALOG_PARENT=*widget_id*] [, FILE=*variable*]
[, FILTER=*string*] [, /FIX_FILTER] [, IMAGE=*variable*]
[, PATH=*string*] [, QUERY=*variable*] [, RED=*variable*]
[, GREEN=*variable*] [, BLUE=*variable*]
[,TITLE=*string*])

DIALOG_WRITE_IMAGE - Presents GUI for writing image files.
Result = DIALOG_WRITE_IMAGE (*Image* [, *R*, *G*, *B*]
[, DIALOG_PARENT=*widget_id*] [, FILENAME=*string*]
[, /FIX_TYPE] [, /NOWRITE] [, OPTIONS=*variable*]
[, PATH=*string*] [,TITLE=*string*] [, TYPE=*variable*])

DIGITAL_FILTER - Calculates coefficients of a non-recursive, digital filter.
Result = DIGITAL_FILTER(*Flow*, *Fhigh*, *A*, *Nterms*)

DILATE - Implements morphologic dilation operator on binary and grayscale images.
Result = DILATE(*Image*, *Structure* [, X_0 [, Y_0 [, Z_0]]]
[, /CONSTRAINED [, BACKGROUND=*value*]] [, /GRAY
[, /PRESERVE_TYPE | , /UINT | , /ULONG]]
[, VALUES=*array*])

DINDGEN - Returns a double-precision array with each element set to its subscript.
Result = DINDGEN(D_1, ..., D_8)

DISSOLVE - Provides a digital “dissolve” effect for images.
DISSOLVE, *Image* [, WAIT=*seconds*] [, /ORDER]
[, SIZ=*pixels*] [, X0=*pixels*, Y0=*pixels*]

DIST - Creates array with each element proportional to its frequency.
Result = DIST(*N* [, *M*])

DLM_LOAD - Explicitly causes a DLM to be loaded.
DLM_LOAD, $DLMNameStr_1$
[, $DLMNameStr_2$,..., $DLMNameStr_n$]

DO_APPLE_SCRIPT - Compiles, executes AppleScript code.
DO_APPLE_SCRIPT, *Script* [, /AG_STRING]
[, RESULT=*variable*]

DOC_LIBRARY - Extracts documentation headers from IDL programs.
DOC_LIBRARY [, *Name*] [, /PRINT]
UNIX keywords: [, DIRECTORY=*string*] [, /MULTI]
VMS keywords: [, /FILE] [, PATH=*string*] [, /OUTPUTS]

DOUBLE - Converts argument to double-precision type.
Result = DOUBLE(*Expression*[, *Offset* [, Dim_1, ..., Dim_n]])

DRAW_ROI - Draws region or group of regions to current Direct Graphics device.
DRAW_ROI, *oROI* [, /LINE_FILL] [, SPACING=*value*]
Graphics Keywords: [, CLIP=*[X_0, Y_0, X_1, Y_1]*]
[, COLOR=*value*] [, /DATA | , /DEVICE | , /NORMAL]
[, LINESTYLE={0 | 1 | 2 | 3 | 4 | 5}] [, /NOCLIP]
[, ORIENTATION=*ccw_degrees_from_horiz*]
[, PSYM=*integer*{0 to 10}] [, SYMSIZE=*value*] [, /T3D]
[, THICK=*value*]

E

EFONT - Interactive vector font editor and display tool.
EFONT, *Init_Font* [, /BLOCK] [, GROUP=*widget_id*]

EIGENQL - Computes eigenvalues and eigenvectors of a real, symmetric array.
Result = EIGENQL(*A* [, /ABSOLUTE] [, /ASCENDING]
[, /DOUBLE] [, EIGENVECTORS=*variable*]
[, /OVERWRITE | , RESIDUAL=*variable*])

EIGENVEC - Computes eigenvectors of a real, non-symmetric array.
Result = EIGENVEC(*A*, *Eval* [, /DOUBLE]
[, ITMAX=*value*] [, RESIDUAL=*variable*])

ELMHES - Reduces nonsymmetric array to upper Hessenberg form.
Result = ELMHES(*A* [, /COLUMN] [, /DOUBLE]
[, /NO_BALANCE])

EMPTY - Empties the graphics output buffer.
EMPTY

ENABLE_SYSRTN - Enables/disables IDL system routines.
ENABLE_SYSRTN [, *Routines*] [, /DISABLE]
[, /EXCLUSIVE] [, /FUNCTIONS]

EOF - Tests the specified file for the end-of-file condition.
Result = EOF(*Unit*)

EOS_* Routines - See "EOS Routines" on page 39.

ERASE - Erases the screen of the current graphics device, or starts a new page if the device is a printer.

ERASE [, *Background_Color*] [, CHANNEL=*value*] [, COLOR=*value*]

ERODE - Implements the erosion operator on binary and grayscale images and vectors.

Result = ERODE(*Image*, *Structure* [, X_0 [, Y_0 [, Z_0]]] [, /GRAY [, /PRESERVE_TYPE | , /UINT | , /ULONG]] [, VALUES=*array*])

ERRORF - Returns the value of an error function.

Result = ERRORF(*X*)

ERRPLOT - Plots error bars over a previously drawn plot.

ERRPLOT, [*X*,] *Low*, *High* [, WIDTH=*value*]

EXECUTE - Compiles and executes IDL statements contained in a string.

Result = EXECUTE(*String* [, *QuietCompile*])

EXIT - Quits IDL and exits back to the operating system.

EXIT [, /NO_CONFIRM] [, STATUS=*code*]

EXP - Returns the natural exponential function of *Expression*.

Result = EXP(*Expression*)

EXPAND - Shrinks/expands image using bilinear interpolation.

EXPAND, *A*, *Nx*, *Ny*, *Result* [, FILLVAL=*value*] [, MAXVAL=*value*]

EXPAND_PATH - Expands path-definition string into full path name for use with the !PATH system variable.

Result = EXPAND_PATH(*String* [, /ARRAY] [, COUNT=*variable*])

EXPINT - Returns the value of the exponential integral.

Result = EXPINT(*N*, *X* [, /DOUBLE] [, EPS=*value*] [, ITMAX=*value*])

EXTRAC - Returns sub-matrix of input array. Array operators (e.g., * and :) should usually be used instead.

Result = EXTRAC(*Array*, C_1, C_2, ..., C_n, S_1, S_2, ..., S_n)

EXTRACT_SLICE - Returns 2D planar slice extracted from volume.

Result = EXTRACT_SLICE(*Vol*, *Xsize*, *Ysize*, *Xcenter*, *Ycenter*, *Zcenter*, *Xrot*, *Yrot*, *Zrot* [, ANISOTROPY=*[xspacing, yspacing, zspacing]*] [, OUT_VAL=*value*] [, /RADIANS] [, /SAMPLE] [, VERTICES=*variable*])

or

Result = EXTRACT_SLICE(*Vol*, *Xsize*, *Ysize*, *Xcenter*, *Ycenter*, *Zcenter*, *PlaneNormal*, *Xvec* [, ANISOTROPY=*[xspacing, yspacing, zspacing]*] [, OUT_VAL=*value*] [, /RADIANS] [, /SAMPLE] [, VERTICES=*variable*])

F

F_CVF - Computes the cutoff value in an F distribution.

Result = F_CVF(*P*, *Dfn*, *Dfd*)

F_PDF - Computes the F distribution function.

Result = F_PDF(*V*, *Dfn*, *Dfd*)

FACTORIAL - Computes the factorial *N*!.

Result = FACTORIAL(*N* [, /STIRLING])

FFT - Returns the Fast Fourier Transform of *Array*.

Result = FFT(*Array* [, *Direction*] [, /DOUBLE] [, /INVERSE] [, /OVERWRITE])

FILEPATH - Returns full path to a file in the IDL distribution.

Result = FILEPATH(*Filename* [, ROOT_DIR=*string*] [, SUBDIRECTORY=*string/string_array*] [, /TERMINAL] [, /TMP])

FINDFILE - Finds all files matching *File_Specification*.

Result = FINDFILE(*File_Specification* [, COUNT=*variable*])

FINDGEN - Returns a floating-point array with each element set to its subscript.

Result = FINDGEN(D_1, ..., D_8)

FINITE - Returns True if its argument is finite.

Result = FINITE(*X* [, /INFINITY] [, /NAN])

FIX - Converts argument to integer type, or type specified by TYPE keyword.

Result = FIX(*Expression* [, *Offset* [, Dim_1, ..., Dim_8]] [, /PRINT] [, TYPE=*type code*{0 to 15}])

FLICK - Causes the display to flicker between two images.

FLICK, *A*, *B* [, *Rate*]

FLOAT - Converts argument to single-precision floating-point.

Result = FLOAT(*Expression* [, *Offset* [, Dim_1, ..., Dim_8]])

FLOOR - Returns closest integer less than or equal to argument.

Result = FLOOR(*X*)

FLOW3 - Draws lines representing a 3D flow/velocity field.

FLOW3, *Vx*, *Vy*, *Vz* [, ARROWSIZE=*value*] [, /BLOB] [, LEN=*value*] [, NSTEPS=*value*] [, NVECS=*value*] [, SX=*vector*, SY=*vector*, SZ=*vector*]

FLTARR - Returns a single-precision, floating-point vector or array.

Result = FLTARR(D_1, ..., D_8 [, /NOZERO])

FLUSH - Flushes file unit buffers.

FLUSH, $Unit_1$, ..., $Unit_n$

FOR - Executes one or more statements repeatedly, incrementing or decrementing a variable with each repetition, until a condition is met.

FOR *variable* = *init*, *limit* [, *Increment*] DO *statement*

or

FOR *variable* = *init*, *limit* [, *Increment*] DO BEGIN

statements
ENDFOR

FORMAT_AXIS_VALUES - Formats numbers as strings for use as axis values.
Result = FORMAT_AXIS_VALUES(*Values*)

FORWARD_FUNCTION - Causes argument(s) to be interpreted as functions rather than variables (versions of IDL prior to 5.0 used parentheses to declare arrays).
FORWARD_FUNCTION $Name_1$, $Name_2$, ..., $Name_n$

FREE_LUN - Frees previously-reserved file units.
FREE_LUN, $Unit_1$, ..., $Unit_n$

FSTAT - Returns information about a specified file unit.
Result = FSTAT(*Unit*)

FULSTR - Restores a sparse matrix to full storage mode.
Result = FULSTR(*A*)

FUNCT - Evaluates sum of a Gaussian and a 2nd-order polynomial and returns value of its partial derivatives.
FUNCT, *X, A, F* [, *Pder*]

FUNCTION - Defines a function.
FUNCTION *Function_Name*, $parameter_1$, ..., $parameter_n$

FV_TEST - Performs the F-variance test.
Result = FV_TEST(*X, Y*)

FX_ROOT - Computes real and complex roots of a univariate nonlinear function using an optimal Müller's method.
Result = FX_ROOT(*X, Func* [, /DOUBLE] [, ITMAX=*value*] [, /STOP] [, TOL=*value*])

FZ_ROOTS - Finds the roots of a complex polynomial using Laguerre's method.
Result = FZ_ROOTS(*C* [, /DOUBLE] [, EPS=*value*] [, /NO_POLISH])

G

GAMMA - Returns the gamma function of *X*.
Result = GAMMA(*X*)

GAMMA_CT - Applies gamma correction to a color table.
GAMMA_CT, *Gamma* [, /CURRENT] [, /INTENSITY]

GAUSS_CVF - Computes cutoff value in Gaussian distribution.
Result = GAUSS_CVF(*P*)

GAUSS_PDF - Computes Gaussian distribution function.
Result = GAUSS_PDF(*V*)

GAUSS2DFIT - Fits a 2D elliptical Gaussian equation to rectilinearly gridded data.
Result = GAUSS2DFIT(*Z, A* [, *X, Y*] [, /NEGATIVE] [, /TILT])

GAUSSFIT - Fits the sum of a Gaussian and a quadratic.
Result = GAUSSFIT(*X, Y* [, *A*] [, ESTIMATES=*array*] [, NTERMS=*integer*{3 to 6}])

GAUSSINT - Returns integral of Gaussian probability function.
Result = GAUSSINT(*X*)

GET_DRIVE_LIST - Returns string array of the names of valid drives/volumes for the file system. (Windows / Macintosh only)
Result = GET_DRIVE_LIST( )

GET_KBRD - Gets one input IDL character.
Result = GET_KBRD(*Wait*)

GET_LUN - Reserves a logical unit number (file unit).
GET_LUN, *Unit*

GET_SCREEN_SIZE - Returns dimensions of the screen.
Result = GET_SCREEN_SIZE([*Display_name*] [, RESOLUTION=*variable*])
X Windows Keywords: [, DISPLAY_NAME=*string*]

GET_SYMBOL (VMS Only) - Returns value of a VMS DCL interpreter symbol.
Result = GET_SYMBOL(*Name* [, TYPE={1 | 2}])

GETENV - Returns the value of an environment variable.
Result = GETENV(*Name*)

GOTO - Transfers program control to point specified by *label*.
GOTO, *label*

GRID_TPS - Uses thin plate splines to interpolate a set of values over a regular 2D grid, from irregularly sampled data values.
Interp = GRID_TPS (*Xp, Yp, Values* [, COEFFICIENTS=*variable*] [, NGRID = *[nx, ny]*] [, START = *[x0, y0]*] [, DELTA = *[dx, dy]*])

GRID3 - Creates a regularly-gridded 3D dataset from a set of scattered 3D nodes.
Result = GRID3(*X, Y, Z, F, Gx, Gy, Gz* [, DELTA=*scalar/vector*] [, DTOL=*value*] [, GRID=*value*] [, NGRID=*value*] [, START=*[x, y, z]*])

GS_ITER - Solves linear system using Gauss-Seidel iteration.
Result = GS_ITER(*A, B* [, /CHECK] [, LAMBDA=*value*{0.0 to 2.0}] [, MAX_ITER=*value*] [, TOL=*value*] [, X_0=*vector*])

H

H_EQ_CT - Histogram-equalizes the color tables for an image or a region of the display.
H_EQ_CT [, *Image*]

H_EQ_INT - Interactively histogram-equalizes the color tables of an image or a region of the display.
H_EQ_INT [, *Image*]

HANNING - Creates Hanning and Hamming windows.
Result=HANNING(N_1 [, N_2] [, ALPHA=*value*{0.5 to 1.0}])

HDF_* Routines - See "HDF Routines" on page 44.

HDF_BROWSER - Opens GUI to view contents of HDF, HDF-EOS, or NetCDF file.

Template = HDF_BROWSER([*Filename*] [, CANCEL=*variable*] [, GROUP=*widget_id*] [, PREFIX=*string*])

HDF_READ - Extracts HDF, HDF-EOS, and NetCDF data and metadata into an output structure.

Result = HDF_READ([*Filename*] [, DFR8=*variable*] [, DF24=*variable*] [, PREFIX=*string*] [, TEMPLATE =*value*])

HEAP_GC - Performs garbage collection on heap variables.

HEAP_GC [, /OBJ | , /PTR] [, /VERBOSE]

HELP - Provides information about the current IDL session.

HELP, *Expression*$_1$, ..., *Expression*$_n$ [, /ALL_KEYS] [, /BREAKPOINTS] [, /BRIEF] [, CALLS=*variable*] [, /DEVICE] [, /DLM] [, /FILES] [, /FULL] [, /FUNCTIONS] [, /HEAP_VARIABLES] [, /KEYS] [, /LAST_MESSAGE] [, /MEMORY] [, /MESSAGES] [, NAMES=*string_of_variable_names*] [, /OBJECTS] [, OUTPUT=*variable*] [, /PROCEDURES] [, /RECALL_COMMANDS] [, /ROUTINES] [, /SOURCE_FILES] [, /STRUCTURES] [, /SYSTEM_VARIABLES] [, /TRACEBACK]

HILBERT - Constructs a Hilbert transform.

Result = HILBERT(*X* [, *D*])

HIST_2D - Returns histogram of two variables.

Result = HIST_2D(V_1, V_2 [, BIN1=*width*] [, BIN2=*height*] [, MAX1=*value*] [, MAX2=*value*] [, MIN1=*value*] [, MIN2=*value*])

HIST_EQUAL - Histogram-equalizes an image.

Result = HIST_EQUAL(*A* [, BINSIZE=*value*] [, MINV=*value*] [, MAXV=*value*] [, TOP=*value*])

HISTOGRAM - Computes the density function of an array.

Result = HISTOGRAM(*Array* [, BINSIZE=*value*] [, INPUT=*variable*] [, MAX=*value*] [, MIN=*value*] [, /NAN] [, OMAX=*variable*] [, OMIN=*variable*] [, REVERSE_INDICES=*variable*])

HLS - Creates color table in Hue, Lightness, Saturation color system.

HLS, *Litlo*, *Lithi*, *Satlo*, *Sathi*, *Hue*, *Loops* [, *Colr*]

HQR - Returns all eigenvalues of an upper Hessenberg array.

Result = HQR(*A* [, /COLUMN] [, /DOUBLE])

HSV - Creates color table based on Hue and Saturation Value color system.

HSV, *Vlo*, *Vhi*, *Satlo*, *Sathi*, *Hue*, *Loops* [, *Colr*]

I

IBETA - Computes the incomplete beta function.

Result = IBETA(*A*, *B*, *X*)

IDENTITY - Returns an identity array.

Result = IDENTITY(*N* [, /DOUBLE])

IDL_Container Object - See "IDL_Container" on page 50.

IDLanROI Object - See "IDLanROI" on page 50.

IDLanROIGroup Object - See "IDLanROIGroup" on page 51.

IDLffDICOM Object - See "IDLffDICOM" on page 51.

IDLffDXF Object - See "IDLffDXF" on page 52.

IDLffLanguageCat Object - See "IDLffLanguageCat" on page 52

IDLgr* Objects - IDLgr* objects and their methods are described starting with "IDLgrAxis" on page 52.

IF...THEN...ELSE - Conditionally executes a statement or block of statements.

IF *expression* THEN *statement* [ELSE *statement*]

or

IF *expression* THEN BEGIN
  statements
ENDIF [ELSE BEGIN
  statements
ENDELSE]

IGAMMA - Computes the incomplete gamma function.

Result = IGAMMA(*A*, *X* [, METHOD=*variable*])

IMAGE_CONT - Overlays an image with a contour plot.

IMAGE_CONT, *A* [, /ASPECT] [, /INTERP] [, /WINDOW_SCALE]

IMAGE_STATISTICS - Computes sample statistics for a given array of values.

IMAGE_STATISTICS, *Data* [, /LABELED | [, /WEIGHTED] [, WEIGHT_SUM=*variable*]] [, /VECTOR] [, LUT=*array*] [, MASK=*array*] [, COUNT=*variable*] [, MEAN=*variable*] [, STDDEV=*variable*] [, DATA_SUM=*variable*] [, SUM_OF_SQUARES=*variable*] [, MINIMUM=*variable*] [, MAXIMUM=*variable*] [, VARIANCE=*variable*]

IMAGINARY - Returns the imaginary part of a complex value.

Result = IMAGINARY(*Complex_Expression*)

INDGEN - Return an integer array with each element set to its subscript.

Result = INDGEN(D_1, ..., D_8) [, /BYTE | , /COMPLEX | , /DCOMPLEX | , /DOUBLE | , /FLOAT | , /L64 | , /LONG | , /STRING | , /UINT | , /UL64 | , /ULONG] [, TYPE=*value*]

INT_2D - Computes the double integral of a bivariate function.

Result = INT_2D(*Fxy*, *AB_Limits*, *PQ_Limits*, *Pts* [, /DOUBLE] [, /ORDER])

INT_3D - Computes the triple integral of a trivariate function.
Result = INT_3D(*Fxyz*, *AB_Limits*, *PQ_Limits*, *UV_Limits*, *Pts* [, /DOUBLE])

INT_TABULATED - Integrates a tabulated set of data.
Result = INT_TABULATED(*X*, *F* [, /SORT])

INTARR - Creates an integer vector or array.
Result = INTARR(D_1, ..., D_8 [, /NOZERO])

INTERPOL - Performs linear interpolation on vectors.
For regular grids: *Result* = INTERPOL(V, N [, /LSQUADRATIC] [, /QUADRATIC] [, /SPLINE])
For irregular grids: *Result* = INTERPOL(V, X, U [, /LSQUADRATIC] [, /QUADRATIC] [, /SPLINE])

INTERPOLATE - Returns an array of interpolates.
Result = INTERPOLATE(*P, X* [, *Y* [, *Z*]] [, CUBIC=*value*{-1 to 0}] [, /GRID] [, MISSING=*value*])

INVERT - Computes the inverse of a square array.
Result = INVERT(*Array* [, *Status*] [, /DOUBLE])

IOCTL - Performs special functions on UNIX files.
Result = IOCTL(*File_Unit* [, *Request*, *Arg*] [, /BY_VALUE] [, /MT_OFFLINE] [, /MT_REWIND] [, MT_SKIP_FILE=[-]*number_of_files*] [, MT_SKIP_RECORD=[-]*number_of_records*] [, /MT_WEOF] [, /SUPRESS_ERROR])

ISHFT - Performs integer bit shift.
Result = ISHFT(P_1, P_2)

ISOCONTOUR - Interprets the contouring algorithm found in the IDLgrContour object.
ISOCONTOUR, *Values*, *Outconn*, *Outverts* [, AUXDATA_IN=*array*, AUXDATA_OUT=*variable*] [, C_VALUE=*vector*] [, GEOMX=*vector*] [, GEOMY=*vector*] [, GEOMZ=*vector*] [, /FILL] [, LEVEL_VALUES=*variable*] [, N_LEVELS=*levels*] [, /OUTCONN_INDICES] [, POLYGONS=*array of polygon descriptions*]

ISOSURFACE - Returns topologically consistent triangles by using oriented tetrahedral decomposition.
ISOSURFACE, *Data*, *Value*, *Outverts*, *Outconn* [, GEOM_XYZ=*array*, TETRAHEDRA=*array*] [, AUXDATA_IN=*array*, AUXDATA_OUT=*variable*]

J

JOURNAL - Logs IDL commands to a file.IDL.
JOURNAL [, *Arg*]

JULDAY - Returns Julian Day Number for given month, day, and year.
Result = JULDAY(*Month*,*Day*,*Year*,*Hour*,*Minute*,*Second*)

K

KEYWORD_SET - Returns True if *Expression* is defined and non-zero or an array.
Result = KEYWORD_SET(*Expression*)

KRIG2D - Interpolates set of points using kriging.
Result = KRIG2D(*Z* [, *X*, *Y*] [, EXPONENTIAL=*vector*] [, SPHERICAL=*vector*] [, A=*value*] [, C0=*value*] [, C1=*value*] [, /REGULAR] [, XGRID=*[xstart, xspacing]*] [, XVALUES=*array*] [, YGRID=*[ystart, yspacing]*] [, YVALUES=*array*] [, GS=*[xspacing, yspacing]*] [, BOUNDS=*[xmin, ymin, xmax, ymax]*] [, NX=*value*] [, NY=*value*])

KURTOSIS - Computes statistical kurtosis of *n*-element vector.
Result = KURTOSIS(*X* [, /DOUBLE] [, /NAN])

KW_TEST - Performs Kruskal-Wallis H-test.
Result = KW_TEST(*X* [, DF=*variable*] [, MISSING=*nonzero_value*])

L

L64INDGEN - Returns a 64-bit integer array with each element set to its subscript.
Result = L64INDGEN(D_1, ..., D_8)

LABEL_DATE - Labels axes with dates. Use with [XYZ]TICKFORMAT keyword.
Result = LABEL_DATE(DATE_FORMAT = *String* [, MONTHS=*string_array*])
or
PLOT, *x*, *y*, XTICKFORMAT = 'LABEL_DATE'

LABEL_REGION - Labels regions (blobs) of a bi-level image.
Result = LABEL_REGION(*Data* [, /ALL_NEIGHBORS] [, /ULONG])

LADFIT - Fits using least absolute deviation method.
Result = LADFIT(*X*, *Y* [, ABSDEV=*variable*] [, /DOUBLE])

LEEFILT - Performs the Lee filter algorithm on an image array.
Result = LEEFILT(*A* [, *N* [, *Sig*]] [, /EXACT])

LINBCG - Solves a set of linear equations using the iterative biconjugate gradient method. Use with SPRSIN.
Result = LINBCG(*A*, *B*, *X* [, /DOUBLE] [, ITOL={4 | 5 | 6 | 7}] [, TOL=*value*] [, ITER=*variable*] [, ITMAX=*value*])

LINDGEN - Returns a longword integer array with each element set to its subscript.
Result = LINDGEN(D_1, ..., D_8)

LINFIT - Fits by minimizing the Chi-square error statistic.
Result = LINFIT(*X*, *Y* [, CHISQ=*variable*] [, /DOUBLE] [, PROB=*variable*] [, SDEV=*value*] [, SIGMA=*variable*])

LINKIMAGE - Merges routines written in other languages with IDL at run-time.

LINKIMAGE, *Name*, *Image* [, *Type* [, *Entry*]] [, /DEVICE] [, /FUNCT] [, /KEYWORDS] [, MAX_ARGS=*value*] [, MIN_ARGS=*value*]
VMS Keywords: [, DEFAULT=*string*]

LIVE_CONTOUR - Displays contour plots using a GUI.

LIVE_CONTOUR [, Z_1,..., Z_{25}] [, /BUFFER] [, DIMENSIONS=*[width, height]*{normal units}] [, DRAW_DIMENSIONS=*[width, height]*{devive units}] [, ERROR=*variable*] [, /INDEXED_COLOR] [, INSTANCING={-1 | 0 | 1}] [, LOCATION=*[x, y]*{normal units}] [, /MANAGE_STYLE] [, NAME=*structure*] [, /NO_DRAW] [, /NO_SELECTION] [, /NO_STATUS] [, /NO_TOOLBAR] [, PARENT_BASE=*widget_id* | , TLB_LOCATION=*[Xoffset, Yoffset]*{device units}] [, PREFERENCE_FILE=*filename*{full path}] [, REFERENCE_OUT=*variable*] [, RENDERER={0 | 1}] [, REPLACE={*structure* | {0 | 1 | 2 | 3 | 4}}] [, STYLE=*name_or_reference*] [, TEMPLATE_FILE=*filename*] [, TITLE=*string*] [, WINDOW_IN=*string*] [, {X | Y}INDEPENDENT=*value*] [, {/X | /Y}LOG] [, {X | Y}RANGE=*[min, max]*{data units}] [, {X | Y}_TICKNAME=*array*]

LIVE_CONTROL - Sets the properties of a visualization in a LIVE tool from the IDL command line.

LIVE_CONTROL, [*Name*] [, /DIALOG] [, ERROR=*variable*] [, /NO_DRAW] [, PROPERTIES=*structure*] [, /SELECT] [, /UPDATE_DATA] [, WINDOW_IN=*string*]

LIVE_DESTROY - Destroys a window visualization or an element in a visualization.

LIVE_DESTROY, [$Name_1$,..., $Name_{25}$] [, /ENVIRONMENT] [, ERROR=*variable*] [, /NO_DRAW] [, /PURGE] [, WINDOW_IN=*string*]

LIVE_EXPORT - Exports visualization or window to a file.

LIVE_EXPORT [, /APPEND] [, COMPRESSION={0 | 1 | 2}{TIFF only}] [, /DIALOG] [, DIMENSIONS=*[width, height]*] [, ERROR=*variable*] [, FILENAME=*string*] [, ORDER={0 | 1}{JPEG or TIFF}] [, /PROGRESSIVE{JPEG only}] [, QUALITY={0 | 1 | 2}{for VRML} | {0 to 100}{for JPEG}] [, RESOLUTION=*value*] [, TYPE={'BMP' | 'GIF' | 'JPG' | 'PIC' | 'SRF' | 'TIF' | 'XWD' | 'VRML'}] [, UNITS={0 | 1 | 2}] [, VISUALIZATION_IN=*string*] [, WINDOW_IN=*string*]

LIVE_IMAGE - Displays visualizations using a GUI.

LIVE_IMAGE, *Image* [, RED=*byte_vector*] [, GREEN=*byte_vector*] [, BLUE=*byte_vector*] [, /BUFFER] [, DIMENSIONS=*[width, height]*{normal units}] [, DRAW_DIMENSIONS=*[width, height]*{devive units}] [, ERROR=*variable*] [, /INDEXED_COLOR] [, INSTANCING={-1 | 0 | 1}] [, LOCATION=*[x, y]*{normal units}] [, /MANAGE_STYLE] [, NAME=*structure*] [, /NO_DRAW] [, /NO_SELECTION] [, /NO_STATUS] [, /NO_TOOLBAR] [, PARENT_BASE=*widget_id* | , TLB_LOCATION=*[Xoffset, Yoffset]*{device units}] [, PREFERENCE_FILE=*filename*{full path}] [, REFERENCE_OUT=*variable*] [, RENDERER={0 | 1}] [, REPLACE={*structure* | {0 | 1 | 2 | 3 | 4}}] [, STYLE=*name_or_reference*] [, TEMPLATE_FILE=*filename*] [, TITLE=*string*] [, WINDOW_IN=*string*]

LIVE_INFO - Gets the properties of a LIVE tool.

LIVE_INFO, [*Name*] [, ERROR=*variable*] [, PROPERTIES=*variable*] [, WINDOW_IN=*string*]

LIVE_LINE - Provides an interface for line annotation.

LIVE_LINE [, ARROW_ANGLE=*value*{1.0 to 179.0}] [, /ARROW_END] [, ARROW_SIZE=*value*{0.0 to 0.3}] [, /ARROW_START] [, COLOR=*'color name'*] [, /DIALOG] [, DIMENSIONS=*[width, height]*] [, ERROR=*variable*] [, /HIDE] [, LINESTYLE={0 | 1 | 2 | 3 | 4 | 5}] [, LOCATION=*[x, y]*] [, NAME=*string*] [, /NO_DRAW] [, /NO_SELECTION] [, REFERENCE_OUT=*variable*] [, THICK=pixels{1 to 10}] [, VISUALIZATION_IN=*string*] [, WINDOW_IN=*string*]

LIVE_LOAD - Loads into memory the complete set of routines necessary to run all LIVE tools and Insight.

LIVE_LOAD

LIVE_OPLOT - Inserts data into pre-existing plots.

LIVE_OPLOT, *Yvector1* [,... , *Yvector25*] [, ERROR=*variable*] [, INDEPENDENT=*vector*] [, NAME=*structure*] [, /NEW_AXES] [, /NO_DRAW] [, /NO_SELECTION] [, REFERENCE_OUT=*variable*] [, REPLACE={*structure* | {0 | 1 | 2 | 3 | 4}}] [, SUBTYPE={'LinePlot' | 'ScatterPlot' | 'Histogram' | 'PolarPlot'}] [, VISUALIZATION_IN=*string*] [, WINDOW_IN=*string*] [, {X | Y}_TICKNAME=*array*] [, {X | Y}AXIS_IN=*string*]

LIVE_PLOT - Displays a plot using a GUI.

LIVE_PLOT, *Yvector1* [, *Yvector2*,..., *Yvector25*] [, /BUFFER] [, DIMENSIONS=*[width, height]*{normal units}] [, DRAW_DIMENSIONS=*[width, height]*{devive units}] [, ERROR=*variable*] [, /HISTOGRAM | , /LINE | , /POLAR | , /SCATTER] [, /INDEXED_COLOR] [, INSTANCING={-1 | 0 | 1}] [, LOCATION=*[x, y]*{normal units}] [, INDEPENDENT=*vector*] [, /MANAGE_STYLE] [, NAME=*structure*] [, /NO_DRAW] [, /NO_SELECTION] [, /NO_STATUS] [, /NO_TOOLBAR] [, PARENT_BASE=*widget_id* | ,

TLB_LOCATION=*[Xoffset, Yoffset]*{device units}]
[, PREFERENCE_FILE=*filename*{full path}]
[, REFERENCE_OUT=*variable*] [, RENDERER={0 | 1}]
[, REPLACE={*structure* | {0 | 1 | 2 | 3 | 4}}]
[, STYLE=*name_or_reference*]
[, TEMPLATE_FILE=*filename*] [, TITLE=*string*]
[, WINDOW_IN=*string*] [, {/X | /Y}LOG] [, {X | Y}RANGE=*[min, max]*{data units}] [, {X | Y}_TICKNAME=*array*]

LIVE_PRINT - Prints a given window to the printer.
LIVE_PRINT [, /DIALOG] [, ERROR=*variable*]
[, /SETUP] [, WINDOW_IN=*string*]

LIVE_RECT - Provides an interface for insertion of rectangles.
LIVE_RECT [, COLOR='*color name'*] [, /DIALOG]
[, DIMENSIONS=*[width, height]*] [, ERROR=*variable*]
[, /HIDE] [, LINESTYLE={0 | 1 | 2 | 3 | 4 | 5}]
[, LOCATION=*[x, y]*] [, NAME=*string*] [, /NO_DRAW]
[, /NO_SELECTION] [, REFERENCE_OUT=*variable*]
[, THICK=*pixels*{1 to 10}]
[, VISUALIZATION_IN=*string*] [, WINDOW_IN=*string*]

LIVE_STYLE - Controls style settings for a LIVE_ tool.
Style = LIVE_STYLE ({ 'contour' | 'image' | 'plot' | 'surface'} [, BASE_STYLE=*style_name*]
[, COLORBAR_PROPERTIES=*structure*]
[, ERROR=*variable*]
[, GRAPHIC_PROPERTIES=*structure*]
[, GROUP=*widget_id*]
[, LEGEND_PROPERTIES=*structure*] [, NAME=*string*]
[, /SAVE] [, TEMPLATE_FILE=*filename*]
[, VISUALIZATION_PROPERTIES=*structure*]
[, {X | Y | Z}AXIS_PROPERTIES=*structure*])

LIVE_SURFACE - Displays a surface using a GUI.
LIVE_SURFACE, *Data*, *Data2*,... [, /BUFFER]
[, DIMENSIONS=*[width, height]*{normal units}]
[, DRAW_DIMENSIONS=*[width, height]*{devive units}]
[, ERROR=*variable*] [, /INDEXED_COLOR]
[, INSTANCING={-1 | 0 | 1}] [, LOCATION=*[x, y]*{normal units}] [, /MANAGE_STYLE]
[, NAME=*structure*] [, /NO_DRAW]
[, /NO_SELECTION] [, /NO_STATUS]
[, /NO_TOOLBAR] [, PARENT_BASE=*widget_id* | , TLB_LOCATION=*[Xoffset, Yoffset]*{device units}]
[, PREFERENCE_FILE=*filename*{full path}]
[, REFERENCE_OUT=*variable*] [, RENDERER={0 | 1}]
[, REPLACE={*structure* | {0 | 1 | 2 | 3 | 4}}]
[, STYLE=*name_or_reference*]
[, TEMPLATE_FILE=*filename*] [, TITLE=*string*]
[, WINDOW_IN=*string*]
[, {X | Y}INDEPENDENT=*vector*] [, {/X | /Y}LOG]
[, {X | Y}RANGE=*[min, max]*{data units}]
[, {X | Y}_TICKNAME=*array*]

LIVE_TEXT - Provides an interface for text annotation.
LIVE_TEXT[, *Text*] [, ALIGNMENT=*value*{0.0 to 1.0}]
[, COLOR='*color name'*] [, /DIALOG]
[, /ENABLE_FORMATTING] [, ERROR=*variable*]
[, FONTNAME=*string*] [, FONTSIZE=*points*{9 to 72}]
[, /HIDE] [, LOCATION=*[x, y]*] [, NAME=*string*]
[, /NO_DRAW] [, /NO_SELECTION]
[, REFERENCE_OUT=*variable*]
[, TEXTANGLE=*value*{0.0 to 360.0}]
[, VERTICAL_ALIGNMENT=*value*{0.0 to 1.0}]
[, VISUALIZATION_IN=*string*] [, WINDOW_IN=*string*]

LJLCT - Loads standard color tables for LJ-250/252 printer.
LJLCT

LL_ARC_DISTANCE - Returns the longitude and latitude of a point a given arc distance and azimuth.
Result = LL_ARC_DISTANCE(*Lon_lat0*, *Arc_Dist*, *Az*
[, /DEGREES])

LMFIT - Does a non-linear least squares fit.
Result = LMFIT(*X*, *Y*, *A* [, ALPHA=*variable*]
[, CHISQ=*variable*] [, CONVERGENCE=*variable*]
[, COVAR=*variable*] [, /DOUBLE] [, FITA=*vector*]
[, FUNCTION_NAME=*string*] [, ITER=*variable*]
[, ITMAX=*value*] [, ITMIN=*value*] [, SIGMA=*variable*]
[, TOL=*value*] [, WEIGHTS=*vector*])

LMGR - Determines the type of license used by the current IDL session.
Result = LMGR([, /CLIENTSERVER | , /DEMO | , /EMBEDDED | , /RUNTIME | , /STUDENT | , /TRIAL]
[, EXPIRE_DATE=*variable*] [, /FORCE_DEMO]
[, INSTALL_NUM=*variable*] [, LMHOSTID=*variable*]
[, SITE_NOTICE=*variable*])

LNGAMMA - Returns logarithm of the gamma function of *X*.
Result = LNGAMMA(*X*)

LNP_TEST - Computes the Lomb Normalized Periodogram.
Result = LNP_TEST(*X*, *Y* [, HIFAC=*scale_factor*]
[, JMAX=*variable*] [, OFAC=*value*] [, WK1=*variable*]
[, WK2=*variable*])

LOADCT - Loads one of the predefined IDL color tables.
LOADCT [, *Table*] [, BOTTOM=*value*] [, FILE=*string*]
[, GET_NAMES=*variable*] [, NCOLORS=*value*]
[, /SILENT]

LOCALE_GET - Returns the current locale of the operating platform.
Result = LOCALE_GET( )

LON64ARR - Returns a 64-bit integer vector or array.
Result = LON64ARR(D_1, ..., D_8 [, /NOZERO])

LONARR - Returns a longword integer vector or array.
Result = LONARR(D_1, ..., D_8 [, /NOZERO])

LONG - Converts argument to longword integer type.
Result = LONG(*Expression*[, *Offset* [, Dim_1, ..., Dim_8]])

LONG64 - Converts argument to 64-bit integer type.
Result = LONG64(*Expression*[, *Offset* [, D_1, ..., D_8]])

LSODE - Advances a solution to a system of ordinary differential equations one time-step H.
Result = LSODE(Y, X, H, Derivs[, Status]
[, ATOL=*value*] [, RTOL=*value*])

LU_COMPLEX - Solves complex linear system using LU decomposition.
Result = LU_COMPLEX(*A, B* [, /DOUBLE]
[, /INVERSE] [, /SPARSE])

LUDC - Replaces array with the LU decomposition.
LUDC, *A, Index* [, /COLUMN] [, /DOUBLE]
[, INTERCHANGES=*variable*]

LUMPROVE - Uses LU decomposition to iteratively improve an approximate solution.
Result = LUMPROVE(*A, Alud, Index, B, X* [, /COLUMN]
[, /DOUBLE])

LUSOL - Solves a set of linear equations. Use with LUDC.
Result = LUSOL(*A, Index, B* [, /COLUMN] [, /DOUBLE])

M

M_CORRELATE - Computes multiple correlation coefficient.
Result = M_CORRELATE(*X, Y*)

MACHAR - Determines and returns machine-specific parameters affecting floating-point arithmetic.
Result = MACHAR([, /DOUBLE])

MAKE_ARRAY - Returns an array of the specified type, dimensions, and initialization.
Result = MAKE_ARRAY ([D_1, ..., D_8] [, /BYTE | , /COMPLEX | , /DCOMPLEX | , /DOUBLE | , /FLOAT | , /INT | , /L64 | , /LONG | , /OBJ, | , /PTR | , /STRING | , /UINT | , /UL64 | , /ULONG] [, DIMENSION=*vector*]
[, /INDEX] [, /NOZERO] [, SIZE=*vector*]
[, TYPE=*type_code*] [, VALUE=*value*])

MAP_CONTINENTS - Draws continental boundaries, filled continents, political boundaries, coastlines, and/or rivers, over an existing map projection established by MAP_SET.
MAP_CONTINENTS [, /COASTS] [, COLOR=*index*]
[, /COUNTRIES] [,FILL_CONTINENTS={1 | 2}[, ORIENTATION=*value*]] [, /HIRES]
[, MLINESTYLE={0 | 1 | 2 | 3 | 4 | 5}]
[, MLINETHICK=*value*] [, /RIVERS]
[, SPACING=*centimeters*] [, /USA]

MAP_GRID - Draws parallels and meridians over a map projection.
MAP_GRID [, /BOX_AXES] [, CHARSIZE=*value*]
[, CLIP_TEXT=0] [, COLOR=*index*]
[, GLINESTYLE={0 | 1 | 2 | 3 | 4 | 5}]
[, GLINETHICK=*value*]
[, LABEL=*n*{label_every_*n*th_gridline}]
[, LATALIGN=*value*{0.0 to 1.0}] [, LATDEL=*degrees*]
[, LATLAB=*longitude*] [, LATNAMES=*array*,
LATS=*vector*] [, LONALIGN=*value*{0.0 to 1.0}]
[, LONDEL=*degrees*] [, LONLAB=*latitude*]
[, LONNAMES=*array*, LONS=*vector*]
[, ORIENTATION=*clockwise_degrees_from_horiz*]

MAP_IMAGE - Returns an image warped to fit the current map projection. (Use when map data is larger than the display).
Result = MAP_IMAGE(*Image* [, *Startx*, *Starty* [, *Xsize*, *Ysize*]] [, LATMIN=*degrees*{-90 to 90}]
[, LATMAX=*degrees*{-90 to 90}]
[, LONMIN=*degrees*{-180 to 180}]
[, LONMAX=*degrees*{-180 to 180}] [, /BILINEAR]
[, COMPRESS=*value*] [, SCALE=*value*]
[, MAX_VALUE=*value*] [, MIN_VALUE=*value*]
[, MISSING=*value*])

MAP_PATCH - Returns an image warped to fit the current map projection. (Use when map data is smaller than the display).
Result = MAP_PATCH(*Image_Orig* [, *Lons*, *Lats*]
[, LAT0=*value*] [, LAT1=*value*] [, LON0=*value*]
[, LON1=*value*] [, MAX_VALUE=*value*]
[, MISSING=*value*] [, /TRIANGULATE]
[, XSIZE=*variable*] [, XSTART=*variable*]
[, YSIZE=*variable*] [, YSTART=*variable*])

MAP_PROJ_INFO - Returns information about current map and/or the available projections.
MAP_PROJ_INFO [, *iproj*] [, AZIMUTHAL=*variable*]
[, CIRCLE=*variable*] [, CYLINDRICAL=*variable*]
[, /CURRENT] [, LL_LIMITS=*variable*]
[, NAME=*variable*] [, PROJ_NAMES=*variable*]
[, UV_LIMITS=*variable*] [, UV_RANGE=*variable*]

MAP_SET - Establishes map projection type and limits.
MAP_SET [, *P0lat*, *P0lon*, *Rot*]
Keywords—Projection Types: [, /AITOFF | , /ALBERS | , /AZIMUTHAL | , /CONIC | , /CYLINDRICAL | , /GNOMIC | , /GOODESHOMOLOSINE | , /HAMMER | , /LAMBERT | , /MERCATOR | , /MILLER | , /MOLLEWIDE | , /ORTHOGRAPHIC | , /ROBINSON | , /SATELLITE | , /SINUSOIDAL | , /STEREOGRAPHIC | , /TRANSVERSE_MERCATOR]
Keywords—Map Characteristics: [, /ADVANCE]
[, CHARSIZE=*value*] [, /CLIP] [, COLOR=*index*]
[, /CONTINENTS [, CON_COLOR=*index*] [, /HIRES]]
[, E_CONTINENTS=*structure*] [, E_GRID=*structure*]
[, E_HORIZON=*structure*] [, GLINESTYLE={0 | 1 | 2 | 3 | 4 | 5}] [, GLINETHICK=*value*] [, /GRID] [, /HORIZON]
[, LABEL=*n*{label every *n*th gridline}]
[, LATALIGN=*value*{0.0 to 1.0}] [, LATDEL=*degrees*]
[, LATLAB=*longitude*] [, LONDEL=*degrees*]
[, LONLAB=*latitude*] [, MLINESTYLE={0 | 1 | 2 | 3 | 4 | 5}] [, MLINETHICK=*value*] [, /NOBORDER]
[, /NOERASE] [, TITLE=*string*] [, /USA]
[, XMARGIN=*value*] [, YMARGIN=*value*]
Keywords—Projection Parameters:
[, CENTRAL_AZIMUTH=*degrees_east_of_north*]
[, ELLIPSOID=*array*] [, /ISOTROPIC] [, LIMIT=*vector*]

[, SAT_P=*vector*] [, SCALE=*value*]
[, STANDARD_PARALLELS=*array*]
Graphics Keywords: [, POSITION=*[X_0, Y_0, X_1, Y_1]*]
[, /T3D] [, ZVALUE=*value*{0 to 1}]

MAX - Returns the value of the largest element of *Array.*
Result = MAX(*Array* [, *Max_Subscript*] [, MIN=*variable*]
[, /NAN])

MD_TEST - Performs the Median Delta test.
Result = MD_TEST(*X* [, ABOVE=*variable*]
[, BELOW=*variable*] [, MDC=*variable*])

MEAN - Computes the mean of a numeric vector.
Result = MEAN(*X* [, /DOUBLE] [, /NAN])

MEANABSDEV - Computes the mean absolute deviation of a vector.
Result = MEANABSDEV(*X* [, /DOUBLE] [, /MEDIAN]
[, /NAN])

MEDIAN - Returns the median value of *Array* or applies a median filter.
Result = MEDIAN(*Array* [, *Width*] [, /EVEN])

MESH_CLIP - Clips a polygonal mesh to an arbitrary plane in space and returns a polygonal mesh of the remaining portion.
Result = MESH_CLIP (*Plane*, *Vertsin*, *Connin*, *Vertsout*,
Connout [, AUXDATA_IN=*array*,
AUXDATA_OUT=*variable*] [, CUT_VERTS=*variable*])

MESH_DECIMATE - Accepts additional array of auxiliary data values that can be used together with a weighting function to allow external data to be considered when a particular triangle is removed.
Result = MESH_DECIMATE (*Verts*, *Conn*, *Connout*
[, /VERTICES] [, PERCENT_VERTICES=*percent* |
, PERCENT_POLYGONS=*percent*])

MESH_ISSOLID - Computes various mesh properties and enables IDL to determine if a mesh encloses space (is a solid).
Result = MESH_ISSOLID (*Conn*)

MESH_MERGE - Merges two polygonal meshes..
Result = MESH_MERGE (*Verts*, *Conn*, *Verts1*, *Conn1*
[, /COMBINE_VERTICES] [, TOLERANCE=*value*])

MESH_NUMTRIANGLES - Computes the number of triangles in a polygonal mesh..
Result = MESH_NUMTRIANGLES (*Conn*)

MESH_OBJ - Generates a polygon mesh for various simple objects.
MESH_OBJ, *Type*, *Vertex_List*, *Polygon_List*, *Array1*
[, *Array2*] [, /DEGREES] [, P1=*value*] [, P2=*value*]
[, P3=*value*] [, P4=*value*] [, P5=*value*]

MESH_SMOOTH - Performs spatial smoothing on a polygon mesh.
Result = MESH_SMOOTH (*Verts*, *Conn*
[, ITERATIONS=*value*] [, FIXED_VERTICES=*array*]
[, /FIXED_EDGE_VERTICES] [, LAMBDA=*value*])

MESH_SURFACEAREA - Computes various mesh properties to determine the mesh surface area, including integration of other properties interpolated on the surface of the mesh.
Result = MESH_SURFACEAREA (*Verts*, *Conn*
[, AUXDATA=*array*] [, MOMENT=*variable*])

MESH_VALIDATE - Checks for NaN values in vertices, removes unused vertices, and combines close vertices.
Result = MESH_VALIDATE (*Verts*, *Conn*
[, /REMOVE_NAN] [, /PACK_VERTICES]
[, /COMBINE_VERTICES] [, TOLERANCE=*value*])

MESH_VOLUME - Computes the volume that the mesh encloses.
Result = MESH_VOLUME (*Verts*, *Conn* [, /SIGNED])

MESSAGE - Issues error and informational messages.
MESSAGE, [*Text*] [, /CONTINUE]
[, /INFORMATIONAL] [, /IOERROR] [, /NONAME]
[, /NOPREFIX] [, /NOPRINT] [, /RESET]

MIN - Returns the value of the smallest element of an array.
Result = MIN(*Array* [, *Min_Subscript*] [, MAX=*variable*]
[, /NAN])

MIN_CURVE_SURF - Interpolates points with a minimum curvature surface or a thin-plate-spline surface. Useful with CONTOUR.
Result = MIN_CURVE_SURF(*Z* [, *X*, *Y*] [, /DOUBLE]
[, /TPS] [, /REGULAR] [, XGRID=*[xstart, xspacing]* | ,
XVALUES=*array*] [, YGRID=*[ystart, yspacing]* | ,
YVALUES=*array*] [, GS=*[xspace,yspace]*]
[, BOUNDS=*[xmin, ymin, xmax, ymax]*] [, NX=*value*]
[, NY=*value*] [, XOUT=*vector*] [, YOUT=*vector*]
[, XPOUT=*array*, YPOUT=*array*])

MK_HTML_HELP - Converts text documentation headers to HTML files.
MK_HTML_HELP, *Sources*, *Filename* [, /STRICT]
[, TITLE=*string*] [, /VERBOSE]

MODIFYCT - Saves modified color tables in the IDL color table file.
MODIFYCT, *Itab*, *Name*, *R*, *G*, *B* [, FILE=*filename*]

MOMENT - Computes mean, variance, skewness, and kurtosis.
Result = MOMENT(*X* [, /DOUBLE] [, MDEV=*variable*]
[, /NAN] [, SDEV=*variable*])

MORPH_CLOSE - Applies closing operator to binary or grayscale image.
Result = MORPH_CLOSE (*Image*, *Structure* [, /GRAY]
[, PRESERVE_TYPE=*bytearray* | /UINT | /ULONG]
[, VALUES=*array*])

MORPH_DISTANCE - Estimates *N*-dimensional distance maps, which contain for each foreground pixel the distance to the nearest background pixel, using a given norm.
Result = MORPH_DISTANCE (*Data* [, /BACKGROUND]
[, NEIGHBOR_SAMPLING={1 | 2 | 3 }] [, /NO_COPY])

MORPH_GRADIENT - Applies the morphological gradient operator to a grayscale image.
Result = MORPH_GRADIENT (*Image*, *Structure*
[, PRESERVE_TYPE=*bytearray* | /UINT | /ULONG]
[, VALUES=*array*])

MORPH_HITORMISS - Applies the hit-or-miss operator to a binary image.
Result = MORPH_HITORMISS (*Image*, *HitStructure*, *MissStructure*)

MORPH_OPEN - Applies the opening operator to a binary or gray-scale image.
Result = MORPH_OPEN (*Image*, *Structure* [, /GRAY]
[, PRESERVE_TYPE=*bytearray* | /UINT | /ULONG]
[, VALUES=*array*])

MORPH_THIN - Performs a thinning operation on binary images.
Result = MORPH_THIN (*Image*, *HitStructure*, *MissStructure*)

MORPH_TOPHAT - Applies top-hat operator to a grayscale image.
Result = MORPH_TOPHAT (*Image*, *Structure*
[, PRESERVE_TYPE=*bytearray* | /UINT | /ULONG]
[, VALUES=*array*])

MPEG_CLOSE - Closes an MPEG sequence.
MPEG_CLOSE, *mpegID*

MPEG_OPEN - Opens an MPEG sequence.
mpegID = MPEG_OPEN(*Dimensions*
[, FILENAME=*string*])

MPEG_PUT - Inserts an image array into an MPEG sequence
MPEG_PUT, *mpegID* [, /COLOR]
[, FRAME=*frame_number*] [, IMAGE=*array* | ,
WINDOW=*index*] [, /ORDER]

MPEG_SAVE - Encodes and saves an open MPEG sequence.
MPEG_SAVE, *mpegID* [, FILENAME=*string*]

MSG_CAT_CLOSE - Closes a catalog file from the stored cache.
MSG_CAT_CLOSE, *object*

MSG_CAT_COMPILE - Creates an IDL language catalog file.
MSG_CAT_COMPILE, *input*[, *output*]
[, LOCALE_ALIAS=*string*] [, /MBCS]

MSG_CAT_OPEN - Returns a catalog object for the given parameters if found.
Result = MSG_CAT_OPEN(*application*
[, DEFAULT_FILENAME=*filename*]
[, FILENAME=*string*] [, FOUND=*variable*]
[, LOCALE=*string*] [, PATH=*string*]
[, SUB_QUERY=*value*])

MULTI - Replicates current color table to enhance contrast.
MULTI, *N*

N

N_ELEMENTS - Returns the number of elements contained in an expression or variable.
Result = N_ELEMENTS(*Expression*)

N_PARAMS - Returns the number of non-keyword parameters used in calling an IDL procedure or function.
Result = N_PARAMS( )

N_TAGS - Returns the number of tags in a structure.
Result = N_TAGS(*Expression* [, /LENGTH])

NCDF_* Routines - See "NetCDF Routines" on page 49.

NEWTON - Solves nonlinear equations using Newton's method.
Result = NEWTON(*X*, *Vecfunc* [, CHECK=*variable*]
[, /DOUBLE] [, ITMAX=*value*] [, STEPMAX=*value*]
[, TOLF=*value*] [, TOLMIN=*value*] [, TOLX=*value*])

NORM - Computes Euclidean norm of vector or Infinity norm of array.
Result = NORM(*A* [, /DOUBLE])

O

OBJ_CLASS - Determines the class name of an object.
Result = OBJ_CLASS([*Arg*] [, COUNT=*variable*]
[, /SUPERCLASS{must specify Arg}])

OBJ_DESTROY - Destroys an object reference.
OBJ_DESTROY, *ObjRef* [, Arg_1, …, Arg_n]

OBJ_ISA - Determines inheritance relationship of an object.
Result = OBJ_ISA(*ObjectInstance*, *ClassName*)

OBJ_NEW - Creates an object reference.
Result = OBJ_NEW([*ObjectClassName* [, Arg_1…..Arg_n]])

OBJ_VALID - Verifies validity of object references.
Result = OBJ_VALID([*Arg*] [, CAST=*integer*]
[, COUNT=*variable*])

OBJARR - Creates an array of object references.
Result = OBJARR(D_1, …, D_8 [, /NOZERO])

ON_ERROR - Designates the error recovery method.
ON_ERROR, *N*

ON_IOERROR - Declares I/O error exception handler.
ON_IOERROR, *Label*
. . .
Label: *Statement to perform upon I/O error*

ONLINE_HELP - Invokes hypertext help viewer from programs.
ONLINE_HELP [, *Topic*] [, BOOK='*filename*']
[, /CONTEXT] [, /FULL_PATH] [, /QUIT]

OPEN - Opens files for reading, updating, or writing.
OPENR, *Unit*, *File* [, *Record_Length*]
OPENW, *Unit*, *File* [, *Record_Length*]
OPENU, *Unit*, *File* [, *Record_Length*]

Keywords (all platforms): [, /APPEND | , /COMPRESS] [, BUFSIZE={0 | 1 | *value*>512}] [, /DELETE] [, ERROR=*variable*] [, /F77_UNFORMATTED] [, /GET_LUN] [, /MORE] [, /SWAP_ENDIAN] [, SWAP_IF_BIG_ENDIAN] [, /SWAP_IF_LITTLE_ENDIAN] [, /VAX_FLOAT] [, WIDTH=*value*] [, /XDR]
Macintosh Keywords: [, MACCREATOR=*string*] [, MACTYPE= *string*]
Windows Keywords: [, /BINARY] [, /NOAUTOMODE]
UNIX Keywords: [, /NOSTDIO]
VMS Keywords: [, /BLOCK | , /SHARED | , /UDF_BLOCK] [, DEFAULT='*.extension*'] [, /EXTENDSIZE] [, /FIXED] [, /FORTRAN] [, INITIALSIZE=*blocks*] [, /KEYED] [, /LIST] [, /NONE] [, /PRINT] [, /SEGMENTED] [, /STREAM] [, /SUBMIT] [, /SUPERSEDE] [, /TRUNCATE_ON_CLOSE] [, /VARIABLE]

OPLOT - Plots vector data over a previously-drawn plot.
OPLOT, [*X*,] *Y* [, MAX_VALUE=*value*] [, MIN_VALUE=*value*] [, NSUM=*value*] [, /POLAR] [, THICK=*value*]
Graphics Keywords: [, CLIP=*[X_0, Y_0, X_1, Y_1]*] [, COLOR=*value*] [, LINESTYLE={0 | 1 | 2 | 3 | 4 | 5}] [, /NOCLIP] [, PSYM=*integer*{0 to 10}] [, SYMSIZE=*value*] [, /T3D] [, ZVALUE=*value*{0 to 1}]

OPLOTERR - Draws error bars over a previously drawn plot.
OPLOTERR, [*X* ,] *Y* , *Err* [, *Psym*]

P

P_CORRELATE - Computes partial correlation coefficient.
Result = P_CORRELATE(*X, Y, C*)

PARTICLE_TRACE - Traces the path of a massless particle through a vector field.
PARTICLE_TRACE, *Data, Seeds, Verts, Conn* [, *Normals*] [, MAX_ITERATIONS=*value*] [, ANISOTROPY=*array*] [, INTEGRATION={0 | 1}] [, SEED_NORMAL=*vector*] [, TOLERANCE=*value*] [, MAX_STEPSIZE=*value*] [, /UNIFORM]

PCOMP - Computes principal components/derived variables.
Result = PCOMP(*A* [, COEFFICIENTS=*variable*] [, /COVARIANCE] [, /DOUBLE] [, EIGENVALUES=*variable*] [, NVARIABLES=*value*] [, /STANDARDIZE] [, VARIANCES=*variable*])

PLOT - Plots vector arguments as X versus Y graphs.
PLOT, [*X*,] *Y* [, MAX_VALUE=*value*] [, MIN_VALUE=*value*] [, NSUM=*value*] [, /POLAR] [, THICK=*value*] [, /XLOG] [, /YLOG] [, /YNOZERO]
Graphics Keywords: [, BACKGROUND=*color_index*] [, CHARSIZE=*value*] [, CHARTHICK=*integer*] [, CLIP=*[X_0, Y_0, X_1, Y_1]*] [, COLOR=*value*][, /DATA | , /DEVICE | , /NORMAL] [, FONT=*integer*] [, LINESTYLE={0 | 1 | 2 | 3 | 4 | 5}] [, /NOCLIP] [, /NODATA] [, /NOERASE] [, POSITION=*[X_0, Y_0, X_1, Y_1]*] [, PSYM=*integer*{0 to 10}] [, SUBTITLE=*string*] [, SYMSIZE=*value*] [, /T3D] [, THICK=*value*] [, TICKLEN=*value*] [, TITLE=*string*] [, {X | Y | Z}CHARSIZE=*value*] [, {X | Y | Z}GRIDSTYLE=*integer*{0 to 5}] [, {X | Y | Z}MARGIN=*[left, right]*] [, {X | Y | Z}MINOR=*integer*] [, {X | Y | Z}RANGE=*[min, max]*] [, {X | Y | Z}STYLE=*value*] [, {X | Y | Z}THICK=*value*] [, {X | Y | Z}TICKFORMAT=*string*] [, {X | Y | Z}TICKLEN=*value*] [, {X | Y | Z}TICKNAME=*string_array*] [, {X | Y | Z}TICKS=*integer*] [, {X | Y | Z}TICKV=*array*] [, {X | Y | Z}TICK_GET=*variable*] [, {X | Y | Z}TITLE=*string*] [, ZVALUE=*value*{0 to 1}]

PLOT_3DBOX - Plots function of two variables inside 3D box.
PLOT_3DBOX, *X, Y, Z* [, GRIDSTYLE={0 | 1 | 2 | 3 | 4 | 5}] [, PSYM=*integer*{1 to 10}] [, /SOLID_WALLS] [, /XY_PLANE] [, XYSTYLE={0 | 1 | 2 | 3 | 4 | 5}] [, /XZ_PLANE] [, XZSTYLE={0 | 1 | 2 | 3 | 4 | 5}] [, /YZ_PLANE] [, YZSTYLE={0 | 1 | 2 | 3 | 4 | 5}] [, AX=*degrees*] [, AZ=*degrees*] [, ZAXIS={1 | 2 | 3 | 4}]
Graphics Keywords: Accepts all graphics keywords accepted by PLOT except for: FONT, PSYM, SYMSIZE, {XYZ}TICK_GET, and ZVALUE.

PLOT_FIELD - Plots a 2D field using arrows.
PLOT_FIELD, *U, V* [, ASPECT=*ratio*] [, LENGTH=*value*] [, N=*num_arrows*] [, TITLE=*string*]

PLOTERR - Plots individual data points with error bars.
PLOTERR, [*X* ,] *Y* , *Err* [, TYPE={1 | 2 | 3 | 4}] [, PSYM=*integer*{1 to 10}]

PLOTS - Plots vectors and points.
PLOTS, *X* [, *Y* [, *Z*]] [, /CONTINUE]
Graphics Keywords: [, CLIP=*[X_0, Y_0, X_1, Y_1]*] [, COLOR=*value*] [, /DATA | , /DEVICE | , /NORMAL] [, LINESTYLE={0 | 1 | 2 | 3 | 4 | 5}] [, /NOCLIP] [, PSYM=*integer*{0 to 10}] [, SYMSIZE=*value*] [, /T3D] [, THICK=*value*] [, Z=*value*]

PNT_LINE - Returns the perpendicular distance between a point and a line.
Result = PNT_LINE(*P0, L0, L1* [, *Pl*] [, /INTERVAL])

POINT_LUN - Sets or gets current position of the file pointer.
POINT_LUN, *Unit*, *Position*

POLAR_CONTOUR - Draws a contour plot from data in polar coordinates.
POLAR_CONTOUR, *Z, Theta, R* [, C_ANNOTATION=*vector_of_strings*]

[, C_CHARSIZE=*value*] [, C_CHARTHICK=*integer*]
[, C_COLORS=*vector*] [, C_LINESTYLE=*vector*]
[, /FILL | , CELL_FILL [, C_ORIENTATION=*degrees*]
[, C_SPACING=*value*]] [, C_THICK=*vector*]
[, /CLOSED] [, /IRREGULAR] [, LEVELS=*vector* |
NLEVELS=*integer*{1 to 29}] [, MAX_VALUE=*value*]
[, MIN_VALUE=*value*] [, /OVERPLOT]
[, /PATH_DATA_COORDS |
,TRIANGULATION=*variable*] [, /XLOG] [, /YLOG]
[, /ZAXIS] [, SHOW_TRIANGULATION=*color_index*]

POLAR_SURFACE - Interpolates a surface from polar coordinates to rectangular coordinates.
Result = POLAR_SURFACE(*Z, R, Theta* [, /GRID]
[, SPACING=*[xspacing, yspacing]*] [, BOUNDS=*[x_0, y_0, x_1, y_1]*] [, /QUINTIC] [, MISSING=*value*])

POLY - Evaluates polynomial function of a variable.
Result = POLY(*X, C*)

POLY_2D - Performs polynomial warping of images.
Result = POLY_2D(*Array, P, Q* [, *Interp* [, *Dim_x, Dim_y*]]
[, CUBIC={-1 to 0}] [, MISSING=*value*])

POLY_AREA - Returns the area of a polygon given the coordinates of its vertices.
Result = POLY_AREA(*X, Y*)

POLY_FIT - Performs a least-square polynomial fit.
Result = POLY_FIT(*X, Y, NDegree* [, *Yfit, Yband, Sigma, Corrm*] [, /DOUBLE])

POLYFILL - Fills the interior of a polygon.
POLYFILL, *X* [, *Y* [, *Z*]] [, FILL_PATTERN=*index*]
[, IMAGE_COORD=*array*] [, /IMAGE_INTERP]
[, /LINE_FILL] [, PATTERN=*array*]
[, SPACING=*centimeters*] [, TRANSPARENT=*value*]
Graphics Keywords: [, CLIP=*[X_0, Y_0, X_1, Y_1]*]
[, COLOR=*value*] [, /DATA | , /DEVICE | , /NORMAL]
[, LINESTYLE={0 | 1 | 2 | 3 | 4 | 5}] [, /NOCLIP]
[, ORIENTATION=*ccw_degrees_from_horiz*] [, /T3D]
[, THICK=*value*] [, Z=*value*]

POLYFILLV - Returns subscripts of pixels inside a polygon.
Result = POLYFILLV(*X, Y,* S_x, S_y [, *Run_Length*])

POLYFITW - Performs a weighted least-square polynomial fit.
Result = POLYFITW(*X, Y, Weights, NDegree* [, *Yfit, Yband, Sigma, Corrm*])

POLYSHADE - Creates a shaded surface representation from a set of polygons.
Result = POLYSHADE(*Vertices, Polygons*)
or
Result = POLYSHADE(*X, Y, Z, Polygons*)
Keywords: [, DATA] [, /NORMAL]
[, POLY_SHADES=*array*] [, SHADES=*array*] [, /T3D]
[, TOP=*value*] [, XSIZE=*columns*] [, YSIZE=*rows*]

POLYWARP - Performs polynomial spatial warping.
POLYWARP, *Xi, Yi, Xo, Yo, Degree, Kx, Ky*

POPD - Removes the top directory on the working directory stack maintained by PUSHD/POPD.
POPD

POWELL - Minimizes a function using the Powell method.
POWELL, *P, Xi, Ftol, Fmin, Func* [, /DOUBLE]
[, ITER=*variable*] [, ITMAX=*value*]

PRIMES - Computes the first *K* prime numbers.
Result = PRIMES(*K*)

PRINT/PRINTF - Writes formatted output to screen or file.
PRINT [, *$Expr_1$, ..., $Expr_n$*]
PRINTF [, *Unit, $Expr_1$, ..., $Expr_n$*]
Keywords: [, AM_PM=*[string, string]*]
[, DAYS_OF_WEEK=*string_array*{7 names}]
[, FORMAT=*value*] [, MONTHS=*string_array*{12 names}] [, /STDIO_NON_FINITE]
VMS Keywords: [, /REWRITE]

PRINTD - Prints contents of the directory stack maintained by PUSHD/POPD.
PRINTD

PRO - Defines a procedure.
PRO *Procedure_Name, $argument_1$, ..., $argument_n$*
. . .
END

PROFILE - Extracts a profile from an image.
Result = PROFILE(*Image* [, *XX, YY*] [, /NOMARK]
[, XSTART=*value*] [, YSTART=*value*])

PROFILER - Accesses the IDL Code Profiler used to analyze performance of applications.
PROFILER [, *Module*] [, /CLEAR] [, DATA=*variable*]
[, OUTPUT=*variable*] [, /REPORT] [, /RESET]
[, /SYSTEM]

PROFILES - Interactively examines image profiles.
PROFILES, *Image* [, /ORDER] [, SX=*value*] [, SY=*value*]
[, WSIZE=*value*]

PROJECT_VOL - Returns a translucent rendering of a volume projected onto a plane.
Return = PROJECT_VOL(*Vol, X_Sample, Y_Sample, Z_Sample* [, DEPTH_Q=*value*] [, OPAQUE=*3D_array*]
[, TRANS=*array*])

PS_SHOW_FONTS - Displays all the PostScript fonts that IDL knows about.
PS_SHOW_FONTS [, /NOLATIN]

PSAFM - Converts Adobe Font Metrics file to IDL format.
PSAFM, *Input_Filename, Output_Filename*

PSEUDO - Creates pseudo-color table based on Lightness, Hue, and Brightness system.
PSEUDO, *Litlo, Lithi, Satlo, Sathi, Hue, Loops* [, *Colr*]

PTR_FREE - Destroys a pointer.
PTR_FREE, P_1, ... …, P_n

PTR_NEW - Creates a pointer.
Result = PTR_NEW([*InitExpr*] [, /ALLOCATE_HEAP] [, /NO_COPY])

PTR_VALID - Verifies the validity of pointers.
Result = PTR_VALID([*Arg*] [, /CAST] [, COUNT=*variable*])

PTRARR - Creates an array of pointers.
Result = PTRARR(D_1, ... …, D_8 [, /ALLOCATE_HEAP | , /NOZERO])

PUSHD - Pushes a directory to top of directory stack maintained by PUSHD/POPD.
PUSHD, *Dir*

Q

QROMB - Evaluates integral over a closed interval.
Result = QROMB(*Func*, *A*, *B* [, /DOUBLE] [, EPS=*value*] [, JMAX=*value*] [, K=*value*])

QROMO - Evaluates integral over an open interval.
Result = QROMO(*Func*, *A* [, *B*] [, /DOUBLE] [, EPS=*value*] [, JMAX=*value*] [, K=*value*] [, /MIDEXP | , /MIDINF | , /MIDPNT | , /MIDSQL | , /MIDSQU])

QSIMP - Evaluates integral using Simpson's rule.
Result = QSIMP(*Func*, *A*, *B* [, /DOUBLE] [, EPS=*value*] [, JMAX=*value*])

QUERY_BMP - Obtains information about a BMP image file.
Result = QUERY_BMP (*Filename* [, *Info*])

QUERY_DICOM - Tests a file for compatibility with READ_DICOM.
Result = QUERY_DICOM(*Filename*[, *Info*] [, IMAGE_INDEX=*index*])

QUERY_GIF - Obtains information about a GIF image file.
Result = QUERY_GIF (*Filename* [, *Info*])

QUERY_IMAGE - Reads the header of a file and determines if it is recognized as an image file.
Result = QUERY_IMAGE (*Filename*[, *Info*] [, CHANNELS=*variable*] [, DIMENSIONS=*variable*] [, HAS_PALETTE=*variable*] [, IMAGE_INDEX=*index*] [, NUM_IMAGES=*variable*] [, PIXEL_TYPE=*variable*] [, SUPPORTED_READ=*variable*] [, SUPPORTED_WRITE=*variable*] [, TYPE=*variable*])

QUERY_JPEG - Obtains information about a JPEG image file.
Result = QUERY_JPEG (*Filename* [, *Info*])

QUERY_PICT - Obtains information about a PICT image file.
Result = QUERY_PICT (*Filename*, *Info*)

QUERY_PNG - Obtains information about a PNG image file.
Result = QUERY_PNG (*Filename* [, *Info*])

QUERY_PPM - Obtains information about a PPM image file.
Result = QUERY_PPM (*Filename* [, *Info*] [, MAXVAL=*variable*])

QUERY_SRF - Obtains information about an SRF image file.
Result = QUERY_SRF (*Filename* [, *Info*])

QUERY_TIFF - Obtains information about a TIFF image file.
Result = QUERY_TIFF (*Filename* [, *Info*] [, IMAGE_INDEX=*index*])

QUERY_WAV - Checks that the file is actually a .WAV file and that the READ_WAV function can read the data in the file.
Result = QUERY_WAV (*Filename*[, *Info*])

R

R_CORRELATE - Computes rank correlation.
Result = R_CORRELATE(*X*, *Y* [, D=*variable*] [, /KENDALL] [, PROBD=*variable*] [, ZD=*variable*])

R_TEST - Runs test for randomness.
Result = R_TEST(*X* [, N0=*variable*] [, N1=*variable*] [, R=*variable*])

RANDOMN - Returns normally-distributed random numbers.
Result = RANDOMN(*Seed* [, D_1, ..., D_8] [, BINOMIAL=*[trials, probability]*] [, GAMMA=*integer*{>0}] [, /NORMAL] [, POISSON=*value*] [, /UNIFORM])

RANDOMU - Returns uniformly-distributed random numbers.
Result = RANDOMU(*Seed* [, D_1, ..., D_8] [, BINOMIAL=*[trials, probability]*] [, GAMMA=*integer*{>0}] [, /NORMAL] [, POISSON=*value*] [, /UNIFORM])

RANKS - Computes magnitude-based ranks.
Result = RANKS(*X*)

RDPIX - Interactively displays image pixel values.
RDPIX, *Image* [, *X0*, *Y0*]

READ/READF - Reads formatted input from keyboard or file.
READ, [*Prompt*,] Var_1, ..., Var_n
READF, [*Prompt*,] *Unit*, Var_1, ..., Var_n
Keywords: [, AM_PM=*[string, string]*] [, DAYS_OF_WEEK=*string_array*{7 names}] [, FORMAT=*value*] [, MONTHS=*string_array*{12 names}] [, PROMPT=*string*]
VMS Keywords: [, KEY_ID=*value*] [, KEY_MATCH=*relation*] [, KEY_VALUE=*value*]

READ_ASCII - Reads data from an ASCII file.
Result = READ_ASCII([*Filename*] [, COMMENT_SYMBOL=*string*] [, COUNT=*variable*]

[, DATA_START=*lines_to_skip*] [, DELIMITER=*string*]
[, HEADER=*variable*] [, MISSING_VALUE=*value*]
[, NUM_RECORDS=*value*] [, RECORD_START=*index*]
[, TEMPLATE=*value*] [, /VERBOSE])

READ_BINARY - Reads the contents of a binary file using a passed template or basic command line keywords.
Result = READ_BINARY ([*Filename* | *FileUnit*]
[, TEMPLATE=*template*] | [[, DATA_START=*value*]
[, DATA_TYPE=*typecodes*] [, DATA_DIMS=*array*]
[, ENDIAN=*string*]])

READ_BMP - Reads Microsoft Windows bitmap file (.BMP).
Result = READ_BMP(*Filename*, [, *R*, *G*, *B* [, *Ihdr*]
[, /RGB])

READ_DICOM - Reads an image from a DICOM file.
Result = READ_DICOM (*Filename* [, *Red*, *Green*, *Blue*]
[, IMAGE_INDEX=*index*])

READ_GIF - Reads GIF file.
READ_GIF, *Filename*, *Image* [, *R*, *G*, *B*] [, /CLOSE]
[, /MULTIPLE]

READ_IMAGE - Reads the image contents of a file and returns the image in an IDL variable.
Result = READ_IMAGE (*Filename* [, *Red*, *Green*, *Blue*]
[, ALLOWED_FORMATS=*string*] [, FORMAT=*string*]
[, IMAGE_INDEX=*index*])

READ_INTERFILE - Reads Interfile (v3.3) file.
READ_INTERFILE, *File*, *Data*

READ_JPEG - Reads JPEG file.
READ_JPEG [, *Filename* | , UNIT=*lun*] , *Image*
[, *Colortable*] [, BUFFER=*variable*] [, COLORS=*value*{8 to 256}] [, DITHER={0 | 1 | 2}] [, /GRAYSCALE]
[, /ORDER] [, TRUE={1 | 2 | 3}]
[, /TWO_PASS_QUANTIZE]

READ_PICT - Reads Macintosh PICT (version 2) bitmap file.
READ_PICT, *Filename*, *Image* [, *R*, *G*, *B*]

READ_PNG - Reads Portable Network Graphics (PNG) file.
Result = READ_PNG (*Filename*[, *R*, *G*, *B*] [, /VERBOSE]
[, /TRANSPARENT])

READ_PPM - Reads PGM (gray scale) or PPM (portable pixmap for color) file.
READ_PPM, *Filename*, *Image* [, MAXVAL=*variable*]

READ_SPR - Reads a row-indexed sparse matrix from a file.
Result = READ_SPR(*Filename*)

READ_SRF - Reads Sun Raster Format file.
READ_SRF, *Filename*, *Image* [, *R*, *G*, *B*]

READ_SYLK - Reads Symbolic Link format spreadsheet file.
Result = READ_SYLK(*File* [, /ARRAY] [, /COLMAJOR]
[, NCOLS=*columns*] [, NROWS=*rows*]
[, STARTCOL=*column*] [, STARTROW=*row*]
[, /USEDOUBLES] [, /USELONGS])

READ_TIFF - Reads TIFF format file.
Result = READ_TIFF(*Filename* [, *R*, *G*, *B*]
[, GEOTIFF=*variable*] [, IMAGE_INDEX=*value*]
[, ORDER=*variable*] [, PLANARCONFIG=*variable*]
[, SUB_RECT=*[x, y, width, height]*] [, /UNSIGNED]
[, /VERBOSE])

READ_WAV - Reads the audio stream from the named .WAV file.
Result = READ_WAV (*Filename* [, *Rate*])

READ_WAVE - Reads Wavefront Advanced Visualizer file.
READ_WAVE, *File*, *Variables*, *Names*, *Dimensions*
[, MESHNAMES=*variable*]

READ_X11_BITMAP - Reads X11 bitmap file.
READ_X11_BITMAP, *File*, *Bitmap* [, *X*, *Y*]
[, /EXPAND_TO_BYTES]

READ_XWD - Reads X Windows Dump file.
Result = READ_XWD(*Filename*[, *R*, *G*, *B*])

READS - Reads formatted input from a string variable.
READS, *Input*, Var_1, ..., Var_n [, AM_PM=*[string, string]*]
[, DAYS_OF_WEEK=*string_array*{7 names}]
[, FORMAT=*value*] [, MONTHS=*string_array*{12 names}]

READU - Reads unformatted binary data from a file.
READU, *Unit*, Var_1, ..., Var_n
UNIX Keywords: [, TRANSFER_COUNT=*variable*]
VMS Keywords: [, KEY_ID=*index*]
[, KEY_MATCH=*relation*] [, KEY_VALUE=*value*]

REBIN - Resizes a vector or array by integer multiples.
Result = REBIN(*Array*, D_1 [, ..., D_8] [, /SAMPLE])

RECALL_COMMANDS - Returns entries in IDL's command recall buffer.
Result = RECALL_COMMANDS( )

RECON3 - Reconstructs a 3D representation of an object from 2D images.
Result = RECON3(*Images*, *Obj_Rot*, *Obj_Pos*, *Focal*, *Dist*,*Vol_Pos*, *Img_Ref*, *Img_Mag*, *Vol_Size*
[, MISSING=*value*] [, MODE=*value*])

REDUCE_COLORS - Reduces the number of colors used in an image by eliminating unused pixel values.
REDUCE_COLORS, *Image*, *Values*

REFORM - Changes array dimensions without changing the total number of elements.
Result = REFORM(*Array*, D_1, ..., D_8 [, /OVERWRITE])

REGRESS - Performs multiple linear regression fit.
Result = REGRESS(*X*, *Y*, *Weights* [, *Yfit*, *Const*, *Sigma*, *Ftest*, *R*, *Rmul*, *Chisq*, *Status*] [, /RELATIVE_WEIGHT])

REPEAT...UNTIL - Repeats statement(s) until expression evaluates to true. Subject is always executed at least once.

REPEAT *statement* UNTIL *expression*

or

REPEAT BEGIN

statements

ENDREP UNTIL *expression*

REPLICATE - Creates an array of given dimensions, filled with specified value.

Result = REPLICATE(*Value*, D_1 [, ..., D_8])

REPLICATE_INPLACE - Updates an array by replacing all or selected parts of it with a specified value.

REPLICATE_INPLACE, *X*, *Value* [, *D1*, *Loc1* [, *D2*, *Range*]]

RESOLVE_ALL - Compiles any uncompiled routines.

RESOLVE_ALL [, /CONTINUE_ON_ERROR] [, /QUIET]

RESOLVE_ROUTINE - Compiles a routine.

RESOLVE_ROUTINE, *Name* [, /EITHER | , /IS_FUNCTION] [, /NO_RECOMPLIE]

RESTORE - Restores IDL variables and routines saved in an IDL SAVE file.

RESTORE [, *Filename*] [, FILENAME=*name*] [, /RELAXED_STRUCTURE_ASSIGNMENT] [, RESTORED_OBJECTS=*variable*] [, /VERBOSE]

RETALL - Returns control to the main program level.

RETALL

RETURN - Returns control to the next-higher program level.

RETURN [, *Return_value*]

REVERSE - Reverses the order of rows or columns in an array.

Result = REVERSE(*Array* [, *Subscript_Index*])

REWIND (VMS only) - Rewinds tape on designated IDL tape unit.

REWIND, *Unit*

RIEMANN - Computes the Riemann sum.

RIEMANN, *P*, *A*, *Theta* [, /BACKPROJECT] [, /BILINEAR] [, CENTER=*value*] [, COR=*vector*] [, CUBIC=*value*{-1 to 0}] [, D=*spacing*] [, ROW=*value*]

RK4 - Solves differential equations using fourth-order Runge-Kutta method.

Result = RK4(*Y*, *Dydx*, *X*, *H*, *Derivs* [, /DOUBLE])

ROBERTS - Returns an approximation of Roberts edge enhancement.

Result = ROBERTS(*Image*)

ROT - Rotates an image by any amount.

Result = ROT(*A*, *Angle*, [*Mag*, X_0, Y_0] [, /INTERP] [, CUBIC=*value*{-1 to 0}] [, MISSING=*value*] [, /PIVOT])

ROTATE - Rotates/transposes an array in multiples of 90 degrees.

Result = ROTATE(*Array*, *Direction*)

ROUND - Returns the integer closest to its argument.

Result = ROUND(*X*)

ROUTINE_INFO - Provides information about compiled procedures and functions.

Result = ROUTINE_INFO([*Routine* [[, /PARAMETERS{must specify *Routine*}] [, /SOURCE] [, /UNRESOLVED] [, /VARIABLES] | , /SYSTEM]] [, /DISABLED] [, /ENABLED] [, /FUNCTIONS])

RS_TEST - Performs the Wilcoxon Rank-Sum test.

Result = RS_TEST(*X*, *Y* [, UX=*variable*] [, UY=*variable*])

S

S_TEST - Performs the Sign test.

Result = S_TEST(*X*, *Y* [, ZDIFF=*variable*])

SAVE - Saves variables, system variables, and IDL routines in a file for later use.

SAVE [, Var_1, ..., Var_n] [, /ALL] [, /COMM, /VARIABLES] [, /COMPRESS] [, FILENAME=*string*] [, ROUTINES] [, /SYSTEM_VARIABLES] [, /VERBOSE]

SCALE3 - Sets up axis ranges and viewing angles for 3D plots.

SCALE3 [, XRANGE=*vector*] [, YRANGE=*vector*] [, ZRANGE=*vector*] [, AX=*degrees*] [, AZ=*degrees*]

SCALE3D - Scales 3D unit cube into the viewing area.

SCALE3D

SEARCH2D - Finds “objects” or regions of similar data within a 2D array.

Result = SEARCH2D(*Array*, *Xpos*, *Ypos*, *Min_Val*, *Max_Val* [, /DECREASE, /INCREASE [, LPF_BAND=*integer*{≥3}]] [, /DIAGONAL])

SEARCH3D - Finds “objects” or regions of similar data values within a volume.

Result = SEARCH3D(*Array*, *Xpos*, *Ypos*, *Zpos*, *Min_Val*, *Max_Val* [, /DECREASE, /INCREASE [, LPF_BAND=*integer*{≥3}]] [, /DIAGONAL])

SET_PLOT - Sets the output device used by the IDL graphics procedures.

SET_PLOT, *Device* [, /COPY] [, /INTERPOLATE]

SET_SHADING - Sets the light source shading parameters.

SET_SHADING [, /GOURAUD] [, LIGHT=*[x, y, z]*] [, /REJECT] [, VALUES=*[darkest, brightest]*]

SET_SYMBOL (VMS Only) - Defines a VMS DCL interpreter symbol.

SET_SYMBOL, *Name*, *Value* [, TYPE={1 | 2}]

SETENV (UNIX/Windows Only) - Adds or changes an environment variable.

SETENV, *Environment_Expression*

SETLOG (VMS Only) - Defines a VMS logical name.
SETLOG, *Lognam*, *Value* [, /CONCEALED]
[, /CONFINE] [, /NO_ALIAS] [, TABLE=*string*]
[, /TERMINAL]

SETUP_KEYS - Sets function keys for use with UNIX versions of IDL.
SETUP_KEYS [, /EIGHTBIT] [, /SUN | , /VT200 | ,
/HP9000 | , /MIPS | , /PSTERM | , /SGI]
[, /APP_KEYPAD] [, /NUM_KEYPAD]

SFIT - Performs polynomial fit to a surface.
Result = SFIT(*Data*, *Degree* [, KX=*variable*])

SHADE_SURF - Creates a shaded-surface representation of gridded data.
SHADE_SURF, *Z* [, *X*, *Y*] [, AX=*degrees*] [, AZ=*degrees*]
[, IMAGE=*variable*] [, MAX_VALUE=*value*]
[, MIN_VALUE=*value*] [, PIXELS=*pixels*] [, /SAVE]
[, SHADES=*array*] [, /XLOG] [, /YLOG]
Graphics Keywords: [, CHARSIZE=*value*]
[, CHARTHICK=*integer*] [, CLIP=*[X_0, Y_0, X_1, Y_1]*]
[, COLOR=*value*][, /DATA | , /DEVICE | , /NORMAL]
[, FONT=*integer*] [, /NOCLIP] [, /NODATA]
[, POSITION=*[X_0, Y_0, X_1, Y_1]*] [, SUBTITLE=*string*]
[, /T3D] [, THICK=*value*] [, TICKLEN=*value*]
[, TITLE=*string*]
[, {X | Y | Z}CHARSIZE=*value*]
[, {X | Y | Z}GRIDSTYLE=*integer*{0 to 5}]
[, {X | Y | Z}MARGIN=*[left, right]*]
[, {X | Y | Z}MINOR=*integer*]
[, {X | Y | Z}RANGE=*[min, max]*]
[, {X | Y | Z}STYLE=*value*] [, {X | Y | Z}THICK=*value*]
[, {X | Y | Z}TICKFORMAT=*string*]
[, {X | Y | Z}TICKLEN=*value*]
[, {X | Y | Z}TICKNAME=*string_array*]
[, {X | Y | Z}TICKS=*integer*] [, {X | Y | Z}TICKV=*array*]
[, {X | Y | Z}TICK_GET=*variable*]
[, {X | Y | Z}TITLE=*string*] [, ZVALUE=*value*{0 to 1}]

SHADE_SURF_IRR - Creates a shaded-surface representation of an irregularly gridded dataset.
SHADE_SURF_IRR, *Z*, *X*, *Y* [, AX=*degrees*]
[, AZ=*degrees*] [, IMAGE=*variable*] [, PLIST=*variable*]
[, /T3D]

SHADE_VOLUME - Contours a volume to create a list of vertices and polygons that can be displayed using POLYSHADE.
SHADE_VOLUME, *Volume*, *Value*, *Vertex*, *Poly* [, /LOW]
[, SHADES=*array*] [, /VERBOSE] [, XRANGE=*vector*]
[, YRANGE=*vector*] [, ZRANGE=*vector*]

SHIFT - Shifts elements of vectors or arrays by a specified number of elements.
Result = SHIFT(*Array*, S_1, ..., S_n)

SHOW3 - Displays array as image, surface plot, and contour plot simultaneously.
SHOW3, *Image* [, *X*, *Y*] [, /INTERP]
[, E_CONTOUR=*structure*] [, E_SURFACE=*structure*]
[, SSCALE=*scale*]

SHOWFONT - Displays a TrueType or vector font
SHOWFONT, *Font*, *Name* [, /ENCAPSULATED]
[, /TT_FONT]

SIN - Returns the trigonometric sine of *X*.
Result = SIN(*X*)

SINDGEN - Returns a string array with each element set to its subscript.
Result = SINDGEN(D_1, ..., D_8)

SINH - Returns the hyperbolic sine of X.
Result = SINH(*X*)

SIZE - Returns array size and type information.
Result = SIZE(*Expression* [, /DIMENSIONS | ,
/FILE_LUN | , /N_DIMENSIONS | , /N_ELEMENTS | ,
/STRUCTURE | , /TNAME | , /TYPE])

SKEWNESS - Computes statistical skewness of an *n*-element vector.
Result = SKEWNESS(*X* [, /DOUBLE] [, /NAN])

SKIPF - Skips records or files on the designated magnetic tape unit.
SKIPF, *Unit*, *Files* or SKIPF, *Unit*, *Records*, *R*

SLICER3 - Interactive volume visualization tool.
SLICER3 [, *hData3D*]
[, DATA_NAMES=*string/string_array*] [, /DETACH]
[, GROUP=*widget_id*] [, /MODAL]

SLIDE_IMAGE - Creates a scrolling graphics window for examining large images.
SLIDE_IMAGE [, *Image*] [, /BLOCK] [, CONGRID=0]
[, FULL_WINDOW=*variable*] [, GROUP=*widget_id*]
[, /ORDER] [, /REGISTER] [, RETAIN={0 | 1 | 2}]
[, SLIDE_WINDOW=*variable*] [, SHOW_FULL=0]
[, TITLE=*string*] [, TOP_ID=*variable*] [, XSIZE=*width*]
[, XVISIBLE=*width*] [, YSIZE=*height*]
[, YVISIBLE=*height*]

SMOOTH - Smooths with a boxcar average.
Result = SMOOTH(*Array*, *Width* [, /EDGE_TRUNCATE]
[, /NAN])

SOBEL - Implements Sobel edge-enhancement.
Result = SOBEL(*Image*)

SORT - Sorts the elements of an array in ascending order.
Result = SORT(*Array*)

SPAWN - Spawns child process for access to operating system.
SPAWN [, *Command(s)* [, *Result*]]
Keywords (all platforms): [, COUNT=*variable*]
[, PID=*variable*]
Macintosh Keywords: [, MACCREATOR=*string*]

UNIX Keywords: [, /NOSHELL] [, /NOTTYRESET] [, /SH] [, /UNIT{*Command* required, *Result* not allowed}]
VMS Keywords: [, /NOCLISYM] [, /NOLOGNAM] [[, /NOTIFY] , /NOWAIT]

SPH_4PNT - Returns center and radius of a sphere given 4 points.
SPH_4PNT, *X*, *Y*, *Z*, *Xc*, *Yc*, *Zc*, *R*

SPH_SCAT - Performs spherical gridding.
Result = SPH_SCAT(*Lon*, *Lat*, *F* [, BOUNDS=*[lonmin, latmin, lonmax, latmax]*] [, BOUT=*variable*] [, GOUT=*variable*] [, GS=*[lonspacing, latspacing]*] [, NLON=*value*] [, NLAT=*value*])

SPL_INIT - Establishes the type of interpolating spline.
Result = SPL_INIT(*X*, *Y* [, /DOUBLE] [, YP0=*value*] [, YPN_1=*value*])

SPL_INTERP - Performs cubic spline interpolation (Numerical Recipes).
Result = SPL_INTERP(*X*, *Y*, *Y2*, *X2* [, /DOUBLE])

SPLINE - Performs cubic spline interpolation.
Result = SPLINE(*X*, *Y*, *T* [, *Sigma*])

SPLINE_P - Performs parametric cubic spline interpolation.
SPLINE_P, *X*, *Y*, *Xr*, *Yr* [, INTERVAL=*value*] [, TAN0=*[X_0, Y_0]*] [, TAN1=*[X_{n-1}, Y_{n-1}]*]

SPRSAB - Performs matrix multiplication on sparse matrices.
Result = SPRSAB(*A*, *B* [, /DOUBLE] [, THRESH=*value*])

SPRSAX - Multiplies sparse matrix by a vector.
Result = SPRSAX(*A*, *X* [, /DOUBLE])

SPRSIN - Converts matrix to row-index sparse matrix.
Result = SPRSIN(*A* [, /COLUMN] [, /DOUBLE] [, THRESH=*value*]) or
Result = SPRSIN(*Columns*, *Rows*, *Values*, *N* [, /DOUBLE] [, THRESH=*value*])

SQRT - Returns the square root of *X*.
Result = SQRT(*X*)

STANDARDIZE - Computes standardized variables.
Result = STANDARDIZE(*A* [, /DOUBLE])

STDDEV - Computes the standard deviation of an *n*-element vector.
Result = STDDEV(*X* [, /DOUBLE] [, /NAN])

STOP - Stops the execution of a running program or batch file.
STOP [, *$Expr_1$*, ..., *$Expr_n$*]

STRARR - Returns string array containing zero-length strings.
Result = STRARR(D_1, ..., D_8)

STRCMP - Compares two strings.
Result = STRCMP(*String1*, *String2* [, *N*] [, /FOLD_CASE])

STRCOMPRESS - Removes whitespace from a string.
Result = STRCOMPRESS(*String* [, /REMOVE_ALL])

STREAMLINE - Generates the visualization graphics from a path.
STREAMLINE, *Verts*, *Conn*, *Normals*, *Outverts*, *Outconn* [, ANISOTROPY=*array*] [, SIZE=*vector*] [, PROFILE=*array*]

STREGEX - Performs regular expression matching.
Result = STREGEX(*StringExpression*, *RegularExpression* [, /BOOLEAN | , /EXTRACT | , LENGTH=*variable* [, /SUBEXPR]] [, /FOLD_CASE])

STRETCH - Stretches color table for contrast enhancement.
STRETCH [, *Low*, *High* [, *Gamma*]] [, /CHOP]

STRING - Converts its arguments to string type.
Result = STRING(*$Expression_1$*, ..., *$Expression_n$* [, AM_PM=*[string, string]*] [, DAYS_OF_WEEK=*string_array*{7 names}] [, FORMAT=*value*] [, MONTHS=*string_array*{12 names}] [, /PRINT])

STRJOIN - Collapses a string scalar or array into merged strings.
Result = STRJOIN(*String* [, *Delimiter*] [, /SINGLE])

STRLEN - Returns the length of a string.
Result = STRLEN(*Expression*)

STRLOWCASE - Converts a string to lower case.
Result = STRLOWCASE(*String*)

STRMATCH - Compares search string against input string expression.
Result = STRMATCH(*String*, *SearchString* [, /FOLD_CASE])

STRMESSAGE - Returns the text of an error number.
Result = STRMESSAGE(*Err* [, /BLOCK | , /CODE | , /NAME])

STRMID - Extracts a substring from a string.
Result = STRMID(*Expression*, *First_Character* [, *Length*] [, /REVERSE_OFFSET])

STRPOS - Finds first occurrence of a substring within a string.
Result = STRPOS(*Expression*, *Search String* [, *Pos*] [, /REVERSE_OFFSET] [, /REVERSE_SEARCH])

STRPUT - Inserts the contents of one string into another.
STRPUT, *Destination*, *Source* [, *Position*]

STRSPLIT - Splits its input string argument into separate substrings, according to the specified pattern.
Result = STRSPLIT(*String* [, *Pattern*] [, ESCAPE=*string* | , /REGEX [, /FOLD_CASE]] [, /EXTRACT | , LENGTH=*variable*] [, /PRESERVE_NULL])

STRTRIM - Removes leading and/or trailing blanks from string.
Result = STRTRIM(*String* [, *Flag*])

STRUCT_ASSIGN - Performs “relaxed structure assignment” to copy a structure.
STRUCT_ASSIGN, *Source*, *Destination* [, /NOZERO] [, /VERBOSE]

STRUCT_HIDE - Prevents the IDL HELP procedure from displaying information about structures or objects.
STRUCT_HIDE, *Arg*$_1$ [, *Arg*$_2$, ..., *Arg*$_n$]

STRUPCASE - Converts a string to upper case.
Result = STRUPCASE(*String*)

SURFACE - Plots an array as a wireframe mesh surface.
SURFACE, *Z* [, *X*, *Y*] [, AX=*degrees*] [, AZ=*degrees*] [, BOTTOM=*index*] [, /HORIZONTAL] [, /LEGO] [, /LOWER_ONLY | , /UPPER_ONLY] [, MAX_VALUE=*value*] [, MIN_VALUE=*value*] [, /SAVE] [, SHADES=*array*] [, SKIRT=*value*] [, /XLOG] [, /YLOG] [, ZAXIS={1 | 2 | 3 | 4}] [, /ZLOG]
Graphics Keywords: Accepts all graphics keywords accepted by PLOT except for: PSYM, SYMSIZE.

SURFR - Sets up 3D transformations by duplicating rotation, translation, and scaling of SURFACE.
SURFR [, AX=*degrees*] [, AZ=*degrees*]

SVDC - Computes Singular Value Decomposition of an array.
SVDC, *A*, *W*, *U*, *V* [, /COLUMN] [, /DOUBLE]

SVDFIT - Performs least squares fit using SVD method.
Result = SVDFIT(*X*, *Y* [, *M*] [, A=*vector*] [, CHISQ=*variable*] [, COVAR=*variable*] [, /DOUBLE] [, FUNCTION_NAME=*string*] [, /LEGENDRE] [, SIGMA=*variable*] [, SINGULAR=*variable*] [, VARIANCE=*variable*] [, WEIGHTS=*vector*] [, YFIT=*variable*])

SVSOL - Solves set of linear equations using back-substitution.
Result = SVSOL(*U*, *W*, *V*, *B* [, /COLUMN] [, /DOUBLE])

SWAP_ENDIAN - Reverses the byte ordering of scalars, arrays or structures.
Result = SWAP_ENDIAN(*Variable*)

SYSTIME - Returns the current system time.
Result = SYSTIME(*Arg*) [, /JULIAN] [, /SECONDS]

T

T_CVF - Computes the cutoff value in a Student's t distribution.
Result = T_CVF(*P, Df*)

T_PDF - Computes Student's t distribution.
Result = T_PDF(*V, Df*)

T3D - Performs various 3D transformations.
T3D [, OBLIQUE=*vector*] [, PERSPECTIVE=*p*{eye at (0,0,*p*)}] [, /RESET] [, ROTATE=*[x, y, z]*] [, SCALE=*[x, y, z]*] [, TRANSLATIONS=*[x, y, z]*] [, /XYEXCH | , /XZEXCH | , /YZEXCH]

TAG_NAMES - Returns the names of tags in a structure.
Result = TAG_NAMES(*Expression* [, /STRUCTURE_NAME])

TAN - Returns the tangent of *X*.
Result = TAN(*X*)

TANH - Returns the hyperbolic tangent of X.
Result = TANH(*X*)

TAPRD - Reads the next record on a tape.
TAPRD, *Array, Unit* [, *Byte_Reverse*]

TAPWRT - Writes data to a tape.
TAPWRT, *Array, Unit* [, *Byte_Reverse*]

TEK_COLOR - Loads color table based on Tektronix printer.
TEK_COLOR [, *Start_Index, Colors*]

TEMPORARY - Returns a temporary copy of a variable, and sets the original variable to "undefined".
Result = TEMPORARY(*Variable*)

TETRA_CLIP - Clips a tetrahedral mesh to an arbitrary plane in space and returns a tetrahedral mesh of the remaining portion.
Result = TETRA_CLIP (*Plane*, *Vertsin*, *Connin*, *Vertsout*, *Connout* [, AUXDATA_IN=*array*, AUXDATA_OUT=*variable*] [, CUT_VERTS=*variable*])

TETRA_SURFACE - Extracts a polygonal mesh as the exterior surface of a tetrahedral mesh.
Result = TETRA_SURFACE (*Verts*, *Connin*)

TETRA_VOLUME - Computes properties of tetrahedral mesh array.
Result = TETRA_VOLUME (*Verts*, *Conn* [, AUXDATA=*array*] [, MOMENT=*variable*])

THIN - Returns the "skeleton" of a bi-level image.
Result = THIN(*Image*[, /NEIGHBOR_COUNT] [, /PRUNE])

THREED - Plots a 2D array as a pseudo 3D plot.
THREED, *A* [, *Sp*] [, TITLE=*string*] [, XTITLE=*string*] [, YTITLE=*string*]

TIME_TEST2 - Performs speed benchmarks for IDL.
TIME_TEST2 [, *Filename*]

TM_TEST - Performs t-means test.
Result = TM_TEST(*X*, *Y* [, /PAIRED] [, /UNEQUAL])

TOTAL - Sums of the elements of an array.
Result = TOTAL(*Array* [, *Dimension*] [, /CUMULATIVE] [, /DOUBLE] [, /NAN])

TrackBall Object - See "TrackBall" on page 63.

TRACE - Computes the trace of an array.
Result = TRACE(*A* [, /DOUBLE])

TRANSPOSE - Transposes an array.
Result = TRANSPOSE(*Array* [, *P*])

TRI_SURF - Interpolates gridded set of points with a smooth quintic surface.
Result = TRI_SURF(*Z* [, *X*, *Y*] [, /EXTRAPOLATE] [, MISSING=*value*] [, /REGULAR] [, XGRID=*[xstart, xspacing]* | [, XVALUES=*array*]] [, YGRID=*[yxstart,*

yspacing] | [, YVALUES=*array*]] [, GS=*[xspacing, yspacing]*] [, BOUNDS=*[xmin, ymin, xmax, ymax]*] [, NX=*value*] [, NY=*value*])

TRIANGULATE - Constructs Delaunay triangulation of a planar set of points.
TRIANGULATE, *X, Y, Triangles* [, *B*] [, CONNECTIVITY=*variable*] [, /DEGREES] [, FVALUE=*variable*] [, REPEATS=*variable*] [, SPHERE=*variable*]

TRIGRID - Interpolates irregularly-gridded data to a regular grid.
Result = TRIGRID(*X, Y, Z, Triangles* [, *GS, Limits*])
For spherical gridding: *Result* = TRIGRID(*F* , *GS, Limits*, SPHERE=*S*)
Keywords: [, /DEGREES] [, EXTRAPOLATE=*array* | , /QUINTIC] [, INPUT=*variable*] [, MAX_VALUE=*value*] [, MIN_VALUE=*value*] [, MISSING=*value*] [, NX=*value*] [, NY=*value*] [, SPHERE=*variable*] [, XGRID=*variable*] [, YGRID=*variable*]

TRIQL - Determines eigenvalues and eigenvectors of tridiagonal array.
TRIQL, *D, E, A* [, /DOUBLE]

TRIRED - Reduces a real, symmetric array to tridiagonal form.
TRIRED, *A, D, E* [, /DOUBLE]

TRISOL - Solves tridiagonal systems of linear equations.
Result = TRISOL(*A, B, C, R* [, /DOUBLE])

TRNLOG (VMS Only) - Searches the VMS logical name tables for a specified logical name.
Result = TRNLOG(*Lognam, Value* [, ACMODE={0 | 1 | 2 | 3}] [, /FULL_TRANSLATION] [, /ISSUE_ERROR] [, RESULT_ACMODE=*variable*] [, RESULT_TABLE=*variable*] [, TABLE=*string*])

TS_COEF - Computes the coefficients for autoregressive time-series.
Result = TS_COEF(*X, P* [, MSE=*variable*])

TS_DIFF - Computes the forward differences of a time-series.
Result = TS_DIFF(*X, K* [, /DOUBLE])

TS_FCAST - Computes future or past values of a stationary time-series.
Result = TS_FCAST(*X, P, Nvalues* [, /BACKCAST] [, /DOUBLE])

TS_SMOOTH - Computes moving averages of a time-series.
Result = TS_SMOOTH(*X, Nvalues* [, /BACKWARD] [, /DOUBLE] [, /FORWARD] [, ORDER=*value*])

TV - Displays an image.
TV, *Image* [, *Position*]
or
TV, *Image* [, *X, Y* [, *Channel*]]
Keywords: [, /CENTIMETERS | , /INCHES] [, CHANNEL=*value*] [, /ORDER] [, TRUE={1 | 2 | 3}] [, /WORDS] [, XSIZE=*value*] [, YSIZE=*value*] [, /DATA | , /DEVICE | , /NORMAL] [, /T3D | Z=*value*]

TVCRS - Manipulates the image display cursor.
TVCRS [, *ON_OFF*]
or
TVCRS [, *X, Y*]
Keywords: [, /CENTIMETERS | , /INCHES] [, /HIDE_CURSOR] [, /DATA | , /DEVICE | , /NORMAL] [, /T3D | Z=*value*]

TVLCT - Loads display color tables.
TVLCT, V_1, V_2, V_3 [, *Start*] [, /GET] [, /HLS | , /HSV]
or
TVLCT, *V* [, *Start*] [, /GET] [, /HLS | , /HSV]

TVRD - Reads an image from a window into a variable.
Result = TVRD([X_0 [, Y_0 [, N_x [, N_y [, *Channel*]]]]] [, CHANNEL=*value*] [, /ORDER] [, TRUE={1 | 2 | 3}] [, /WORDS])

TVSCL - Scales and displays an image.
TVSCL, *Image* [, *Position*]
or
TVSCL, *Image* [, *X, Y* [, *Channel*]]
Keywords: [, /CENTIMETERS | , /INCHES] [, CHANNEL=*value*] [, /NAN] [, /ORDER] [, TOP=*value*] [, TRUE={1 | 2 | 3}] [, /WORDS] [, XSIZE=*value*] [, YSIZE=*value*] [, /DATA | , /DEVICE | , /NORMAL] [, /T3D | Z=*value*]

U

UINDGEN - Returns unsigned integer array with each element set to its subscript.
Result = UINDGEN(D_1, ..., D_8)

UINT - Converts argument to unsigned integer type.
Result = UINT(*Expression*[, *Offset* [, Dim_1, ..., Dim_8]])

UINTARR - Returns an unsigned integer vector or array.
Result = UINTARR(D_1, ..., D_8 [, /NOZERO])

UL64INDGEN - Returns an unsigned 64-bit integer array with each element set to its subscript.
Result = UL64INDGEN(D_1, ..., D_8)

ULINDGEN - Returns an unsigned longword array with each element set to its subscript.
Result = ULINDGEN(D_1, ..., D_8)

ULON64ARR - Returns an unsigned 64-bit integer vector or array.
Result = ULON64ARR(D_1, ..., D_8)

ULONARR - Returns an unsigned longword integer vector or array.
Result = ULONARR(D_1, ..., D_8 [, /NOZERO])

ULONG - Converts argument to unsigned longword integer type.
Result = ULONG(*Expression*[, *Offset* [, Dim_1, ..., Dim_8]])

ULONG64 - Converts argument to unsigned 64-bit integer type.
Result = ULONG64(*Expression*[, *Offset* [, Dim_1, ..., Dim_8]])

UNIQ - Returns subscripts of the unique elements in an array.
Result = UNIQ(*Array* [, *Index*])

USERSYM - Defines a new plotting symbol.
USERSYM, *X* [, *Y*] [, COLOR=*value*] [, /FILL]
[, THICK=*value*]

V

VALUE_LOCATE - Finds the intervals within a given monotonic vector that brackets a given set of one or more search values.
Result = VALUE_LOCATE (*Vector*, *Value*)

VARIANCE - Computes the statistical variance of an *n*-element vector.
Result = VARIANCE(*X* [, /DOUBLE] [, /NAN])

VAX_FLOAT - Determines the default value for the VAX_FLOAT keyword or if an open file unit has the VAX_FLOAT attribute set.
Result = VAX_FLOAT([*Default*] [, FILE_UNIT=*lun*])

VECTOR_FIELD - Places colored, oriented vectors of specified length at each vertex in an input vertex array.
VECTOR_FIELD, *Field*, *Outverts*, *Outconn*
[, ANISOTROPY=*array*] [, SCALE=*value*]
[, VERTICES=*array*]

VEL - Draws a velocity (flow) field with streamlines.
VEL, *U*, *V* [, NVECS=*value*] [, XMAX=
value{*xsize*/*ysize*}] [, LENGTH=*value*{*longest*/*steps*}]
[, NSTEPS=*value*] [, TITLE=*string*]

VELOVECT - Draws a 2D velocity field plot.
VELOVECT, *U*, *V* [, *X*, *Y*] [, COLOR=*index*]
[, MISSING=*value* [, /DOTS]] [, LENGTH=*value*]
[Also accepts all PLOT keywords]

VERT_T3D - Transforms a 3D array by a 4x4 transformation matrix.
Result = VERT_T3D(*Vertex_List* [, MATRIX=*4x4_array*]
[, /NO_COPY] [, /NO_DIVIDE
[, SAVE_DIVIDE=*variable*]])

VOIGT - Calculates intensity of atomic absorption line (Voight) profile.
Result = VOIGT(*A*, *U*)

VORONOI - Computes Voronoi polygon given Delaunay triangulation.
VORONOI, *X*, *Y*, *I0*, *C*, *Xp*, *Yp*, *Rect*

VOXEL_PROJ - Generates volume visualizations using voxel technique.
Result = VOXEL_PROJ(*V* [, *RGBO*]
[, BACKGROUND=*array*] [, CUTTING_PLANE=*array*]
[, /INTERPOLATE] [, /MAXIMUM_INTENSITY]
[, STEP=*[Sx, Sy, Sz]*] [, XSIZE=*pixels*] [, YSIZE=*pixels*]
[, ZBUFFER=*int_array*] [, ZPIXELS=*byte_array*])

W

WAIT - Suspends execution of an IDL program for a specified period.
WAIT, *Seconds*

WARP_TRI - Warps an image using control points.
Result = WARP_TRI(*Xo*, *Yo*, *Xi*, *Yi*, *Image*
[, OUTPUT_SIZE=*vector*] [, /QUINTIC]
[, /EXTRAPOLATE])

WATERSHED - Applies the morphological watershed operator to a grayscale image.
Result = WATERSHED (*Image*
[, CONNECTIVITY={4 | 8}])

WDELETE - Deletes IDL graphics windows.
WDELETE [, *Window_Index* [, ...]]

WEOF - Writes an end-of-file mark on the designated tape unit.
WEOF, *Unit*

WF_DRAW - Draws weather fronts with smoothing.
WF_DRAW, *X*, *Y* [[, /COLD | , FRONT_TYPE=1] |
[, /WARM | , FRONT_TYPE=2] | [, /OCCLUDED | ,
FRONT_TYPE=3] | [, /STATIONARY | ,
FRONT_TYPE=4] | [, /CONVERGENCE | ,
FRONT_TYPE=5]] [, COLOR=*value*] [, /DATA | ,
/DEVICE | , /NORM] [, INTERVAL=*value*]
[, PSYM=*value*] [, SYM_HT=*value*]
[, SYM_LEN=*value*] [, THICK=*value*]

WHERE - Returns subscripts of nonzero array elements.
Result = WHERE(*Array_Expression* [, *Count*])

WHILE...DO - Performs statement(s) as long as expression evaluates to true. Subject is never executed if condition is initially false.
WHILE *expression* DO *statement*
or
WHILE *expression* DO BEGIN
statements
ENDWHILE

WIDGET_BASE - Creates base widget (containers for other widgets).
Result = WIDGET_BASE([*Parent*] [, /ALIGN_BOTTOM
| , /ALIGN_CENTER | , /ALIGN_LEFT | ,
/ALIGN_RIGHT | , /ALIGN_TOP]
[, APP_MBAR=*variable*{same as mbar on Windows and
Motif} | , /MBAR | , /MODAL]
[, /BASE_ALIGN_BOTTOM | ,
/BASE_ALIGN_CENTER | , /BASE_ALIGN_LEFT | ,
/BASE_ALIGN_RIGHT | , /BASE_ALIGN_TOP]
[, /COLUMN | , /ROW] [, EVENT_FUNC=*string*]
[, EVENT_PRO=*string*] [, /EXCLUSIVE | ,
/NONEXCLUSIVE] [, /FLOATING] [, FRAME=*width*]
[, FUNC_GET_VALUE=*string*] [, /GRID_LAYOUT]
[, GROUP_LEADER=*widget_id*{must specify for modal
dialogs}] [, /KBRD_FOCUS_EVENTS]

[, KILL_NOTIFY=*string*] [, /MAP{not for modal bases}]
[, /NO_COPY] [, NOTIFY_REALIZE=*string*]
[, PRO_SET_VALUE=*string*] [, SCR_XSIZE=*width*]
[, SCR_YSIZE=*height*] [, /SCROLL{not for modal bases}] [, /SENSITIVE] [, SPACE=*value*{ignored if exclusive or nonexclusive}] [, TITLE=*string*]
[, TLB_FRAME_ATTR=*value*{top-level bases only}]
[, /TLB_KILL_REQUEST_EVENTS{top-level bases only}] [, /TLB_SIZE_EVENTS{top-level bases only}]
[, /TRACKING_EVENTS] [, UNAME=*string*]
[, UNITS={0 | 1 | 2}] [, UVALUE=*value*]
[, XOFFSET=*value*] [, XPAD=*value*{ignored if exclusive or nonexclusive}] [, XSIZE=*value*]
[, X_SCROLL_SIZE=*value*] [, YOFFSET=*value*]
[, YPAD=*value*{ignored if exclusive or nonexclusive}]
[, YSIZE=*value*] [, Y_SCROLL_SIZE=*value*])
X Windows Keywords: [, DISPLAY_NAME=*string*]
[, RESOURCE_NAME=*string*]
[, RNAME_MBAR=*string*]

WIDGET_BUTTON - Creates button widgets.

Result = WIDGET_BUTTON(*Parent*
[, /ALIGN_CENTER | , /ALIGN_LEFT | , /ALIGN_RIGHT] [, /BITMAP] [, /DYNAMIC_RESIZE]
[, EVENT_FUNC=*string*] [, EVENT_PRO=*string*]
[, FONT=*value*] [, FRAME=*width*]
[, FUNC_GET_VALUE=*string*]
[, GROUP_LEADER=*widget_id*] [, /HELP]
[, KILL_NOTIFY=*string*] [, /MENU] [, /NO_COPY]
[, /NO_RELEASE] [, NOTIFY_REALIZE=*string*]
[, PRO_SET_VALUE=*string*] [, SCR_XSIZE=*width*]
[, SCR_YSIZE=*height*] [, /SENSITIVE] [, /SEPARATOR]
[, /TRACKING_EVENTS] [, UNAME=*string*]
[, UNITS={0 | 1 | 2}] [, UVALUE=*value*]
[, VALUE=*value*] [, X_BITMAP_EXTRA=*bits*]
[, XOFFSET=*value*] [, XSIZE=*value*] [, YOFFSET=*value*]
[, YSIZE=*value*])
X Windows Keywords: [, RESOURCE_NAME=*string*]

WIDGET_CONTROL - Realizes, manages, and destroys widgets.

WIDGET_CONTROL [, *Widget_ID*]
All widgets: [, BAD_ID=*variable*] [, /CLEAR_EVENTS]
[, DEFAULT_FONT=*string*{do not specify *Widget_ID*}]
[, /DELAY_DESTROY{do not specify *Widget_ID*}]
[, /DESTROY] [, EVENT_FUNC=*string*]
[, EVENT_PRO=*string*] [, FUNC_GET_VALUE=*string*]
[, GET_UVALUE=*variable*]
[, GROUP_LEADER=*widget_id*] [, /HOURGLASS{do not specify *Widget_ID*}] [, KILL_NOTIFY=*string*]
[, /MAP] [, /NO_COPY] [, NOTIFY_REALIZE=*string*]
[, PRO_SET_VALUE=*string*] [, /REALIZE]
[, /RESET{do not specify *Widget_ID*}]
[, SCR_XSIZE=*width*] [, SCR_YSIZE=*height*]
[, SEND_EVENT=*structure*] [, /SENSITIVE]
[, SET_UNAME=*string*] [, SET_UVALUE=*value*]
[, /SHOW] [, TIMER=*value*]
[, TLB_GET_OFFSET=*variable*]
[, TLB_GET_SIZE=*variable*]
[, /TLB_KILL_REQUEST_EVENTS]
[, TLB_SET_TITLE=*string*]
[, TLB_SET_XOFFSET=*value*]
[, TLB_SET_YOFFSET=*value*]
[, /TRACKING_EVENTS] [, UNITS={0 | 1 | 2}]
[, /UPDATE] [, XOFFSET=*value*] [, XSIZE=*value*]
[, YOFFSET=*value*] [, YSIZE=*value*]
widget_base: [, CANCEL_BUTTON=*widget_id*{for modal bases}] [, DEFAULT_BUTTON=*widget_id*{for modal bases}] [, /ICONIFY]
[, /KBRD_FOCUS_EVENTS]
[, /TLB_KILL_REQUEST_EVENTS]
widget_button: [, /BITMAP] [, /DYNAMIC_RESIZE]
[, GET_VALUE=*value*] [, /INPUT_FOCUS]
[, /SET_BUTTON] [, SET_VALUE=*value*]
[, X_BITMAP_EXTRA=*bits*]
widget_draw: [, /DRAW_BUTTON_EVENTS]
[, /DRAW_EXPOSE_EVENTS]
[, /DRAW_MOTION_EVENTS]
[, /DRAW_VIEWPORT_EVENTS]
[, DRAW_XSIZE=*integer*] [, DRAW_YSIZE=*integer*]
[, GET_DRAW_VIEW=*variable*]
[, GET_UVALUE=*variable*] [, GET_VALUE=*variable*]
[, /INPUT_FOCUS] [, SET_DRAW_VIEW=*[x, y]*]
widget_droplist: [, /DYNAMIC_RESIZE]
[, SET_DROPLIST_SELECT=*integer*]
[, SET_VALUE=*value*]
widget_label: [, /DYNAMIC_RESIZE]
[, GET_VALUE=*value*] [, SET_VALUE=*value*]
widget_list: [, SET_LIST_SELECT=*value*]
[, SET_LIST_TOP=*integer*] [, SET_VALUE=*value*]
widget_slider: [, GET_VALUE=*value*]
[, SET_SLIDER_MAX=*value*]
[, SET_SLIDER_MIN=*value*] [, SET_VALUE=*value*]
widget_table: [, ALIGNMENT={0 | 1 | 2}]
[, /ALL_TABLE_EVENTS] [, AM_PM=*[string, string]*]
[, COLUMN_LABELS=*string_array*]
[, COLUMN_WIDTHS=*array*]
[, DAYS_OF_WEEK=*string_array*{7 names}]
[, /DELETE_COLUMNS{not for row_major mode}]
[, /DELETE_ROWS{not for column_major mode}]
[, /EDITABLE] [, EDIT_CELL=*[integer, integer]*]
[, FORMAT=*value*] [, GET_VALUE=*variable*]
[, INSERT_COLUMNS=*value*][, INSERT_ROWS=*value*]
[, /KBRD_FOCUS_EVENTS]
[, MONTHS=*string_array*{12 names}]
[, ROW_LABELS=*string_array*]
[, ROW_HEIGHTS=*array*]
[, SET_TABLE_SELECT=*[left, top, right, bottom]*]
[, SET_TABLE_VIEW=*[integer, integer]*]
[, SET_TEXT_SELECT=*[integer, integer]*]

[, SET_VALUE=*value*] [, TABLE_XSIZE=*columns*]
[, TABLE_YSIZE=*rows*] [, /USE_TABLE_SELECT | ,
USE_TABLE_SELECT=*[left, top, right, bottom]*]
[, /USE_TEXT_SELECT]
widget_text: [, /ALL_TEXT_EVENTS] [, /APPEND]
[, /EDITABLE] [, GET_VALUE=*variable*]
[, /INPUT_FOCUS] [, /KBRD_FOCUS_EVENTS]
[, /NO_NEWLINE] [, SET_TEXT_SELECT=*[integer, integer]*] [, SET_TEXT_TOP_LINE=*line_number*]
[, SET_VALUE=*value*] [, /USE_TEXT_SELECT]

WIDGET_DRAW - Creates drawable widgets.
Result = WIDGET_DRAW(*Parent* [, /APP_SCROLL]
[, /BUTTON_EVENTS] [, /COLOR_MODEL]
[, COLORS=*integer*] [, EVENT_FUNC=*string*]
[, EVENT_PRO=*string*] [, /EXPOSE_EVENTS]
[, FRAME=*width*] [, FUNC_GET_VALUE=*string*]
[, GRAPHICS_LEVEL=2]
[, GROUP_LEADER=*widget_id*]
[, KILL_NOTIFY=*string*] [, /MOTION_EVENTS]
[, /NO_COPY] [, NOTIFY_REALIZE=*string*]
[, PRO_SET_VALUE=*string*] [, RENDERER={0 | 1}]
[, RESOURCE_NAME=*string*] [, RETAIN={0 | 1 | 2}]
[, SCR_XSIZE=*width*] [, SCR_YSIZE=*height*]
[, /SCROLL] [, /SENSITIVE] [, /TRACKING_EVENTS]
[, UNAME=*string*] [, UNITS={0 | 1 | 2}]
[, UVALUE=*value*] [, VALUE=*value*]
[, /VIEWPORT_EVENTS] [, XOFFSET=*value*]
[, XSIZE=*value*] [, X_SCROLL_SIZE=*width*]
[, YOFFSET=*value*] [, YSIZE=*value*]
[, Y_SCROLL_SIZE=*height*])

WIDGET_DROPLIST - Creates droplist widgets.
Result = WIDGET_DROPLIST(*Parent*
[, /DYNAMIC_RESIZE] [, EVENT_FUNC=*string*]
[, EVENT_PRO=*string*] [, FUNC_GET_VALUE=*string*]
[, GROUP_LEADER=*widget_id*]
[, KILL_NOTIFY=*string*] [, /NO_COPY]
[, NOTIFY_REALIZE=*string*]
[, PRO_SET_VALUE=*string*]
[, RESOURCE_NAME=*string*] [, SCR_XSIZE=*width*]
[, SCR_YSIZE=*height*] [, /SENSITIVE] [, TITLE=*string*]
[, /TRACKING_EVENTS] [, UNAME=*string*]
[, UNITS={0 | 1 | 2}] [, UVALUE=*value*]
[, VALUE=*value*] [, XOFFSET=*value*] [, XSIZE=*value*]
[, YOFFSET=*value*] [, YSIZE=*value*])

WIDGET_EVENT - Returns events for the widget hierarchy.
Result = WIDGET_EVENT([*Widget_ID*])
[, BAD_ID=*variable*] [, /NOWAIT]
[, /SAVE_HOURGLASS]
UNIX Keywords: [, /YIELD_TO_TTY]

WIDGET_INFO - Obtains information about widgets.
Result = WIDGET_INFO([*Widget_ID*])
All widgets: [, /ACTIVE] [, /CHILD] [, /EVENT_FUNC]
[, /EVENT_PRO] [, FIND_BY_UNAME=*string*]
[, /GEOMETRY] [, /KBRD_FOCUS_EVENTS]
[, MANAGED=*variable*] [, /NAME] [, /PARENT]
[, /REALIZED] [, /SIBLING] [, /TRACKING_EVENTS]
[, /TYPE] [, UNITS={0 | 1 | 2}] [, /UNAME] [, /UPDATE]
[, /VALID_ID] [, /VERSION]
widget_base: [, /MODAL]
[, /TLB_KILL_REQUEST_EVENTS]
widget_button: [, /DYNAMIC_RESIZE]
widget_draw: [, /DRAW_BUTTON_EVENTS]
[, /DRAW_EXPOSE_EVENTS]
[, /DRAW_MOTION_EVENTS]
[, /DRAW_VIEWPORT_EVENTS]
widget_droplist: [, /DROPLIST_NUMBER]
[, /DROPLIST_SELECT] [, /DYNAMIC_RESIZE]
widget_label: [, /DYNAMIC_RESIZE]
widget_list: [, /LIST_MULTIPLE] [, /LIST_NUMBER]
[, /LIST_NUM_VISIBLE] [, /LIST_SELECT]
[, /LIST_TOP]
widget_slider: [, /SLIDER_MIN_MAX]
widget_table: [, /COLUMN_WIDTHS]
[, /ROW_HEIGHTS{not supported in Windows}]
[, /TABLE_ALL_EVENTS] [, /TABLE_EDITABLE]
[, /TABLE_EDIT_CELL] [, /TABLE_SELECT]
[, /TABLE_VIEW] [, /USE_TABLE_SELECT]
widget_text: [, /TEXT_ALL_EVENTS]
[, /TEXT_EDITABLE] [, /TEXT_NUMBER]
[, TEXT_OFFSET_TO_XY=*integer*] [, /TEXT_SELECT]
[, /TEXT_TOP_LINE]
[, TEXT_XY_TO_OFFSET=*[column, line]*]

WIDGET_LABEL - Creates label widgets.
Result = WIDGET_LABEL(*Parent* [, /ALIGN_CENTER |
, /ALIGN_LEFT | , /ALIGN_RIGHT]
[, /DYNAMIC_RESIZE] [, FONT=*value*]
[, FRAME=*width*] [, FUNC_GET_VALUE=*string*]
[, GROUP_LEADER=*widget_id*]
[, KILL_NOTIFY=*string*] [, /NO_COPY]
[, NOTIFY_REALIZE=*string*]
[, PRO_SET_VALUE=*string*]
[, RESOURCE_NAME=*string*] [, SCR_XSIZE=*width*]
[, SCR_YSIZE=*height*] [, /SENSITIVE]
[, /TRACKING_EVENTS] [, UNAME=*string*]
[, UNITS={0 | 1 | 2}] [, UVALUE=*value*]
[, VALUE=*value*] [, XOFFSET=*value*] [, XSIZE=*value*]
[, YOFFSET=*value*] [, YSIZE=*value*])

WIDGET_LIST - Creates list widgets.
Result = WIDGET_LIST(*Parent*
[, EVENT_FUNC=*string*] [, EVENT_PRO=*string*]
[, FONT=*value*] [, FRAME=*width*]
[, FUNC_GET_VALUE=*string*]
[, GROUP_LEADER=*widget_id*]
[, KILL_NOTIFY=*string*] [, /MULTIPLE] [, /NO_COPY]

[, NOTIFY_REALIZE=*string*]
[, PRO_SET_VALUE=*string*]
[, RESOURCE_NAME=*string*] [, SCR_XSIZE=*width*]
[, SCR_YSIZE=*height*] [, /SENSITIVE]
[, /TRACKING_EVENTS] [, UNAME=*string*]
[, UNITS={0 | 1 | 2}] [, UVALUE=*value*]
[, VALUE=*value*] [, XOFFSET=*value*] [, XSIZE=*value*]
[, YOFFSET=*value*] [, YSIZE=*value*])

WIDGET_SLIDER - Creates slider widgets.

Result = WIDGET_SLIDER(*Parent* [, /DRAG]
[, EVENT_FUNC=*string*] [, EVENT_PRO=*string*]
[, FONT=*value*] [, FRAME=*width*]
[, FUNC_GET_VALUE=*string*]
[, GROUP_LEADER=*widget_id*]
[, KILL_NOTIFY=*string*] [, MAXIMUM=*value*]
[, MINIMUM=*value*] [, /NO_COPY]
[, NOTIFY_REALIZE=*string*]
[, PRO_SET_VALUE=*string*]
[, RESOURCE_NAME=*string*] [, SCR_XSIZE=*width*]
[, SCR_YSIZE=*height*] [, SCROLL=*units*]
[, /SENSITIVE] [, /SUPPRESS_VALUE]
[, /TRACKING_EVENTS] [, TITLE=*string*]
[, UNAME=*string*] [, UNITS={0 | 1 | 2}]
[, UVALUE=*value*] [, VALUE=*value*] [, /VERTICAL]
[, XOFFSET=*value*] [, XSIZE=*value*] [, YOFFSET=*value*]
[, YSIZE=*value*])

WIDGET_TABLE - Creates table widgets.

Result = WIDGET_TABLE(*Parent* [, ALIGNMENT={0 | 1 | 2}] [, /ALL_EVENTS] [, AM_PM=*[string, string]*]
[, COLUMN_LABELS=*string_array*]
[, /COLUMN_MAJOR | , /ROW_MAJOR]
[, COLUMN_WIDTHS=*array*]
[, DAYS_OF_WEEK=*string_array*{7 names}]
[, /EDITABLE] [, EVENT_FUNC=*string*]
[, EVENT_PRO=*string*] [, FONT=*value*]
[, FORMAT=*value*] [, FRAME=*width*]
[, FUNC_GET_VALUE=*string*]
[, GROUP_LEADER=*widget_id*]
[, /KBRD_FOCUS_EVENTS] [, KILL_NOTIFY=*string*]
[, MONTHS=*string_array*{12 names}] [, /NO_COPY]
[, /NO_HEADERS] [, NOTIFY_REALIZE=*string*]
[, PRO_SET_VALUE=*string*]
[, /RESIZEABLE_COLUMNS]
[, /RESIZEABLE_ROWS{not supported in Windows}]
[, RESOURCE_NAME=*string*]
[, ROW_HEIGHTS=*array*{not supported in Windows}]
[, ROW_LABELS=*string_array*] [, SCR_XSIZE=*width*]
[, SCR_YSIZE=*height*] [, /SCROLL] [, /SENSITIVE]
[, /TRACKING_EVENTS] [, UNAME=*string*]
[, UNITS={0 | 1 | 2}] [, UVALUE=*value*]
[, VALUE=*value*] [, XOFFSET=*value*] [, XSIZE=*value*]
[, X_SCROLL_SIZE=*width*] [, YOFFSET=*value*]
[, YSIZE=*value*] [, Y_SCROLL_SIZE=*height*])

WIDGET_TEXT - Creates text widgets.

Result = WIDGET_TEXT(*Parent* [, /ALL_EVENTS]
[, /EDITABLE] [, EVENT_FUNC=*string*]
[, EVENT_PRO=*string*] [, FONT=*value*]
[, FRAME=*width*] [, FUNC_GET_VALUE=*string*]
[, GROUP_LEADER=*widget_id*]
[, /KBRD_FOCUS_EVENTS] [, KILL_NOTIFY=*string*]
[, /NO_COPY] [, /NO_NEWLINE]
[, NOTIFY_REALIZE=*string*]
[, PRO_SET_VALUE=*string*]
[, RESOURCE_NAME=*string*] [, SCR_XSIZE=*width*]
[, SCR_YSIZE=*height*] [, /SCROLL] [, /SENSITIVE]
[, /TRACKING_EVENTS] [, UNAME=*string*]
[, UNITS={0 | 1 | 2}] [, UVALUE=*value*]
[, VALUE=*value*] [, /WRAP] [, XOFFSET=*value*]
[, XSIZE=*value*] [, YOFFSET=*value*] [, YSIZE=*value*])

WINDOW - Creates window for the display of graphics or text.

WINDOW [, *Window_Index*] [, COLORS=*value*]
[, /FREE] [, /PIXMAP] [, RETAIN={0 | 1 | 2}]
[, TITLE=*string*] [, XPOS=*value*] [, YPOS=*value*]
[, XSIZE=*pixels*] [, YSIZE=*pixels*]

WRITE_BMP - Writes Microsoft Windows Version 3 device independent bitmap file (.BMP).

WRITE_BMP, *Filename*, *Image*[, *R*, *G*, *B*] [, /FOUR_BIT]
[, IHDR=*structure*] [, HEADER_DEFINE=*h*{define *h* before call}]

WRITE_GIF - Writes a Graphics Interchange Format (GIF) file.

WRITE_GIF, *Filename*, *Image*[, *R*, *G*, *B*] [, /MULTIPLE
[, /CLOSE]]

WRITE_IMAGE - Writes an image and its color table vectors, if any, to a file of a specified type.

WRITE_IMAGE, *Filename*, *Format*, *Data* [, *Red*, *Green*, *Blue*] [, /APPEND]

WRITE_JPEG - Writes JPEG file.

WRITE_JPEG[, *Filename*], *Image* [, /ORDER]
[, /PROGRESSIVE] [, QUALITY=*value*{0 to 100}]
[, TRUE={1 | 2 | 3}] [, UNIT=*lun*]

WRITE_NRIF - Writes NCAR Raster Interchange Format rasterfile.

WRITE_NRIF, *File*, *Image* [, *R*, *G*, *B*]

WRITE_PICT - Writes Macintosh PICT (version 2) bitmap file.

WRITE_PICT, *Filename* [, *Image*, *R*, *G*, *B*]

WRITE_PNG - Writes Portable Network Graphics (PNG) file.

WRITE_PNG, *Filename*, *Image*[, *R*, *G*, *B*] [, /VERBOSE]
[, TRANSPARENT=*array*]

WRITE_PPM - Writes PPM (true-color) or PGM (gray scale) file.

WRITE_PPM, *Filename*, *Image* [, /ASCII]

WRITE_SPR - Writes row-indexed sparse array structure to a file.

WRITE_SPR, *AS*, *Filename*

WRITE_SRF - Writes Sun Raster File (SRF).
WRITE_SRF, *Filename* [, *Image*, *R*, *G*, *B*] [, /ORDER] [, /WRITE_32]

WRITE_SYLK - Writes SYLK (Symbolic Link) spreadsheet file.
Result = WRITE_SYLK(*File*, *Data* [, STARTCOL=*column*] [, STARTROW=*row*])

WRITE_TIFF - Writes TIFF file with 1 to 3 channels.
WRITE_TIFF, *Filename* [, *Image*, *Order*] [, /APPEND] [, RED=*value*] [, GREEN=*value*] [, BLUE=*value*] [, COMPRESSION={0 | 1 | 2}] [, GEOTIFF=*structure*] [, /LONG | , /SHORT | ,/FLOAT] [, PLANARCONFIG={1 | 2}] [, /VERBOSE] [, XRESOL=*pixels/inch*] [, YRESOL=*pixels/inch*]

WRITE_WAV - Writes the audio stream to the named .WAV file.
WRITE_WAV, *Filename* , *Data* [, *Rate*]

WRITE_WAVE - Writes Wavefront Advanced Visualizer (.WAV) file.
WRITE_WAVE, *File*, *Array* [, /BIN] [, DATANAME=*string*] [, MESHNAME=*string*] [, /NOMESHDEF] [, /VECTOR]

WRITEU - Writes unformatted binary data to a file.
WRITEU, *Unit*, *Expr*$_1$ ..., *Expr*$_n$
UNIX Keywords: [, TRANSFER_COUNT=*variable*]
VMS Keywords: [, /REWRITE]

WSET - Selects the current window.
WSET [, *Window_Index*]

WSHOW - Exposes or hides the designated window.
WSHOW [, *Window_Index* [, *Show*]] [, /ICONIC]

WTN - Returns wavelet transform of the input array.
Result = WTN(*A*, *Coef* [, /COLUMN] [, /DOUBLE] [, /INVERSE] [, /OVERWRITE])

X

XBM_EDIT - Creates, edits bitmap icons for IDL widget button labels.
XBM_EDIT [, /BLOCK] [, FILENAME=*string*] [, GROUP=*widget_id*] [, XSIZE=*pixels*] [, YSIZE=*pixels*]

XDISPLAYFILE - Displays ASCII text file in scrolling text widget.
XDISPLAYFILE, *Filename* [, /BLOCK] [, FONT=*string*] [, GROUP=*widget_id*] [, HEIGHT=*lines*] [, /MODAL] [, TEXT=*string or string array*] [, TITLE=*string*] [, WIDTH=*characters*]

XFONT - Creates modal widget to select and view an X Windows font.
Result = XFONT([, GROUP=*widget_id*] [, /PRESERVE])

XINTERANIMATE - Displays animated sequence of images.
XINTERANIMATE [, *Rate*]
Initialization Keywords: [, SET=*[sizex, sizey, nframes]*] [, /CYCLE] [, /MPEG_OPEN, MPEG_FILENAME=*string*] [, /NO_BLOCK] [, /SHOWLOAD] [, /TRACK] [, TITLE=*string*]
Loading Keywords: [, FRAME=*value*{0 to (*nframes*-1)}[, IMAGE=*value*]] [, /ORDER] [, WINDOW=*[window_num* [, *x0*, *y0*, *sx*, *sy*]*]*]
Running Keywords: [, /CLOSE] [, GROUP=*widget_id*] [, /KEEP_PIXMAPS] [, /MPEG_CLOSE] [, XOFFSET=*pixels*] [, YOFFSET=*pixels*]

XLOADCT - Provides GUI to interactively select and load color tables.
XLOADCT [, /BLOCK] [, BOTTOM=*value*] [, /FILE] [, GROUP=*widget_id*] [, /MODAL] [, NCOLORS=*value*] [, /SILENT] [, UPDATECALLBACK='*procedure_name*' [, UPDATECBDATA=*value*]] [, /USE_CURENT]

XMANAGER - Provides event loop manager for IDL widgets.
XMANAGER [, *Name*, *ID*] [, /CATCH] [, CLEANUP=*string*] [, EVENT_HANDLER=*procedure*] [, GROUP_LEADER=*widget_id*] [, /JUST_REG] [, /NO_BLOCK]

XMNG_TMPL - Template for creating widgets.
XMNG_TMPL [, /BLOCK] [, GROUP=*widget_id*]

XMTOOL - Displays tool for viewing XMANAGER widgets.
XMTOOL [, /BLOCK] [, GROUP=*widget_id*]

XOBJVIEW - Displays object viewer widget.
XOBJVIEW, *Obj* [, /BLOCK] [, GROUP=*widget_id*] [, STATIONARY=*objref(s)*] [, XSIZE=*pixels*] [, YSIZE=*pixels*]

XPALETTE - Displays widget used to create and modify color tables.
XPALETTE [, /BLOCK] [, GROUP=*widget_id*] [, UPDATECALLBACK='*procedure_name*' [, UPDATECBDATA=*value*]]

XREGISTERED - Returns registration status of a given widget.
Result = XREGISTERED(*Name* [, /NO_SHOW])

XSQ_TEST - Computes Chi-square goodness-of-fit test.
Result = XSQ_TEST(*Obfreq*, *Exfreq* [, EXCELL=*variable*] [, OBCELL=*variable*] [, RESIDUAL=*variable*])

XSURFACE - Provides GUI to SURFACE and SHADE_SURF.
XSURFACE, *Data* [, /BLOCK] [, GROUP=*widget_id*]

XVAREDIT - Provides widget-based editor for IDL variables.
XVAREDIT, *Var* [, NAME='*variable_name*'{ignored if variable is a structure}] [, GROUP=*widget_id*] [,X_SCROLL_SIZE=*columns*][,Y_SCROLL_SIZE=*rows*]

XYOUTS - Draws text on currently-selected graphics device.
XYOUTS, [*X*, *Y*,] *String* [, ALIGNMENT=*value*{0.0 to 1.0}] [, CHARSIZE=*value*] [, CHARTHICK=*value*] [, TEXT_AXES={0 | 1 | 2 | 3 | 4 | 5}] [, WIDTH=*variable*] [, CLIP=*[X*$_0$*, Y*$_0$*, X*$_1$*, Y*$_1$*]* | , /NOCLIP] [, COLOR=*index*] [, /DATA | , /DEVICE | , /NORMAL] [, ORIENTATION=*ccw_degrees_from_horiz*] [, /T3D | Z=*value*]

Z

ZOOM - Zooms portions of the display.

ZOOM [, /CONTINUOUS] [, FACT=*integer*] [, /INTERP] [, /KEEP] [, /NEW_WINDOW] [, XSIZE=*value*] [, YSIZE=*value*] [, ZOOM_WINDOW=*variable*]

ZOOM_24 - Zooms portions of true-color (24-bit) display.

ZOOM_24 [, FACT=*integer*] [, /RIGHT] [, XSIZE=*value*] [, YSIZE=*value*]

Scientific Data Formats

CDF Routines

CDF_ATTCREATE - Creates a new attribute.
Result = CDF_ATTCREATE(*Id*, *Attribute_Name*
[, /GLOBAL_SCOPE] [, /VARIABLE_SCOPE])

CDF_ATTDELETE - Deletes attribute from specified CDF file.
CDF_ATTDELETE, *Id*, *Attribute* [, *EntryNum*]
[, /ZVARIABLE]

CDF_ATTEXISTS - Determines whether specified attribute exists.
Result = CDF_ATTEXISTS(*Id*, *Attribute* [, *EntryNum*]
[, /ZVARIABLE])

CDF_ATTGET - Reads an attribute entry from a CDF file.
CDF_ATTGET, *Id*, *Attribute*, *EntryNum*, *Value*
[, CDF_TYPE= *variable*] [, /ZVARIABLE]

CDF_ATTINQ - Obtains information about specified attribute.
CDF_ATTINQ, *Id*, *Attribute*, *Name*, *Scope*, *MaxEntry*
[, *MaxZEntry*]

CDF_ATTNUM - Returns an attribute number.
Result = CDF_ATTNUM(*Id*, *Attribute_Name*)

CDF_ATTPUT - Writes an attribute entry to a CDF file.
CDF_ATTPUT, *Id*, *Attribute*, *EntryNum*, *Value*
[, /ZVARIABLE]

CDF_ATTRENAME - Renames an existing attribute.
CDF_ATTRENAME, *Id*, *OldAttr*, *NewName*

CDF_CLOSE - Closes specified Common Data Format file.
CDF_CLOSE, *Id*

CDF_COMPRESSION - Sets or returns the compression mode for a CDF file and/or variables.
CDF_COMPRESSION, *Id*
[, GET_COMPRESSION=*variable*]
[, GET_GZIP_LEVEL=*variable*]
[, GET_VAR_COMPRESSION=*variable*]
[, GET_VAR_GZIP_LEVEL=*variable*]
[, SET_COMPRESSION = {0 | 1 | 2 | 3 | 5}]
[, SET_GZIP_LEVEL=*integer*{1 to 9}]
[, SET_VAR_COMPRESSION = {0 | 1 | 2 | 3 | 5}]
[, SET_VAR_GZIP_LEVEL=*integer*{1 to 9}]
[, VARIABLE=*variable name or index*] [, /ZVARIABLE]

CDF_CONTROL - Obtains or sets information for a CDF file.
CDF_CONTROL, *Id* [, ATTRIBUTE=*name or number*]
[, GET_ATTR_INFO=*variable*]
[, GET_CACHESIZE=*variable*]
[, GET_COPYRIGHT=*variable*]
[, GET_FILENAME=*variable*]
[, GET_FORMAT=*variable*]
[, GET_NEGTOPOSFP0_MODE=*variable*]
[, GET_NUMATTRS=*variable*]
[, GET_READONLY_MODE=*variable*]
[, GET_RVAR_CACHESIZE=*variable*]
[, GET_VAR_INFO=*variable*] [, GET_ZMODE=*variable*]
[, GET_ZVAR_CACHESIZE=*variable*]
[, SET_CACHESIZE=*value*]
[, SET_EXTENDRECS=*records*]
[, SET_INITIALRECS=*records*]
[, /SET_NEGTOPOSFP0_MODE]
[, SET_PADVALUE=*value*]
[, /SET_READONLY_MODE]
[, SET_RVAR_CACHESIZE=*value*{See Note}]
[, SET_RVARS_CACHESIZE=*value*{See Note}]
[, SET_ZMODE={0 | 1 | 2}]
[, SET_ZVAR_CACHESIZE=*value*{See Note}]
[, SET_ZVARS_CACHESIZE=*value*{See Note}]
[, VARIABLE=*name or index*] [, /ZVARIABLE]
Note: Use only with MULTI_FILE CDF files

CDF_CREATE - Creates a new Common Data Format file.
Result = CDF_CREATE(*Filename*, [*Dimensions*]
[, /CLOBBER] [, /MULTI_FILE | , /SINGLE_FILE]
[, /COL_MAJOR | , /ROW_MAJOR])
Encoding Keywords (pick one):
[, /ALPHAOSF1_ENCODING]
[, /ALPHAVMSD_ENCODING]
[, /ALPHAVMSG_ENCODING]
[, /DECSTATION_ENCODING]
[, /HOST_ENCODING]
[, /HP_ENCODING]
[, /IBMRS_ENCODING]
[, /IBMPC_ENCODING]
[, /MAC_ENCODING]
[, /NETWORK_ENCODING]
[, /NEXT_ENCODING]
[, /SGI_ENCODING]
[, /SUN_ENCODING]
Decoding Keywords (pick one):
[, /ALPHAOSF1_DECODING]
[, /ALPHAVMSD_DECODING]
[, /ALPHAVMSG_DECODING]
[, /DECSTATION_DECODING]
[, /HOST_DECODING]
[, /HP_DECODING]
[, /IBMRS_DECODING]
[, /IBMPC_DECODING]
[, /MAC_DECODING]
[, /NETWORK_DECODING]

[, /NEXT_DECODING]
[, /SGI_DECODING]
[, /SUN_DECODING]

CDF_DELETE - Deletes specified Common Data Format file.
CDF_DELETE, *Id*

CDF_DOC - Gets documentation information about a CDF file.
CDF_DOC, *Id*, *Version*, *Release*, *Copyright*
[, INCREMENT=*variable*]

CDF_ENCODE_EPOCH - Encodes CDF_EPOCH variable into a string.
Result = CDF_ENCODE_EPOCH(*Epoch* [, EPOCH={0 | 1 | 2 | 3}])

CDF_EPOCH - Computes/breaks down CDF_EPOCH values.
CDF_EPOCH, *Epoch*, *Year* [, *Month*, *Day*, *Hour*, *Minute*, *Second*, *Milli*] [, /BREAKDOWN_EPOCH]
[, /COMPUTE_EPOCH]

CDF_ERROR - Returns explanation of a given status code.
Result = CDF_ERROR(*Status*)

CDF_EXISTS - Returns True if CDF data format library is supported on the current IDL platform.
Result = CDF_EXISTS( )

CDF_INQUIRE - Returns global information about CDF file.
Result = CDF_INQUIRE(*Id*)

CDF_LIB_INFO - Returns information about the CDF Library being used.
CDF_LIB_INFO [, COPYRIGHT =*variable*]
[, INCREMENT=*variable*] [, RELEASE=*variable*]
[, SUBINCREMENT=*variable*] [, VERSION=*variable*]

CDF_OPEN - Opens an existing Common Data Format file.
Result = CDF_OPEN(*Filename*)

CDF_PARSE_EPOCH - Parses input string into a double precision value properly formatted for use as CDF_EPOCH variable.
Result = CDF_PARSE_EPOCH(*Epoch_string*)

CDF_VARCREATE - Creates new variable in CDF file.
Result = CDF_VARCREATE(*Id*, *Name* [, *DimVary*]
[, /CDF_BYTE | , /CDF_CHAR | , /CDF_DOUBLE | , /CDF_EPOCH | , /CDF_FLOAT | , /CDF_INT1 | , /CDF_INT2 | , /CDF_INT4 | , /CDF_REAL4 | , /CDF_REAL8 | , /CDF_UCHAR | , /CDF_UINT1 | , /CDF_UINT2 | , /CDF_UINT4]
[, ALLOCATERECS=*records*] [, DIMENSIONS=*array*]
[, NUMELEM=*characters*] [, /REC_NOVARY | , /REC_VARY] [, /ZVARIABLE])

CDF_VARDELETE - Deletes variable from a SINGLE_FILE CDF file.
CDF_VARDELETE, *Id*, *Variable* [, /ZVARIABLE]

CDF_VARGET - Reads multiple values from CDF file variable.
CDF_VARGET, *Id*, *Variable*, *Value* [, COUNT=*vector*]
[, INTERVAL=*vector*] [, OFFSET=*vector*]
[, REC_COUNT=*records*] [, REC_INTERVAL=*value*]
[, REC_START=*record*] [, /STRING{data in CDF file must be type CDF_CHAR or CDF_UCHAR}]
[, /ZVARIABLE]

CDF_VARGET1 - Reads one value from a CDF file variable.
CDF_VARGET1, *Id*, *Variable*, *Value* [, OFFSET=*vector*]
[, REC_START=*record*] [, /STRING{data in CDF file must be type CDF_CHAR or CDF_UCHAR}]
[, /ZVARIABLE]

CDF_VARINQ - Returns structure containing information about specified variable.
Result = CDF_VARINQ(*Id*, *Variable* [, /ZVARIABLE])

CDF_VARNUM - Returns variable number associated with given variable name.
Result = CDF_VARNUM(*Id*, *VarName* [, *IsZVar*])

CDF_VARPUT - Writes value to a variable.
CDF_VARPUT, *Id*, *Variable*, *Value* [, COUNT=*vector*]
[, INTERVAL=*vector*] [, OFFSET=*vector*]
[, REC_INTERVAL=*value*] [, REC_START=*record*]
[, /ZVARIABLE]

CDF_VARRENAME - Renames existing variable.
CDF_VARRENAME, *Id*, *OldVariable*, *NewName*
[, /ZVARIABLE]

EOS Routines

EOS_EH_CONVANG - Converts angles between decimal degrees, radians, and packed degrees-minutes-seconds.
Result = EOS_EH_CONVANG(*inAngle*, *code*)

EOS_EH_GETVERSION - Retrieves the HDF-EOS version string of an HDF-EOS file.
Result = EOS_EH_GETVERSION(*fid*, *version*)

EOS_EH_IDINFO - Returns the HDF file IDs corresponding to the HDF-EOS file ID returned by EOS_SW_OPEN, EOS_GD_OPEN, or EOS_PT_OPEN.
Result = EOS_EH_IDINFO(*fid*, *HDFfid*, *sdInterfaceID*)

EOS_EXISTS - Returns True if HDF EOS format library is supported on the current IDL platform.
Result = EOS_EXISTS( )

EOS_GD_ATTACH - Attaches to the grid using the gridname parameter as the identifier.
Result = EOS_GD_ATTACH(*fid*, *gridname*)

EOS_GD_ATTRINFO - Returns number type and number of elements (count) of a grid attribute.
Result = EOS_GD_ATTRINFO(*gridID*, *attrname*, *numbertype*, *count*)

EOS_GD_CLOSE - Closes the HDF grid file.
Result = EOS_GD_CLOSE(*fid*)

EOS_GD_COMPINFO - Returns the compression code and compression parameters for a given field.
Result = EOS_GD_COMPINFO(*gridID*, *fieldname*, *compcode*, *compparm*)

EOS_GD_CREATE - Creates a grid within the file.
Result = EOS_GD_CREATE(*fid*, *gridname*, *xdimsize*, *ydimsize*, *upleftpt*, *lowrightpt*)

EOS_GD_DEFBOXREGION - Defines a longitude-latitude box region for a grid.
Result = EOS_GD_DEFBOXREGION(*gridID*, *cornerlon*, *cornerlat*)

EOS_GD_DEFCOMP - Sets the HDF field compression for subsequent grid field definitions.
Result = EOS_GD_DEFCOMP(*gridID*, *compcode* [, *compparm*])

EOS_GD_DEFDIM - Defines dimensions used by field definition routines to establish size of the field.
Result = EOS_GD_DEFDIM(*gridID*, *dimname*, *dim*)

EOS_GD_DEFFIELD - Defines data fields to be stored in the grid.
Result = EOS_GD_DEFFIELD(*gridID*, *fieldname*, *dimlist*, *numbertype* [, /MERGE])

EOS_GD_DEFORIGIN - Defines the origin of the grid data.
Result = EOS_GD_DEFORIGIN(*gridID*, *origincode*)

EOS_GD_DEFPIXREG - Defines whether the pixel center or pixel corner is used when requesting the location of a given pixel.
Result = EOS_GD_DEFPIXREG(*gridID*, *pixreg*)

EOS_GD_DEFPROJ - Defines the GCTP projection and projection parameters of the grid.
Result = EOS_GD_DEFPROJ(*gridID*, *projcode*, *zonecode*, *spherecode*, *projparm*)

EOS_GD_DEFTILE - Defines the tiling dimensions for fields defined following this function call.
Result = EOS_GD_DEFTILE(*gridID*, *tilecode* [, *tilerank*, *tiledims*])

EOS_GD_DEFTIMEPERIOD - Defines a time period for a grid.
Result = EOS_GD_DEFTIMEPERIOD(*gridID*, *periodID*, *starttime*, *stoptime*)

EOS_GD_DEFVRTREGION - Subsets on a monotonic field or contiguous elements of a dimension.
Result = EOS_GD_DEFVRTREGION(*gridID*, *regionID*, *vertObj*, *range*)

EOS_GD_DETACH - Detaches from grid interface.
Result = EOS_GD_DETACH(*gridID*)

EOS_GD_DIMINFO - Retrieves the size of the specified dimension.
Result = EOS_GD_DIMINFO(*gridID*, *dimname*)

EOS_GD_DUPREGION - Copies information stored in current region or period to a new region or period.
Result = EOS_GD_DUPREGION(*regionID*)

EOS_GD_EXTRACTREGION - Reads data into the data buffer from a subsetted region as defined by EOS_GD_DEFBOXREGION.
Result = EOS_GD_EXTRACTREGION(*gridID*, *regionID*, *fieldname*, *buffer*)

EOS_GD_FIELDINFO - Retrieves information on a specific data field.
Result = EOS_GD_FIELDINFO(*gridID*, *fieldname*, *rank*, *dims*, *numbertype*, *dimlist*)

EOS_GD_GETFILLVALUE - Retrieves fill value for specified field.
Result = EOS_GD_GETFILLVALUE(*gridID*, *fieldname*, *fillvalue*)

EOS_GD_GETPIXELS - Returns the pixel rows and columns for specified longitude/latitude pairs.
Result = EOS_GD_GETPIXELS(*gridID*, *nLonLat*, *lonVal*, *latVal*, *pixRow*, *pixCol*)

EOS_GD_GETPIXVALUES - Reads data from a data field for the specified pixels.
Result = EOS_GD_GETPIXVALUES(*gridID*, *nPixels*, *pixRow*, *pixCol*, *fieldname*, *buffer*)

EOS_GD_GRIDINFO - Returns number of rows, columns, and the location of the upper left and lower right corners of the grid image.
Result = EOS_GD_GRIDINFO(*gridID*, *xdimsize*, *ydimsize*, *upleft*, *lowright*)

EOS_GD_INQATTRS - Retrieves information about attributes defined in grid.
Result = EOS_GD_INQATTRS(*gridID*, *attrlist* [, LENGTH (OUT)=*value*])

EOS_GD_INQDIMS - Retrieves information about dimensions defined in grid.
Result = EOS_GD_INQDIMS(*gridID*, *dimname*, *dims*)

EOS_GD_INQFIELDS - Retrieves information about the data fields defined in grid.
Result = EOS_GD_INQFIELDS(*gridID*, *fieldlist*, *rank*, *numbertype*)

EOS_GD_INQGRID - Retrieves number and names of grids defined in HDF-EOS file.
Result = EOS_GD_INQGRID(*filename*, *gridlist* [, LENGTH (OUT)=*value*])

EOS_GD_INTERPOLATE - Performs bilinear interpolation on a grid field.
Result = EOS_GD_INTERPOLATE(*gridID*, *Interp*, *lonVal*, *latVal*, *fieldname*, *interpVal*)

EOS_GD_NENTRIES - Returns number of entries and descriptive string buffer size for a specified entity.
Result = EOS_GD_NENTRIES(*gridID*, *entrycode* [, LENGTH (OUT)=*value*])

EOS_GD_OPEN - Opens an existing file or creates a new file.
Result = EOS_GD_OPEN(*filename*, *access* [, /CREATE] [, /RDWR | , /READ])

EOS_GD_ORIGININFO - Retrieves origin code.
Result = EOS_GD_ORIGININFO(*gridID*, *origincode*)

EOS_GD_PIXREGINFO - Retrieves the pixel registration code.
Result = EOS_GD_PIXREGINFO(*gridID*, *pixregcode*)

EOS_GD_PROJINFO - Retrieves GCTP projection code, zone code, spheroid code, and projection parameters of the grid.
Result = EOS_GD_PROJINFO(*gridID*, *projcode*, *zonecode*, *spherecode*, *projparm*)

EOS_GD_QUERY - Returns information about a specified grid.
Result = EOS_GD_QUERY(*Filename*, *GridName*, [*Info*])

EOS_GD_READATTR - Reads attribute from a grid.
Result = EOS_GD_READATTR(*gridID*, *attrname*, *datbuf*)

EOS_GD_READFIELD - Reads data from a grid field.
Result = EOS_GD_READFIELD(*gridID*, *fieldname*, *buffer* [, EDGE=*array*] [, START=*array*] [, STRIDE=*array*])

EOS_GD_READTILE - Reads from tile within field.
Result = EOS_GD_READTILE(*gridID*, *fieldname*, *tilecoords*, *buffer*)

EOS_GD_REGIONINFO - Returns information about a subsetted region for a particular field.
Result = EOS_GD_REGIONINFO(*gridID*, *regionID*, *fieldname*, *ntype*, *rank*, *dims*, *size*, *upleftpt*, *lowrightpt*)

EOS_GD_SETFILLVALUE - Sets fill value for the specified field.
Result = EOS_GD_SETFILLVALUE(*gridID*, *fieldname*, *fillvalue*)

EOS_GD_SETTILECACHE - Sets tile cache parameters.
Result = EOS_GD_SETTILECACHE(*gridID*, *fieldname*, *maxcache*, *cachecode*)

EOS_GD_TILEINFO - Returns tiling code, tiling rank, and tiling dimensions for a given field.
Result = EOS_GD_TILEINFO(*gridID*, *fieldname*, *tilecode*, *tilerank*, *tiledims*)

EOS_GD_WRITEATTR - Writes/updates attribute in a grid.
Result = EOS_GD_WRITEATTR(*gridID*, *attrname*, *datbuf* [, COUNT=*value*] [, HDF_TYPE=*value*])

EOS_GD_WRITEFIELD - Writes data to a grid field.
Result = EOS_GD_WRITEFIELD(*gridID*, *fieldname*, *data* [, EDGE=*array*] [, START=*array*] [, STRIDE=*array*])

EOS_GD_WRITEFIELDMETA - Writes field metadata for a grid field not defined by the Grid API.
Result = EOS_GD_WRITEFIELDMETA(*gridID*, *fieldname*, *dimlist*, *numbertype*)

EOS_GD_WRITETILE - Writes a single tile of data to a field.
Result = EOS_GD_WRITETILE(*gridID*, *fieldname*, *tilecoords*, *data*)

EOS_PT_ATTACH - Attaches to point using the pointname parameter as the identifier.
Result = EOS_PT_ATTACH(*fid*, *pointname*)

EOS_PT_ATTRINFO - Returns number type and number of elements of a point attribute.
Result = EOS_PT_ATTRINFO(*pointID*, *attrname*, *numbertype*, *count*)

EOS_PT_BCKLINKINFO - Returns linkfield to the previous level.
Result = EOS_PT_BCKLINKINFO(*pointID*, *level*, *linkfield*)

EOS_PT_CLOSE - Closes the HDF point file.
Result = EOS_PT_CLOSE(*fid*)

EOS_PT_CREATE - Creates point as a Vgroup within the HDF file.
Result = EOS_PT_CREATE(*fid*, *pointname*)

EOS_PT_DEFBOXREGION - Defines area of interest for a point.
Result = EOS_PT_DEFBOXREGION(*pointID*, *cornerlon*, *cornerlat*)

EOS_PT_DEFLEVEL - Defines a level within a point.
Result = EOS_PT_DEFLEVEL(*pointID*, *levelname*, *fieldlist*, *fieldtype*, *fieldorder*)

EOS_PT_DEFLINKAGE - Defines the linkfield between two levels.
Result = EOS_PT_DEFLINKAGE(*pointID*, *parent*, *child*, *linkfield*)

EOS_PT_DEFTIMEPERIOD - Defines a time period for a point.
Result = EOS_PT_DEFTIMEPERIOD(*pointID*, *starttime*, *stoptime*)

EOS_PT_DEFVRTREGION - Selects records within a point whose field values are within a given range.
Result = EOS_PT_DEFVRTREGION(*pointID*, *regionID*, *vertObj*, *range*)

EOS_PT_DETACH - Detaches from a point data set.
Result = EOS_PT_DETACH(*pointID*)

EOS_PT_EXTRACTPERIOD - Reads data from the designated level fields into the data buffer from the subsetted time period.
Result = EOS_PT_EXTRACTPERIOD(*pointID*, *periodID*, *level*, *fieldlist*, *buffer*)

EOS_PT_EXTRACTREGION - Reads data from the designated level fields into the data buffer from the subsetted area of interest.
Result = EOS_PT_EXTRACTREGION(*pointID*, *regionID*, *level*, *fieldlist*, *buffer*)

EOS_PT_FWDLINKINFO - Returns the linkfield to the given level.
Result = EOS_PT_FWDLINKINFO(*pointID*, *level*, *linkfield*)

EOS_PT_GETLEVELNAME - Returns the name of a level given the level number (0-based).
Result = EOS_PT_GETLEVELNAME(*pointID*, *level*, *levelname* [, LENGTH (OUT)=*variable*])

EOS_PT_GETRECNUMS - Returns record numbers in one level that are connected to a given set of records in a different level.
Result = EOS_PT_GETRECNUMS(*pointID*, *inlevel*, *outlevel*, *inNrec*, *inRecs*, *outNrec*, *outRecs*)

EOS_PT_INQATTRS - Returns attribute list as a comma-separated string.
Result = EOS_PT_INQATTRS(*pointID*, *attrlist* [, LENGTH=*value*])

EOS_PT_INQPOINT - Retrieves number and names of points defined in HDF-EOS file.
Result = EOS_PT_INQPOINT*(filename*, *pointlist* [, LENGTH (OUT)=*value*])

EOS_PT_LEVELINDX - Returns the level index for a given level.
Result = EOS_PT_LEVELINDX(*pointID*, *levelname*)

EOS_PT_LEVELINFO - Returns information about the fields in a given level.
Result = EOS_PT_LEVELINFO(*pointID*, *level*, *fieldlist*, *fldtype*, *fldorder*)

EOS_PT_NFIELDS - Returns the number of fields in a level.
Result = EOS_PT_NFIELDS(*pointID*, *level* [, LENGTH=*bytes*])

EOS_PT_NLEVELS - Returns the number of levels in a point.
Result = EOS_PT_NLEVELS(*pointID*)

EOS_PT_NRECS - Returns the number of records in a given level.
Result = EOS_PT_NRECS(*pointID*, *level*)

EOS_PT_OPEN - Creates a new file or opens an existing one.
Result = EOS_PT_OPEN(*fieldname* [, /CREATE] [, /RDWR | , /READ])

EOS_PT_PERIODINFO - Returns information about a subsetted time period for a given fieldlist.
Result = EOS_PT_PERIODINFO(*pointID*, *periodID*, *level*, *fieldlist*, *size*)

EOS_PT_PERIODRECS - Returns record numbers within a subsetted time period for a given level.
Result = EOS_PT_PERIODRECS(*pointID*, *periodID*, *level*, *nrec*, *recs*)

EOS_PT_QUERY - Returns information about a specified point.
Result = EOS_PT_QUERY(*Filename*, *PointName*, [*Info*])

EOS_PT_READATTR - Reads attributes.
Result = EOS_PT_READATTR(*pointID*, *attrname*, *datbuf*)

EOS_PT_READLEVEL - Reads data from the specified fields and records of a single level in a point.
Result = EOS_PT_READLEVEL(*pointID*, *level*, *fieldlist*, *nrec*, *recs*, *buffer*)

EOS_PT_REGIONINFO - Returns information about a subsetted area of interest for a given fieldlist.
Result = EOS_PT_REGIONINFO(*pointID*, *regionID*, *level*, *fieldlist*, *size*)

EOS_PT_REGIONRECS - Returns the record numbers within a subsetted geographic region for a given level.
Result = EOS_PT_REGIONRECS(*pointID*, *regionID*, *level*, *nrec*, *recs*)

EOS_PT_SIZEOF - Returns information about specified fields in a point regardless of level.
Result = EOS_PT_SIZEOF(*pointID*, *fieldlist*, *fldlevel*)

EOS_PT_UPDATELEVEL - Updates the specified fields and records of a single level.
Result = EOS_PT_UPDATELEVEL(*pointID*, *level*, *field*, *list*, *nrec*, *recs*, *data*)

EOS_PT_WRITEATTR - Writes/updates an attribute in a point.
Result = EOS_PT_WRITEATTR(*pointID*, *attrname*, *datbuf* [, COUNT=*value*] [, HDF_TYPE=*value*])

EOS_PT_WRITELEVEL - Writes (appends) full records to a level.
Result = EOS_PT_WRITELEVEL(*pointID*, *level*, *nrec*, *data*)

EOS_QUERY - Returns information about the makeup of an HDF-EOS file.
Result = EOS_QUERY(*Filename*, [*Info*])

EOS_SW_ATTACH - Attaches to the swath using the swathname parameter as the identifier.
Result = EOS_SW_ATTACH(*fid*, *swathname*)

EOS_SW_ATTRINFO - Returns number type and number of elements of a swath attribute.
Result = EOS_SW_ATTRINFO(*swathID*, *attrname*, *numbertype*, *count*)

EOS_SW_CLOSE - Closes the HDF swath file.
Result = EOS_SW_CLOSE(*fid*)

EOS_SW_COMPINFO - Returns compression code and compression parameters for a given field.
Result = EOS_SW_COMPINFO(*swathID*, *fieldname*, *compcode*, *compparm*)

EOS_SW_CREATE - Creates a swath within the file.
Result = EOS_SW_CREATE(*fid*, *swathname*)

EOS_SW_DEFBOXREGION - Defines a longitude-latitude box region for a swath.
Result = EOS_SW_DEFBOXREGION(*swathID*, *cornerlon*, *cornerlat*, *mode*)

EOS_SW_DEFCOMP - Sets HDF field compression for subsequent swath field definitions.
Result = EOS_SW_DEFCOMP(*swathID*, *compcode*, [, *compparm*])

EOS_SW_DEFDATAFIELD - Defines geolocation fields to be stored in the swath.
Result = EOS_SW_DEFDATAFIELD(*swathID*, *fieldname*, *dimlist*, *numbertype* [, /MERGE])

EOS_SW_DEFDIM - Defines dimensions that are used by the field definition routines to establish the size of the field.
Result = EOS_SW_DEFDIM(*swathID*, *fieldname*, *dim*)

EOS_SW_DEFDIMMAP - Defines monotonic mapping between the geolocation and data dimensions.
Result = EOS_SW_DEFDIMMAP(*swathID, geodim, datadim, offset, increment*)

EOS_SW_DEFGEOFIELD - Defines geolocation fields to be stored in the swath.
Result = EOS_SW_DEFGEOFIELD(*swathID, fieldname, dimlist, numbertype* [, /MERGE])

EOS_SW_DEFIDXMAP - Defines mapping between a geolocation and data dimension.
Result = EOS_SW_DEFIDXMAP(*swathID, geodim, datadim, index*)

EOS_SW_DEFTIMEPERIOD - Defines a time period for a swath.
Result = EOS_SW_DEFTIMEPERIOD(*swathID, starttime , stoptime, mode*)

EOS_SW_DEFVRTREGION - Subsets along any dimension.
Result = EOS_SW_DEFVRTREGION(*swathID, regionID, vertObj, range*)

EOS_SW_DETACH - Detaches from the swath interface.
Result = EOS_SW_DETACH(*swathID*)

EOS_SW_DIMINFO - Retrieves the size of the specified dimension.
Result = EOS_SW_DIMINFO(*swathID, dimname*)

EOS_SW_DUPREGION - Copies information stored in a current region or period to a new region or period.
Result = EOS_SW_DUPREGION(*regionID*)

EOS_SW_EXTRACTPERIOD - Reads data into the data buffer from the subsetted time period.
Result = EOS_SW_EXTRACTPERIOD(*swathID, periodID, fieldname, external_mode, buffer*)

EOS_SW_EXTRACTREGION - Reads data into the data buffer from the subsetted region.
Result = EOS_SW_EXTRACTREGION(*swathID, regionID, fieldname, external_mode, buffer*)

EOS_SW_FIELDINFO - Retrieves information on a specific data field.
Result = EOS_SW_FIELDINFO(*swathID, fieldname, rank, dims, numbertype, dimlist*)

EOS_SW_GETFILLVALUE - Retrieves fill value for specified field.
Result = EOS_SW_GETFILLVALUE(*swathID, fieldname, fillvalue*)

EOS_SW_IDXMAPINFO - Retrieves size of the indexed array and the array of indexed elements of the specified geolocation mapping.
Result = EOS_SW_IDXMAPINFO(*swathID, geodim, datadim, index*)

EOS_SW_INQATTRS - Retrieves information about attributes defined in swath.
Result = EOS_SW_INQATTRS(*swathID, attrlist* [, LENGTH (OUT)=*value*])

EOS_SW_INQDATAFIELDS - Retrieves information about all of the data fields defined in swath.
Result = EOS_SW_INQDATAFIELDS(*swathID, fieldlist, rank, numbertype*)

EOS_SW_INQDIMS - Retrieves information about all of the dimensions defined in swath.
Result = EOS_SW_INQDIMS(*swathID, dimname, dim*)

EOS_SW_INQGEOFIELDS - Retrieves information about all of the geolocation fields defined in swath.
Result = EOS_SW_INQGEOFIELDS(*swathID, fieldlist, rank, numbertype*)

EOS_SW_INQIDXMAPS - Retrieves information about all indexed geolocation/data mappings in swath.
Result = EOS_SW_INQIDXMAPS(*swathID, idxmap, idxsizes*)

EOS_SW_INQMAPS - Retrieves information about all non-indexed geolocation relations in swath.
Result = EOS_SW_INQMAPS(*swathID, dimmap, offset, increment*)

EOS_SW_INQSWATH - Retrieves number and names of swaths defined in HDF-EOS file.
Result = EOS_SW_INQSWATH(*filename, swathlist* [, LENGTH =*value*])

EOS_SW_MAPINFO - Retrieves offset and increment of the specified geolocation mapping.
Result = EOS_SW_MAPINFO(*swathID, geodim, datadim, offset, increment*)

EOS_SW_NENTRIES - Returns number of entries and descriptive string buffer size for specified entity.
Result = EOS_SW_NENTRIES(*swathID, entrycode* [, LENGTH (OUT)=*value*])

EOS_SW_OPEN - Opens an existing file, or creates a new file.
Result = EOS_SW_OPEN(*filename* [, /CREATE] [, /RDWR | , /READ])

EOS_SW_PERIODINFO - Returns information about a subsetted time period for a given field.
Result = EOS_SW_PERIODINFO(*swathID, periodID, fieldname, ntype, rank, dims, size*)

EOS_SW_QUERY - Returns information about a specified swath.
Result=EOS_SW_QUERY(*Filename, SwathName,* [*Info*])

EOS_SW_READATTR - Reads attribute from a swath field.
Result = EOS_SW_READATTR(*swathID, attrname, datbuf*)

EOS_SW_READFIELD - Reads data from a swath field.
Result = EOS_SW_READFIELD(*swathID, fieldname, buffer* [, EDGE=*array*] [, START=*array*] [, STRIDE=*array*])

EOS_SW_REGIONINFO - Returns information about a subsetted region for a given field.
Result = EOS_SW_REGIONINFO(*swathID, regionID, fieldname, ntype, rank, dims, size*)

EOS_SW_SETFILLVALUE - Sets fill value for the specified field.
Result = EOS_SW_SETFILLVALUE(*swathID, fieldname, fillvalue*)

EOS_SW_WRITEATTR - Writes/updates attribute in a swath.
Result = EOS_SW_WRITEATTR(*swathID, attrname, datbuf* [, COUNT=*value*] [, HDF_TYPE=*value*])

EOS_SW_WRITEDATAMETA - Writes field metadata for an existing data field.
Result = EOS_SW_WRITEDATAMETA(*swathID, fieldname, dimlist, numbertype*)

EOS_SW_WRITEFIELD - Writes data to a swath field.
Result = EOS_SW_WRITEFIELD(*swathID, fieldname, cut, data* [, EDGE=*array*] [, START=*array*] [, STRIDE=*array*])

EOS_SW_WRITEGEOMETA - Writes field metadata for an existing geolocation field.
Result = EOS_SW_WRITEGEOMETA(*swathID, fieldname, dimlist, numbertype*)

HDF Routines

HDF_AN_ANNLEN - Returns number of characters in annotation.
Result = HDF_AN_ANNLEN(*ann_id*)

HDF_AN_ANNLIST - Obtains a list of annotation identifiers.
Result = HDF_AN_ANNLIST(*an_id, annot_type, obj_tag, obj_ref, ann_list*)

HDF_AN_ATYPE2TAG - Returns HDF tag corresponding to given annotation type.
Result = HDF_AN_ATYPE2TAG(*annot_type*)

HDF_AN_CREATE - Creates HDF AN annotation.
Result = HDF_AN_CREATE(*an_id, obj_tag, obj_ref, annot_type*)

HDF_AN_CREATEF - Creates file annotation.
Result = HDF_AN_CREATEF(*an_id, annot_type*)

HDF_AN_END - Terminates access to the HDF AN interface.
HDF_AN_END, *an_id*

HDF_AN_ENDACCESS - Terminates access to an annotation.
HDF_AN_ENDACCESS, *ann_id*

HDF_AN_FILEINFO - Retrieves total number of annotations and stores them in the appropriate parameters.
Result = HDF_AN_FILEINFO(*an_id, n_file_labels, n_file_descs, n_data_labels, n_data_descs*)

HDF_AN_GET_TAGREF - Retrieves HDF tag and reference number of annotation.
Result = HDF_AN_GET_TAGREF(*an_id, index, annot_type, ann_tag, ann_ref*)

HDF_AN_ID2TAGREF - Retrieves HDF tag/reference number pair of annotation.
Result = HDF_AN_ID2TAGREF(*ann_id, ann_tag, ann_ref*)

HDF_AN_NUMANN - Returns total number of annotations of a given type.
Result = HDF_AN_NUMANN(*an_id, annot_type, obj_tag, obj_ref*)

HDF_AN_READANN - Reads specified annotation.
Result = HDF_AN_READANN(*ann_id, annotation* [, LENGTH=*characters*])

HDF_AN_SELECT - Obtains identifier of specified annotation.
Result = HDF_AN_SELECT(*an_id, index, annot_type*)

HDF_AN_START - Initializes interface for specified file.
Result = HDF_AN_START(*file_id*)

HDF_AN_TAG2ATYPE - Returns annotation type of corresponding HDF tag.
Result = HDF_AN_TAG2ATYPE(*ann_tag*)

HDF_AN_TAGREF2ID - Returns identifier of annotation with specified tag.
Result = HDF_AN_TAGREF2ID(*an_id, ann_tag, ann_ref*)

HDF_AN_WRITEANN - Writes annotation text.
Result = HDF_AN_WRITEANN(*ann_id, annotation* [, LENGTH=*characters*])

HDF_BROWSER - See "HDF_BROWSER" on page 16.

HDF_CLOSE - Closes HDF file associated with the given file handle.
HDF_CLOSE, *FileHandle*

HDF_DELDD - Deletes tag or reference from list of data descriptors.
HDF_DELDD, *FileHandle, Tag, Ref*

HDF_DF24_ADDIMAGE - Writes 24-bit raster image to HDF file.
HDF_DF24_ADDIMAGE, *Filename, Image* [, /FORCE_BASELINE{useful only if QUALITY<25}] [, /JPEG | , /RLE] [, QUALITY=*value*{0 to 100}]

HDF_DF24_GETIMAGE - Reads 24-bit raster image from HDF file.
HDF_DF24_GETIMAGE, *Filename, Image* [, /LINE | , /PIXEL | , /PLANE]

HDF_DF24_GETINFO - Retrieves information about the current 24-bit HDF image.
HDF_DF24_GETINFO, *Filename, Width, Height, Interlace*

HDF_DF24_LASTREF - Returns reference number of most recently read or written 24-bit image in an HDF file.
Result = HDF_DF24_LASTREF( )

HDF_DF24_NIMAGES - Returns the number of 24-bit images in an HDF file.
Result = HDF_DF24_NIMAGES(*Filename*)

HDF_DF24_READREF - Sets reference number of image in an HDF file.
HDF_DF24_READREF, *Filename*, *Refno*

HDF_DF24_RESTART - Causes next call to HDF_DF24_GETIMAGE to read first 24-bit image in the HDF file.
HDF_DF24_RESTART

HDF_DFAN_ADDFDS - Adds file description to HDF file.
HDF_DFAN_ADDFDS, *Filename*, *Description*

HDF_DFAN_ADDFID - Adds file annotation to HDF file.
HDF_DFAN_ADDFID, *Filename*, *Label*

HDF_DFAN_GETDESC - Reads description for given tag and reference number in HDF file.
HDF_DFAN_GETDESC, *Filename*, *Tag*, *Ref*, *Description* [, /STRING]

HDF_DFAN_GETFDS - Reads next available file description.
HDF_DFAN_GETFDS, *Filename*, *Description* [, /FIRST] [, /STRING]

HDF_DFAN_GETFID - Reads next available file annotation.
HDF_DFAN_GETFID, *Filename*, *Label* [, /FIRST]

HDF_DFAN_GETLABEL - Reads label for given tag-reference pair.
HDF_DFAN_GETLABEL, *Filename*, *Tag*, *Ref*, *Label*

HDF_DFAN_LABLIST - Retrieves list of reference numbers and labels for given tag.
Result = HDF_DFAN_LABLIST(*Filename*, *Tag*, *Reflist*, *Labellist* [, LISTSIZE=*value*] [, MAXLABEL=*value*] [, STARTPOS=*value*] [, /STRING]

HDF_DFAN_LASTREF - Returns reference number of most recently read or written annotation.
Result = HDF_DFAN_LASTREF( )

HDF_DFAN_PUTDESC - Writes description for given tag and reference number.
HDF_DFAN_PUTDESC, *Filename*, *Tag*, *Ref*, *Description*

HDF_DFAN_PUTLABEL - Writes label for given tag and reference number.
HDF_DFAN_PUTLABEL, *Filename*, *Tag*, *Ref*, *Label*

HDF_DFP_ADDPAL - Appends palette to a HDF file.
HDF_DFP_ADDPAL, *Filename*, *Palette*

HDF_DFP_GETPAL - Reads next available palette from HDF file.
HDF_DFP_GETPAL, *Filename*, *Palette*

HDF_DFP_LASTREF - Returns reference number of most recently read or written palette in HDF file.
Result = HDF_DFP_LASTREF( )

HDF_DFP_NPALS - Returns number of palettes present in HDF file.
Result = HDF_DFP_NPALS(*Filename*)

HDF_DFP_PUTPAL - Appends palette to a HDF file.
HDF_DFP_PUTPAL, *Filename*, *Palette* [, /DELETE] [, /OVERWRITE]

HDF_DFP_READREF - Sets reference number of the palette.
HDF_DFP_READREF, *Filename*, *Refno*

HDF_DFP_RESTART - Causes next call to HDF_DFR8_GETPAL to read from the first palette in HDF file.
HDF_DFP_RESTART

HDF_DFP_WRITEREF - Sets reference number for next palette to be written to a HDF file.
HDF_DFP_WRITEREF, *Filename*, *Refno*

HDF_DFR8_ADDIMAGE - Appends 8-bit raster image to the specified HDF file.
HDF_DFR8_ADDIMAGE, *Filename*, *Image* [, /FORCE_BASELINE{useful only if QUALITY<25}] [, /JPEG | , /RLE] [[, /IMCOMP] , PALETTE=*vector or array*] [, QUALITY=*value*]

HDF_DFR8_GETIMAGE - Retrieves image and palette from HDF file.
HDF_DFR8_GETIMAGE, *Filename*, *Image* [, *Palette*]

HDF_DFR8_GETINFO - Retrieves information about the current 8-bit HDF image.
HDF_DFR8_GETINFO, *Filename*, *Width*, *Height*, *Has_Palette*

HDF_DFR8_LASTREF - Returns reference number of the most recently read or written 8-bit image in HDF file.
Result = HDF_DFR8_LASTREF( )

HDF_DFR8_NIMAGES - Returns number of 8-bit images in specified HDF file.
Result = HDF_DFR8_NIMAGES(*Filename*)

HDF_DFR8_PUTIMAGE - Writes 8-bit raster image as first image in HDF file.
HDF_DFR8_PUTIMAGE, *Filename*, *Image* [, /FORCE_BASELINE{useful only if QUALITY<25}] [[, /IMCOMP] , PALETTE=*vector or array*] [, /JPEG | , /RLE] [, QUALITY=*value*]

HDF_DFR8_READREF - Sets reference number of image to be read from a HDF file by the next call to HDF_DFR8_GETIMAGE.
HDF_DFR8_READREF, *Filename*, *Refno*

HDF_DFR8_RESTART - Causes next call to HDF_DFR8_GETIMAGE to read from first image in HDF file.
HDF_DFR8_RESTART

HDF_DFR8_SETPALETTE - Sets current palette to be used for subsequent images in a HDF file.
HDF_DFR8_SETPALETTE, *Palette*

HDF_DUPDD - Generates new references to existing data in HDF file.
HDF_DUPDD, *FileHandle*, *NewTag*, *NewRef*, *OldTag*, *OldRef*

HDF_EXISTS - Returns True if HDF format library is supported on the current IDL platform.
Result = HDF_EXISTS( )

HDF_GR_ATTRINFO - Retrieves information about specified HDF data object.
Result = HDF_GR_ATTRINFO(*obj_id*, *attr_index*, *name*, *data_type*, *count*)

HDF_GR_CREATE - Creates HDF GR raster image.
Result = HDF_GR_CREATE(*gr_id*, *name*, *ncomp*, *data_type*, *interlace_mode*, *dim_sizes*)

HDF_GR_END - Terminates specified HDF GR interface session.
HDF_GR_END, *gr_id*

HDF_GR_ENDACCESS - Terminates access to specified raster image.
HDF_GR_ENDACCESS, *ri_id*

HDF_GR_FILEINFO - Retrieves number of raster images and global attributes for the specified HDF GR interface.
Result = HDF_GR_FILEINFO(*gr_id*, *n_images*, *n_file_attrs*)

HDF_GR_FINDATTR - Finds index of HDF data object's attribute given its attribute name.
Result = HDF_GR_FINDATTR(*obj_id*, *attr_name*)

HDF_GR_GETATTR - Obtains all values of HDF GR attribute.
Result = HDF_GR_GETATTR(*obj_id*, *attr_index*, *values*)

HDF_GR_GETCHUNKINFO - Retrieves chunking information about HDF GR raster image.
Result = HDF_GR_GETCHUNKINFO(*ri_id*, *dim_length*, *flag*)

HDF_GR_GETIMINFO - Retrieves general information about HDF GR raster image.
Result = HDF_GR_GETIMINFO(*ri_id*, *gr_name*, *ncomp*, *data_type*, *interlace_mode*, *dim_sizes*, *num_attrs*)

HDF_GR_GETLUTID - Gets identifier of HDF GR palette.
Result = HDF_GR_GETLUTID(*ri_id*, *pal_index*)

HDF_GR_GETLUTINFO - Retrieves information about a palette.
Result = HDF_GR_GETLUTINFO(*pal_id*, *ncomp*, *data_type*, *interlace_mode*, *num_entries*)

HDF_GR_IDTOREF - Returns HDF reference number of specified raster image.
Result = HDF_GR_IDTOREF(*ri_id*)

HDF_GR_LUTTOREF - Returns HDF reference number of the specified palette.
Result = HDF_GR_LUTTOREF(*pal_id*)

HDF_GR_NAMETOINDEX - Returns index of raster image given its name
Result = HDF_GR_NAMETOINDEX(*gr_id*, *gr_name*)

HDF_GR_READIMAGE - Reads subsample of specified raster image.
Result = HDF_GR_READIMAGE(*ri_id*, *data* [, EDGE=*array*] [, /INTERLACE] [, START=*array*] [, STRIDE=*array*])

HDF_GR_READLUT - Reads specified palette.
Result = HDF_GR_READLUT(*pal_id*, *pal_data* [, /INTERLACE])

HDF_GR_REFTOINDEX - Returns index of specified raster image.
Result = HDF_GR_REFTOINDEX(*gr_id*, *gr_ref*)

HDF_GR_SELECT - Obtains identifier of specified raster image.
Result = HDF_GR_SELECT(*gr_id*, *index*)

HDF_GR_SETATTR - Attaches attribute to specified object.
Result = HDF_GR_SETATTR(*obj_id*, *attr_name*, *data_type*, *count*, *values*)

HDF_GR_SETCHUNK - Makes specified raster image a chunked raster image.
Result = HDF_GR_SETCHUNK(*ri_id*, *dim_length*, *comp_type*, *comp_prm*)

HDF_GR_SETCHUNKCACHE - Sets maximum number of chunks to be cached.
Result = HDF_GR_SETCHUNKCACHE(*ri_id*, *maxcache*, *flags*)

HDF_GR_SETCOMPRESS - Specifies whether specified raster image will be stored in compressed format.
Result = HDF_GR_SETCOMPRESS(*ri_id*, *comp_type*, *comp_prm*)

HDF_GR_SETEXTERNALFILE - Specifies that raster image will be written to external file.
Result = HDF_GR_SETEXTERNALFILE(*ri_id*, *filename*, *offset*)

HDF_GR_START - Initializes interface for the specified file.
Result = HDF_GR_START(*file_id*)

HDF_GR_WRITEIMAGE - Writes subsample of raster image data.
Result = HDF_GR_WRITEIMAGE(*ri_id*, *data* [, EDGE=*array*] [, INTERLACE={0 | 1 | 2}] [, START=*array*] [, STRIDE=*array*])

HDF_GR_WRITELUT - Writes a palette.
Result = HDF_GR_WRITELUT(*pal_id*, *pal_data* [, DATA_TYPE=*value*] [, INTERLACE_MODE={0 | 1 | 2}] [, NENTRIES=*value*])

HDF_HDF2IDLTYPE - Converts HDF data type code into IDL variable type code.
Result = HDF_HDF2IDLTYPE(*hdftypecode*)

HDF_IDL2HDFTYPE - Converts IDL variable type code into HDF data type code.
Result = HDF_IDL2HDFTYPE(*idltypecode*)

HDF_ISHDF - Determines whether specified file is HDF file.
Result = HDF_ISHDF(*Filename*)

HDF_LIB_INFO - Returns information about the HDF Library being used.

HDF_LIB_INFO, [*FileHandle*] [, MAJOR=*variable*] [, MINOR=*variable*] [, RELEASE=*variable*] [, VERSION=*variable*]

HDF_NEWREF - Returns next available reference number for HDF file.

Result = HDF_NEWREF(*FileHandle*)

HDF_NUMBER - Returns number of tags in HDF file or the number of references associated with a given tag.

Result = HDF_NUMBER(*FileHandle* [, TAG=*integer*])

HDF_OPEN - Opens or creates HDF file for reading and/or writing.

Result = HDF_OPEN(*Filename* [, /ALL] [, /CREATE] [, NUM_DD=*value*] [, /RDWR] [, /READ] [, /WRITE])

HDF_PACKDATA - Packs a set IDL variable into an array of raw byte data.

Result = HDF_PACKDATA(*data1* [, *data2* [, *data3* [, *data4* [, *data5* [, *data6* [, *data7* [, *data8*]]]]]]] [, HDF_ORDER=*array*] [, HDF_TYPE=*array*] [, NREC=*records*])

HDF_READ - See "HDF_READ" on page 16.

HDF_SD_ADDDATA - Writes hyperslab of values to an SD dataset.

HDF_SD_ADDDATA, *SDS_ID*, *Data* [, COUNT=*vector*] [, /NOREVERSE]] [, START=*vector*] [, STRIDE=*vector*]

HDF_SD_ATTRFIND - Locates index of HDF attribute given its name.

Result = HDF_SD_ATTRFIND(*S_ID*, *Name*)

HDF_SD_ATTRINFO - Reads or retrieves information about SD attribute.

HDF_SD_ATTRINFO, *S_ID*, *Attr_Index* [, COUNT=*variable*] [, DATA=*variable*] [, HDF_TYPE=*variable*] [, NAME=*variable*] [, TYPE=*variable*]

HDF_SD_ATTRSET - Writes attributes to an open HDF SD dataset.

HDF_SD_ATTRSET, *S_ID*, *Attr_Name*, *Values* [, *Count*] [, /BYTE] [, /DFNT_CHAR] [, /DFNT_FLOAT32] [, /DFNT_FLOAT64] [, /DFNT_INT8] [, /DFNT_INT16] [, /DFNT_INT32] [, /DFNT_UINT8] [, /DFNT_UINT16] [, /DFNT_UINT32] [, /DOUBLE] [, /FLOAT] [, /INT] [, /LONG] [, /SHORT] [, /STRING]

HDF_SD_CREATE - Creates and defines a Scientific Dataset for an HDF file.

Result = HDF_SD_CREATE(*SD_ID*, *Name*, *Dims* [, /BYTE] [, /DFNT_CHAR] [, /DFNT_FLOAT32] [, /DFNT_FLOAT64] [, /DFNT_INT8] [, /DFNT_INT16] [, /DFNT_INT32] [, /DFNT_UINT8] [, /DFNT_UINT16] [, /DFNT_UINT32] [, /DOUBLE] [, /FLOAT] [, HDF_TYPE=*type*] [, /INT] [, /LONG] [, /SHORT] [, /STRING])

HDF_SD_DIMGET - Retrieves info. about SD dataset dimension.

HDF_SD_DIMGET, *Dim_ID* [, /COUNT] [, COMPATIBILITY=*variable*] [, /FORMAT] [, /LABEL] [, /NAME] [, /NATTR] [, /SCALE] [, /TYPE] [, /UNIT]

HDF_SD_DIMGETID - Returns dimension ID given a dataset "SDS_ID" and dimension number.

Result = HDF_SD_DIMGETID(*SDS_ID*, *Dimension_Number*)

HDF_SD_DIMSET - Sets scale and data strings for SD dimension.

HDF_SD_DIMSET, *Dim_ID* [, /BW_INCOMP] [, FORMAT=*string*] [, LABEL=*string*] [, NAME=*string*] [, SCALE=*vector*] [, UNIT=*string*]

HDF_SD_END - Closes SD interface to an HDF file.

HDF_SD_END, *SD_ID*

HDF_SD_ENDACCESS - Closes SD dataset interface.

HDF_SD_ENDACCESS, *SD_ID*

HDF_SD_FILEINFO - Retrieves the number of datasets and global attributes in HDF file.

HDF_SD_FILEINFO, *SD_ID*, *Datasets*, *Attributes*

HDF_SD_GETDATA - Retrieves a hyperslab of values from SD dataset.

HDF_SD_GETDATA, *SDS_ID*, *Data* [, COUNT=*vector*] [, /NOREVERSE] [, START=*vector*] [, STRIDE=*vector*]

HDF_SD_GETINFO - Retrieves information about SD dataset.

HDF_SD_GETINFO, *SDS_ID* [, CALDATA=*variable*] [, COORDSYS=*variable*] [, DIMS=*variable*] [, FILL=*variable*] [, FORMAT=*variable*] [, HDF_TYPE=*variable*] [, LABEL=*variable*] [, NAME=*variable]* [, NATTS=*variable*] [, NDIMS=*variable*] [, /NOREVERSE] [, RANGE=*variable*] [, TYPE=*variable*] [, UNIT=*variable*]

HDF_SD_IDTOREF - Converts SD data set ID into SD data set reference number.

Result = HDF_SD_IDTOREF(*SDS_ID*)

HDF_SD_ISCOORDVAR - Determines whether supplied dataset ID represents NetCDF "coordinate" variable.

Result = HDF_SD_ISCOORDVAR(*SDS_ID*)

HDF_SD_NAMETOINDEX - Returns SD dataset index given its name and SD interface ID.

Result = HDF_SD_NAMETOINDEX(*SD_ID*, *SDS_Name*)

HDF_SD_REFTOINDEX - Returns SD dataset index given its reference number and SD interface ID.

Result = HDF_SD_REFTOINDEX(*SD_ID*, *Refno*)

HDF_SD_SELECT - Returns SD dataset ID.

Result = HDF_SD_SELECT(*SD_ID*, *Number*)

HDF_SD_SETCOMPRESS - Compresses an existing HDF SD dataset or sets the compression method of a new HDF SD dataset.

HDF_SD_SETCOMPRESS, *SDS_ID*, *comptype* [, EFFORT=*integer*{1 to 9}]

HDF_SD_SETEXTFILE - Moves data values from a dataset into an external file.
HDF_SD_SETEXTFILE, *SDS_ID*, *Filename* [, OFFSET=*bytes*]

HDF_SD_SETINFO - Sets information about SD dataset.
HDF_SD_SETINFO, *SDS_ID* [, FILL=*value*] [, FORMAT=*string*] [, LABEL=*string*] [, RANGE=*[max, min]*] [, UNIT=*string*] [, COORDSYS=*string*] [, CALDATA=*structure*]

HDF_SD_START - Opens or creates HDF file and initializes SD interface.
Result = HDF_SD_START(*Filename* [, /READ | , /RDWR] [, /CREATE])

HDF_UNPACKDATA - Unpacks array of byte data into IDL variables.
HDF_UNPACKDATA, *packeddata*, *data1* [, *data2* [, *data3* [, *data4* [, *data5* [, *data6* [, *data7* [, *data8*]]]]]]] [, HDF_ORDER=*array*] [, HDF_TYPE=*array*] [, NREC=*records*]

HDF_VD_ATTACH - Accesses a VData with the given ID.
Result = HDF_VD_ATTACH(*FileHandle*, *VDataId* [, /READ] [, /WRITE])

HDF_VD_DETACH - Called when done accessing a VData.
HDF_VD_DETACH, *VData*

HDF_VD_FDEFINE - Adds new field specification for VData.
HDF_VD_FDEFINE, *VData*, *Fieldname* [, / BYTE | , /DLONG, | , /DOUBLE | , /DULONG | , /FLOAT | , /INT | , /LONG | , /UINT, | , /ULONG] [, ORDER=*value*]

HDF_VD_FEXIST - Returns true if specified fields exist in HDF file.
Result = HDF_VD_FEXIST(*VData*, *Fieldnames*)

HDF_VD_FIND - Returns reference number of specified VData.
Result = HDF_VD_FIND(*FileHandle*, *Name*)

HDF_VD_GET - Returns information about a VData.
HDF_VD_GET, *VData* [, CLASS=*variable*] [, COUNT=*variable*] [, FIELDS=*variable*] [, INTERLACE=*variable*] [, NAME=*variable*] [, NFIELDS=*variable*] [, REF=*variable*] [, SIZE=*variable*] [, TAG=*variable*]

HDF_VD_GETID - Returns VData reference number for next VData.
Result = HDF_VD_GETID(*FileHandle*, *VDataId*)

HDF_VD_GETINFO - Returns information about each Vdata field.
HDF_VD_GETINFO, *VData*, *Index* [, NAME=*variable*] [, ORDER=*variable*] [, SIZE=*variable*] [, TYPE=*variable*]

HDF_VD_GETNEXT - Returns reference number of the next object inside a VData.
Result = HDF_VD_GETNEXT(*VData*, *Id*)

HDF_VD_INSERT - Adds VData or VGroup to contents of VGroup.
HDF_VD_INSERT, *VGroup*, *VData(or Vgroup)*[, POSITION=*variable*]

HDF_VD_ISVD - Returns True (1) if an object is a VData.
Result = HDF_VD_ISVD(*VGroup*, *Id*)

HDF_VD_ISVG - Returns True (1) if object is a VGroup.
Result = HDF_VG_ISVG(*VGroup*, *Id*)

HDF_VD_LONE - Returns array containing all VDatas that are not contained in another VData.
Result = HDF_VD_LONE(*FileHandle* [, MAXSIZE=*value*])

HDF_VD_READ - Reads data from a VData.
Result = HDF_VD_READ(*VData*, *Data* [, FIELDS=*string*] [, /FULL_INTERLACE | , /NO_INTERLACE] [, NRECORDS=*records*])

HDF_VD_SEEK - Moves read pointer in specified VData to specific record number.
HDF_VD_SEEK, *VData*, *Record*

HDF_VD_SETINFO - Specifies general information about a VData.
HDF_VD_SETINFO, *VData* [, CLASS=*string*] [, /FULL_INTERLACE | , /NO_INTERLACE] [, NAME=*string*]

HDF_VD_WRITE - Stores data in a VData
HDF_VD_WRITE, *VData*, *Fields*, *Data* [, /FULL_INTERLACE | , /NO_INTERLACE] [, NRECORDS=*records*]

HDF_VG_ADDTR - Adds tag and reference to specified VGroup.
HDF_VG_ADDTR, *VGroup*, *Tag*, *Ref*

HDF_VG_ATTACH - Attaches (opens) a VGroup.
Result = HDF_VG_ATTACH(*FileHandle*, *VGroupId* [, /READ] [, /WRITE])

HDF_VG_DETACH - Called when finished accessing a VGroup.
HDF_VG_DETACH, *VGroup*

HDF_VG_GETID - Returns VGroup ID for specified VGroup.
Result = HDF_VG_GETID(*FileHandle*, *VGroupId*)

HDF_VG_GETINFO - Returns information about a VGroup.
HDF_VG_GETINFO, *VGroup* [, CLASS=*variable*] [, NAME=*variable*] [, NENTRIES=*variable*]

HDF_VG_GETNEXT - Returns reference number of the next object in a VGroup.
Result = HDF_VG_GETNEXT(*VGroup*, *Id*)

HDF_VG_GETTR - Returns tag/reference pair at specified position within a VGroup.
HDF_VG_GETTR, *VGroup*, *Index*, *Tags*, *Refs*

HDF_VG_GETTRS - Returns tag/reference pairs of HDF file objects belonging to the specified VGroup.
HDF_VG_GETTRS, *VGroup*, *Tags*, *Refs* [, MAXSIZE=*value*]

HDF_VG_INQTR - Returns true if specified tag/reference pair is linked to the specified Vgroup.
Result = HDF_VG_INQTR(*VGroup*, *Tag*, *Ref*)

HDF_VG_INSERT - Adds VData or VGroup to contents of VGroup.
HDF_VG_INSERT, *VGroup*, *VData(or Vgroup)*[, POSITION=*variable*]

HDF_VG_ISVD - Returns true if object is a VData.
Result = HDF_VG_ISVD(*VGroup*, *Id*)

HDF_VG_ISVG - Returns true if object is a VGroup.
Result = HDF_VG_ISVG(*VGroup*, *Id*)

HDF_VG_LONE - Returns array containing IDs of all VGroups that are not contained in another VGroup.
Result = HDF_VG_LONE(*FileHandle* [, MAXSIZE=*value*])

HDF_VG_NUMBER - Returns number of HDF file objects in specified VGroup.
Result = HDF_VG_NUMBER(*VGroup*)

HDF_VG_SETINFO - Sets the name and class of a VGroup.
HDF_VG_SETINFO, *VGroup* [, CLASS=*string*] [, NAME=*string*]

NetCDF Routines

NCDF_ATTCOPY - Copies attribute from one netCDF file to another.
Result = NCDF_ATTCOPY(*Incdf* [, *Invar* | , /IN_GLOBAL] , *Name*, *Outcdf* [, *Outvar*] [, /OUT_GLOBAL])

NCDF_ATTDEL - Deletes an attribute from a netCDF file.
NCDF_ATTDEL, *Cdfid* [, *Varid* | , /GLOBAL] , *Name*

NCDF_ATTGET - Retrieves value of an attribute from a netCDF file.
NCDF_ATTGET, *Cdfid* [, *Varid* | , /GLOBAL] , *Name*, *Value*

NCDF_ATTINQ - Returns information about a netCDF attribute.
Result = NCDF_ATTINQ(*Cdfid* [, *Varid* | , /GLOBAL] , *Name*)

NCDF_ATTNAME - Returns the name of an attribute given its ID.
Result = NCDF_ATTNAME(*Cdfid* [, *Varid* | , /GLOBAL] , *Attnum*)

NCDF_ATTPUT - Creates an attribute in a netCDF file.
NCDF_ATTPUT, *Cdfid* [, *Varid* | , /GLOBAL] , *Name* , *Value* [, LENGTH=*value*] [, /BYTE | , /CHAR | , /DOUBLE | , /FLOAT | , /LONG | , /SHORT]

NCDF_ATTRENAME - Renames an attribute in a netCDF file.
NCDF_ATTRENAME, *Cdfid* [, *Varid* | , /GLOBAL] *Oldname*, *Newname*

NCDF_CLOSE - Closes an open netCDF file.
NCDF_CLOSE, *Cdfid*

NCDF_CONTROL - Performs miscellaneous netCDF operations.
NCDF_CONTROL, *Cdfid* [, /ABORT] [, /ENDEF] [, /FILL | , /NOFILL] [, /NOVERBOSE | , /VERBOSE] [, OLDFILL=*variable*] [, /REDEF] [, /SYNC]

NCDF_CREATE - Creates a new netCDF file.
Result = NCDF_CREATE(*Filename* [, /CLOBBER | , /NOCLOBBER])

NCDF_DIMDEF - Defines a dimension in a netCDF file given its name and size.
Result = NCDF_DIMDEF(*Cdfid*, *DimName*, *Size* [, /UNLIMITED])

NCDF_DIMID - Returns the ID of a netCDF dimension, given the name of the dimension.
Result = NCDF_DIMID(*Cdfid*, *DimName*)

NCDF_DIMINQ - Retrieves the name and size of a dimension in a netCDF file, given its ID.
NCDF_DIMINQ, *Cdfid*, *Dimid*, *Name*, *Size*

NCDF_DIMRENAME - Renames an existing dimension in a netCDF file that has been opened for writing.
NCDF_DIMRENAME, *Cdfid*, *Dimid*, *NewName*

NCDF_EXISTS - Returns True if the netCDF format library is supported on the current IDL platform.
Result = NCDF_EXISTS( )

NCDF_INQUIRE - Returns information about an open netCDF file.
Result = NCDF_INQUIRE(*Cdfid*)

NCDF_OPEN - Opens an existing netCDF file.
Result = NCDF_OPEN(*Filename* [, /NOWRITE | , /WRITE])

NCDF_VARDEF - Adds a new variable to an open netCDF file in define mode.
Result = NCDF_VARDEF(*Cdfid*, *Name* [, *Dim*] [, /BYTE | , /CHAR | , /DOUBLE | , /FLOAT | , /LONG | , /SHORT])

NCDF_VARGET - Retrieves a hyperslab of values from a netCDF variable.
NCDF_VARGET, *Cdfid*, *Varid*, *Value* [, COUNT=*vector*] [, OFFSET=*vector*] [, STRIDE=*vector*]

NCDF_VARGET1 - Retrieves one element from a netCDF variable.
NCDF_VARGET1, *Cdfid*, *Varid*, *Value* [, OFFSET=*vector*]

NCDF_VARID - Returns the ID of a netCDF variable.
Result = NCDF_VARID(*Cdfid*, *Name*)

NCDF_VARINQ - Returns information about a netCDF variable, given its ID.
Result = NCDF_VARINQ(*Cdfid*, *Varid*)

NCDF_VARPUT - Writes a hyperslab of values to a netCDF variable.
NCDF_VARPUT, *Cdfid*, *Varid*, *Value* [, COUNT=*vector*] [, OFFSET=*vector*] [, STRIDE=*vector*]

NCDF_VARRENAME - Renames a netCDF variable.
NCDF_VARRENAME, *Cdfid*, *Varid*, *Name*

Objects

This section lists all IDL objects and their methods. In addition to the syntax conventions discussed in "IDL Syntax" on page 3, note the following:

- The *Object_Name*::Init method for each object has keywords that are followed by either {Get}, {Set}, or {Get, Set}. Properties retrievable via *Object_Name*::GetProperty are indicated by {Get}; properties settable via *Object_Name*::SetProperty are indicated by {Set}. Properties that are both retrievable and settable are indicated by {Get, Set}. Do not include the braces, Get, or Set in your call.
- Each object's Cleanup method lists two possible syntaxes. The second syntax (*Obj* -> *Object_Name*::Cleanup) can be used only in a subclass' Cleanup method.
- Some objects have Init methods that list two possible syntaxes. The second syntax (*Obj* -> *Object_Name*::Init) can be used only in a subclass' Init method.

IDL_Container - Object used to hold other objects. No superclasses. Subclasses: IDLgrModel IDLgrScene IDLgrView IDLgrViewgroup.

IDL_Container::Add - Adds a child object to the container.
Obj -> [IDL_Container::]Add,*Object* [POSITION=*index*])

IDL_Container::Cleanup - Performs all cleanup on the object.
OBJ_DESTROY, *Obj* or *Obj* -> [IDL_Container::]Cleanup

IDL_Container::Count - Returns the number of objects contained by the container object.
Result = *Obj* -> [IDL_Container::]Count( )

IDL_Container::Get - Returns an array of object references to objects in a container.
Result = *Obj* -> [IDL_Container::]Get(
[, /ALL [, ISA=*class_name(s)*] | , POSITION=*index*]
[COUNT=*variable*])

IDL_Container::Init - Initializes the container object.
Obj = OBJ_NEW('IDL_Container')
Result = *Obj* -> [IDL_Container::]Init( )

IDL_Container::IsContained - Returns true (1) if the specified object is in the container, or false (0) otherwise.
Result = *Obj* -> [IDL_Container::]IsContained(*Object*
[, POSITION=*variable*])

IDL_Container::Move - Moves an object from one position in a container to a new position.
Obj -> [IDL_Container::]Move, *Source*, *Destination*

IDL_Container::Remove - Removes an object from the container.
Obj -> [IDL_Container::]Remove [, *Child_object* | ,
POSITION=*index* | , /ALL]

IDLanROI - Represents a region of interest. Superclass of IDLgrROI.

IDLanROI::AppendData - Appends vertices to the region.
Obj–>[IDLanROI::]AppendData, *X* [, *Y*] [, *Z*]
[, XRANGE=*variable*] [, YRANGE=*variable*]
[, ZRANGE=*variable*]

IDLanROI::Cleanup - Performs all cleanup for the object.
Obj–>[IDLanROI::]Cleanup or OBJ_DESTROY, *Obj*

IDLanROI::ComputeGeometry - Computes the geometrical values for area, perimeter, and/or centroid of the region.
Result = *Obj*–>[IDLanROI::]ComputeGeometry
[, AREA=*variable*] [, CENTROID=*variable*]
[, PERIMETER=*variable*] [, SPATIAL_OFFSET=*vector*]
[, SPATIAL_SCALE=*vector*]

IDLanROI::ComputeMask - Prepares a two-dimensional mask for the region.
Result = *Obj*–>[IDLanROI::]ComputeMask(
[, INITIALIZE={ –1 | 0 | 1 }]
[, DIMENSIONS=*[xdim, ydim]*] | [, MASK_IN=*array*]
[, LOCATION=*[x, y* [, *z]]*] [, MASK_RULE={ 0 | 1 | 2 }]
[, PLANE_NORMAL=*[x, y, z]*]
[, PLANE_XAXIS=*[x,y,z]*])

IDLanROI::ContainsPoints - Determines whether the given data coordinates are contained within the closed polygon region.
Result = *Obj*–>[IDLanROI::]ContainsPoints(*X* [, *Y* [, *Z*]])

IDLanROI::GetProperty - Retrieves the value of a property or group of properties for the region.
Obj–>[IDLanROI::]GetProperty [, ALL=*variable*]
[, ROI_XRANGE=*variable*] [, ROI_YRANGE=*variable*]
[, ROI_ZRANGE=*variable*]

IDLanROI::Init - Initializes a region of interest object.
Result = IDLanROI::Init([*X* [, *Y* [, *Z*]]]
[, BLOCKSIZE{Get, Set}=*vertices*] [, DATA{Get,
Set}=*array*] [, /INTERIOR{Get, Set}]
[, TYPE{Get}={ 0 | 1 | 2 }])
or
Obj = OBJ_NEW('IDLanROI' [, *X* [, *Y* [, *Z*]]])

[, START=*index*] [, XRANGE=*variable*]
[, YRANGE=*variable*][, ZRANGE=*variable*]

IDLanROI::RemoveData - Removes vertices from the region.
Obj–>[IDLanROI::]RemoveData[, COUNT=*vertices*]

IDLanROI::ReplaceData - Replaces vertices in the region with alternate values.
Obj–>[IDLanROI::]ReplaceData, *X*[, *Y*[, *Z*]]
[, START=*index*] [, FINISH=*index*]
[, XRANGE=*variable*] [, YRANGE=*variable*]
[, ZRANGE=*variable*]

IDLanROI::Rotate - Modifies the vertices for the region by applying a rotation.
Obj–>[IDLanROI::]Rotate, *Axis*, *Angle*
[, CENTER=*[x, y*[, *z]]*]

IDLanROI::Scale - Modifies the vertices for the region by applying a scale.
Obj–>[IDLanROI::]Scale, *Sx*[, *Sy*[, *Sz*]]

IDLanROI::SetProperty - Sets the value of a property or group of properties for the region.
Obj–>[IDLanROI::]SetProperty

IDLanROI::Translate - Modifies the vertices for the region by applying a translation.
Obj–>[IDLanROI::]Translate, *Tx*[, *Ty*[, *Tz*]]

IDLanROIGroup - This object is an analytical representation of a group of regions of interest. Subclass of IDL_Container. Superclass of IDLgrROIGroup.

IDLanROIGroup::Add - Adds a region to the region group.
Obj–>[IDLanROIGroup::]Add, *ROI*

IDLanROIGroup::Cleanup - Performs all cleanup for the object.
Obj–>[IDLanROIGroup::]Cleanup
or OBJ_DESTROY, *Obj*

IDLanROIGroup::ContainsPoints - Determines whether the given points (in data coordinates) are contained within the closed polygon regions within this group.
Result = *Obj*–>[IDLanROIGroup::]ContainsPoints(
X[, *Y*[, *Z*]])

IDLanROIGroup::ComputeMask - Prepares a 2-D mask for this group of regions.
Result = *Obj*–>[IDLanROIGroup::]ComputeMask(
[, INITIALIZE={ –1 | 0 | 1 }]
[, DIMENSIONS=*[xdim, ydim]*] | [, MASK_IN=*array*]
[, LOCATION=*[x, y* [, *z]]*] [, MASK_RULE={ 0 | 1 | 2 }])

IDLanROIGroup::ComputeMesh - Triangulates a surface mesh with optional capping from the stack of regions contained within this group.
Result = *Obj*->[IDLanROIGroup::]ComputeMesh(
Vertices, *Conn* [, CAPPED={ 0 | 1 | 2}]
[, SURFACE_AREA=*variable*])

IDLanROIGroup::GetProperty - Retrieves the value of a property or group of properties for the region group.
Obj–>[IDLanROIGroup::]GetProperty[, ALL=*variable*]
[, ROIGROUP_XRANGE=*variable*]
[, ROIGROUP_YRANGE=*variable*]
[, ROIGROUP_ZRANGE=*variable*]

IDLanROIGroup::Init - Initializes a region of interest group object.
Result = IDLanROIGroup::Init()
or *Obj* = OBJ_NEW('IDLanROIGroup')

IDLanROIGroup::Rotate - Modifies the vertices for all regions within the group by applying a rotation.
Obj–>[IDLanROIGroup::]Rotate, *Axis*,
Angle[, CENTER=*[x, y*[, *z*]*]*]

IDLanROIGroup::Scale - Modifies the vertices for the region by applying a scale.
Obj–>[IDLanROIGroup::]Scale, *Sx*[, *Sy*[, *Sz*]]

IDLanROIGroup::Translate - Modifies the vertices of all regions within the group by applying a translation.
Obj–>[IDLanROIGroup::]Translate, *Tx*[, *Ty*[, *Tz*]]

IDLffDICOM - Contains the data for one or more images embedded in a DICOM part 10 file. No superclasses. No subclasses.

IDLffDICOM::Cleanup - Destroys the IDLffDICOM object.
OBJ_DESTROY, *Obj*
or
OBJ -> [IDLffDICOM::]Cleanup

IDLffDICOM::DumpElements - Dumps a description of the DICOM data elements of IDLffDICOM object to the screen or to a file.
Obj -> [IDLffDICOM::]DumpElements [, *Filename*]

IDLffDICOM::GetChildren - Finds the member element references of a DICOM sequence.
array = *Obj* -> [IDLffDICOM::]GetChildren(*Reference*)

IDLffDICOM::GetDescription - Takes optional DICOM group and element arguments and returns array of STRING descriptions.
array = *Obj* -> [IDLffDICOM::]GetDescription([*Group*
[, *Element*]] [, REFERENCE=*list of element references*])

IDLffDICOM::GetElement - Takes optional DICOM group and/or element arguments and returns an array of DICOM Element numbers for those parameters.
array = *Obj* -> [IDLffDICOM::]GetElement([*Group*
[, *Element*]] [, REFERENCE=*list of element references*])

IDLffDICOM::GetGroup - Takes optional DICOM group and/or element arguments and returns an array of DICOM Group numbers for those parameters.
array = *Obj* -> [IDLffDICOM::]GetGroup([*Group*
[, *Element*]] [, REFERENCE=*list of element references*])

IDLffDICOM::GetLength - Takes optional DICOM group and/or element arguments and returns an array of LONGs.
array = *Obj* -> [IDLffDICOM::]GetLength([*Group*
[, *Element*]] [, REFERENCE=*list of element references*])

IDLffDICOM::GetParent - Finds the parent references of a set of elements in a DICOM sequence.

array = *Obj* ->[IDLffDICOM::]GetParent(*ReferenceList*)

IDLffDICOM::GetPreamble - Returns the preamble of a DICOM v3.0 Part 10 file.

array = *Obj* -> [IDLffDICOM::]GetPreamble( )

IDLffDICOM::GetReference - Takes optional DICOM group and/or element arguments and returns an array of references to matching elements in the object.

array = *Obj* -> [IDLffDICOM::]GetReference([*Group* [, *Element*]] [, DESCRIPTION=*string*] [, VR=*DICOM VR string*])

IDLffDICOM::GetValue - Takes optional DICOM group and/or element arguments and returns an array of POINTERs to the values of the elements matching those parameters.

ptrArray = *Obj* -> [IDLffDICOM::]GetValue([*Group* [, *Element*]] [, REFERENCE=*list of element references*] [, /NO_COPY])

IDLffDICOM::GetVR - Takes optional DICOM group and/or element arguments and returns an array of VR (Value Representation) STRINGs for those parameters.

array = *Obj* -> [IDLffDICOM::]GetVR([*Group* [, *Element*]] [, REFERENCE=*list of references*])

IDLffDICOM::Init - Creates a new IDLffDICOM object and optionally reads the specified file as defined in the IDLffDICOM::Read method.

result = OBJ_NEW('IDLffDICOM' [, *Filename*] [, /VERBOSE])

IDLffDICOM::Read - Opens and reads from the specified disk file, places the information into the DICOM object, then closes the file.

result = *Obj* -> [IDLffDICOM::]Read(*Filename* [, ENDIAN={1 | 2 | 3 | 4}])

IDLffDICOM::Reset - Removes all of the elements from the IDLffDICOM object, leaving the object otherwise intact.

Obj -> [IDLffDICOM::]Reset

IDLffDXF - Object that contains geometry, connectivity, and attributes for graphics primitives. No superclasses. No subclasses.

IDLffDXF::Cleanup - Performs all cleanup on the object.

OBJ_DESTROY, *Obj* or *Obj* -> [IDLffDXF::]Cleanup

IDLffDXF::GetContents - Returns the DXF entity types contained in the object.

Result = *Obj*-> [IDLffDXF::]GetContents([*Filter*] [BLOCK=*string*] [, COUNT=*variable*] [LAYER=*string*])

IDLffDXF::GetEntity - Returns an array of vertex data for the requested entity type.

Result = *Obj*-> [IDLffDXF::]GetEntity(*Type* [, BLOCK=*string*] [, INDEX=*value*] [, LAYER=*string*])

IDLffDXF::GetPalette - Returns current color table in the object.

Obj-> [IDLffDXF::]GetPalette, *Red*, *Green*, *Blue*

IDLffDXF::Init - Initializes the DXF object.

Result = OBJ_NEW('IDLffDXF'[, *Filename*])

IDLffDXF::PutEntity - Inserts an entity into the DXF object.

Obj -> [IDLffDXF::]PutEntity, *Data*

IDLffDXF::Read - Reads a file, parsing the DXF object information contained in the file, and inserts it into itself.

Result = *Obj*-> [IDLffDXF::]Read(*Filename*)

IDLffDXF::RemoveEntity - Removes the specified entity or entities from the DXF object.

Obj -> [IDLffDXF::]RemoveEntity[, *Type*] [, INDEX=*value*]

IDLffDXF::Reset - Removes all the entities from the DXF object.

Obj-> [IDLffDXF::]Reset

IDLffDXF::SetPalette - Sets the current color table in the object.

Obj-> [IDLffDXF::]SetPalette, *Red*, *Green*, *Blue*

IDLffDXF::Write - Writes a file for the DXF entity information this object contains.

Result = *Obj*-> [IDLffDXF::]Write(*Filename*)

IDLffLanguageCat - Provides an interface to IDL language catalog files.

IDLffLanguageCat::IsValid - Determines whether the object has a valid catalog.

Result = *Obj* ->[IDLffLanguageCat::]IsValid( )

IDLffLanguageCat::Query - Returns the language string associated with the given key.

Result = *Obj* ->[IDLffLanguageCat::]Query(*key* [, DEFAULT_STRING=*string*])

IDLffLanguageCat::SetCatalog - Sets appropriate catalog file.

Result = *Obj* ->[IDLffLanguageCat::]SetCatalog(*application* [, FILENAME=*string*] [, LOCALE=*string*] [, PATH=*string*])

IDLgrAxis - An axis object represents a single vector that may include a set of tick marks, tick labels, and a title. No superclasses. No subclasses.

IDLgrAxis::Cleanup - Performs all cleanup on the object.

OBJ_DESTROY, *Obj* or *Obj* -> [IDLgrAxis::]Cleanup

IDLgrAxis::GetCTM - Returns the 4 x 4 graphics transform matrix from the current object upward through the graphics tree.

Result = *Obj* -> [IDLgrAxis::]GetCTM([, DESTINATION=*objref*] [, PATH=*objref(s)*] [, TOP=*objref*])

IDLgrAxis::GetProperty - Retrieves the value of a property or group of properties for the axis.

Obj -> [IDLgrAxis::]GetProperty [, ALL=*variable*] [, CRANGE=*variable*] [, PARENT=*variable*] [, XRANGE=*variable*] [, YRANGE=*variable*] [, ZRANGE=*variable*]

Note: See also the {Get} properties in IDLgrAxis::Init

IDLgrAxis::Init - Initializes an axis object.

Obj = OBJ_NEW('IDLgrAxis' [, *Direction*]) or *Result* = *Obj* -> [IDLgrAxis::]Init([*Direction*] [, COLOR{Get, Set}=*index or RGB_vector*]

[, DIRECTION{Get, Set}=*integer*] [, /EXACT{Get, Set}] [, /EXTEND{Get, Set}] [, GRIDSTYLE{Get, Set}=*integer*{0 to 6}or *[repeat*{1 to 255}, *bitmask]*] [, /HIDE{Get, Set}] [, LOCATION{Get, Set}=*[x, y]* or *[x, y, z]*] [, /LOG{Get, Set}] [, MAJOR{Get, Set}=*integer*] [, MINOR{Get, Set}=*integer*] [, NAME{Get, Set}=*string*] [, /NOTEXT{Get, Set}] [, PALETTE{Get, Set}=*objref*] [, RANGE{Get, Set}=*[min, max]*] [, SUBTICKLEN{Get, Set}=*value*] [, TEXTALIGNMENTS{Get, Set}=*[horiz*{0.0 to 1.0}, *vert*{0.0 to 1.0}*]*] [, TEXTBASELINE{Get, Set}=*vector*] [, TEXTPOS{Get, Set}={0 | 1}] [, TEXTUPDIR{Get, Set}=*vector*] [, THICK{Get, Set}=*points*{1 to 10}] [, TICKDIR{Get, Set}={0 | 1}] [, TICKFORMAT{Get, Set}=*string*] [, TICKFRMTDATA{Get, Set}=*value*] [, TICKLEN{Get, Set}=*value*] [, TICKTEXT{Get, Set}=*objref or vector*] [, TICKVALUES{Get, Set}=*vector*] [, TITLE{Get, Set}=*objref*] [, /USE_TEXT_COLOR{Get, Set}] [, UVALUE{Get, Set}=*value*] [, XCOORD_CONV{Get, Set}=*vector*] [, YCOORD_CONV{Get, Set}=*vector*] [, ZCOORD_CONV{Get, Set}=*vector*])

IDLgrAxis::SetProperty - Sets the value of a property or group of properties for the axis.

Obj -> [IDLgrAxis::]SetProperty

Note: See also the {Set} properties in IDLgrAxis::Init

IDLgrBuffer - An in-memory, off-screen destination object. No superclasses. No subclasses.

IDLgrBuffer::Cleanup - Performs all cleanup on the object.

OBJ_DESTROY, *Obj* or *Obj* -> [IDLgrBuffer::]Cleanup

IDLgrBuffer::Draw - Draws picture to this graphics destination.

Obj -> [IDLgrBuffer::]Draw [, *Picture*] [, CREATE_INSTANCE={1 | 2}] [, /DRAW_INSTANCE]

IDLgrBuffer::Erase - Erases this graphics destination.

Obj -> [IDLgrBuffer::]Erase [, COLOR=*index or RGB vector*]

IDLgrBuffer::GetContiguousPixels - Returns an array of long integers whose length is equal to the number of colors available in the index color mode (value of the N_COLORS property).

Return = *Obj* -> [IDLgrBuffer::]GetContiguousPixels( )

IDLgrBuffer::GetDeviceInfo - Returns information that allows IDL applications to make decisions for optimal performance.

Result = *Obj*–>[IDLgrBuffer::]GetDeviceInfo ([, ALL=*variable*] [, MAX_TEXTURE_DIMENSIONS=*variable*] [, MAX_VIEWPORT_DIMENSIONS=*variable*] [, NAME=*variable*] [, NUM_CPUS=*variable*] [, VENDOR=*variable*] [, VERSION=*variable*])

IDLgrBuffer::GetFontnames - Returns the list of available fonts that can be used in IDLgrFont objects.

Return = *Obj* -> [IDLgrBuffer::]GetFontnames(*FamilyName*[, IDL_FONTS={0 | 1 | 2 }] [, STYLES=*string*])

IDLgrBuffer::GetProperty - Retrieves the value of a property or group of properties for the buffer.

Obj -> [IDLgrBuffer::]GetProperty [, ALL=*variable*] [, IMAGE_DATA=*variable*] [, SCREEN_DIMENSIONS=*variable*] [, ZBUFFER_DATA=*variable*]

Note: See also the {Get} properties in IDLgrBuffer::Init

IDLgrBuffer::GetTextDimensions - Retrieves the dimensions of a text object that will be rendered in the buffer.

Result = *Obj* ->[IDLgrBuffer::]GetTextDimensions(*TextObj* [, DESCENT=*variable*] [, PATH=*objref(s)*])

IDLgrBuffer::Init - Initializes the buffer object.

Obj = OBJ_NEW('IDLgrBuffer') or *Result* = *Obj* -> [IDLgrBuffer::]Init([, COLOR_MODEL{Get}={0 | 1}] [, DIMENSIONS{Get, Set}=*[width, height]*] [, GRAPHICS_TREE{Get, Set}=*objref*] [, N_COLORS{Get}=*integer*{2 to 256}] [, PALETTE{Get, Set}=*objref*] [, QUALITY{Get, Set}={0 | 1 | 2}] [, RESOLUTION{Get, Set}=*[xres, yres]*] [, UNITS{Get, Set}={0 | 1 | 2 | 3}] [, UVALUE{Get, Set}=*value*])

IDLgrBuffer::PickData - Maps a point in the 2D device space of the buffer to a point in the 3D data space of an object tree.

Result = *Obj* -> [IDLgrBuffer::]PickData(*View*, *Object*, *Location*, *XYZLocation* [, PATH=*objref(s)*])

IDLgrBuffer::Read - Reads an image from a buffer.

Result = *Obj* -> [IDLgrBuffer::]Read( )

IDLgrBuffer::Select - Returns a list of objects selected at a specified location.

Result = *Obj* -> [IDLgrBuffer::]Select(*Picture, XY* [, DIMENSIONS=*[width, height]*] [, UNITS={0 | 1 | 2 | 3}])

IDLgrBuffer::SetProperty - Sets the value of a property or group of properties for the buffer.

Obj -> [IDLgrBuffer::]SetProperty

Note: See also the {Set} properties in IDLgrBuffer::Init

IDLgrClipboard - A destination object representing the native clipboard. No superclasses. No Subclasses.

IDLgrClipboard::Cleanup - Performs all cleanup on the object.

OBJ_DESTROY,*Obj* or *Obj*->[IDLgrClipboard::]Cleanup

IDLgrClipboard::Draw - Draws a picture to a graphics destination.

Obj -> [IDLgrClipboard::]Draw [, *Picture*] [, FILENAME=*string*] [, POSTSCRIPT=*value*] [, VECTOR={ 0 | 1 }]

IDLgrClipboard::GetContiguousPixels - Returns array of long integers whose length is equal to the number of colors available in the index color mode (value of the N_COLORS property).

Return = *Obj* ->[IDLgrClipboard::]GetContiguousPixels()

IDLgrClipboard::GetDeviceInfo - Returns information that allows IDL applications to make decisions for optimal performance.

Result = *Obj*–>[IDLgrClipboard::]GetDeviceInfo(
[, ALL=*variable*]
[, MAX_TEXTURE_DIMENSIONS=*variable*]
[, MAX_VIEWPORT_DIMENSIONS=*variable*]
[, NAME=*variable*] [, NUM_CPUS=*variable*]
[, VENDOR=*variable*] [, VERSION=*variable*])

IDLgrClipboard::GetFontnames - Returns the list of available fonts that can be used in IDLgrFont objects.

Return = *Obj* -> [IDLgrClipboard::]GetFontnames(
FamilyName [, IDL_FONTS={0 | 1 | 2}]
[, STYLES=*string*])

IDLgrClipboard::GetProperty - Retrieves the value of a property or group of properties for the clipboard buffer.

Obj -> [IDLgrClipboard::]GetProperty [, ALL=*variable*]
[, SCREEN_DIMENSIONS=*variable*]
Note: See also the {Get} properties in IDLgrClipboard::Init

IDLgrClipboard::GetTextDimensions - Retrieves the dimensions of a text object that will be rendered in the clipboard buffer.

Result = *Obj* ->[IDLgrClipboard::]GetTextDimensions(
TextObj [, DESCENT=*variable*] [, PATH=*objref(s)*])

IDLgrClipboard::Init - Initializes the clipboard object.

Obj = OBJ_NEW('IDLgrClipboard')
or
Result = *Obj* -> [IDLgrClipboard::]Init(
[, COLOR_MODEL{Get}={0 | 1}] [, DIMENSIONS{Get, Set}=*[width, height]*] [, GRAPHICS_TREE{Get, Set}=*objref*] [, N_COLORS{Get}=*integer*{2 to 256}] [, PALETTE{Get, Set}=*objref*] [, QUALITY{Get, Set}={0 | 1 | 2}] [, RESOLUTION{Get, Set}=*[xres, yres]*] [, UNITS{Get, Set}={0 | 1 | 2 | 3}] [, UVALUE{Get, Set}=*value*])

IDLgrClipboard::SetProperty - Sets the value of a property or group of properties for the clipboard buffer.

Obj -> [IDLgrClipboard::]SetProperty
Note: See also the {Set} properties in IDLgrClipboard::Init

IDLgrColorbar - Consists of a color-ramp with an optional framing box and annotation axis. Superclasses: IDLgrModel. No subclasses.

IDLgrColorbar::Cleanup - Performs all cleanup on the object.

OBJ_DESTROY,*Obj* or *Obj* -> [IDLgrColorbar::]Cleanup

IDLgrColorbar::ComputeDimensions - Retrieves the dimensions of a colorbar object for the given destination object.

Result = *Obj* ->[IDLgrColorbar::]ComputeDimensions(
DestinationObj [, PATH=*objref(s)*])

IDLgrColorbar::GetProperty - Retrieves the value of a property or group of properties for the colorbar.

Obj -> [IDLgrColorbar::]GetProperty [, ALL=*variable*]
[, PARENT=*variable*] [, XRANGE=*variable*]
[, YRANGE=*variable*] [, ZRANGE=*variable*]
Note: See also the {Get} properties in IDLgrColorbar::Init

IDLgrColorbar::Init - Initializes the colorbar object.

Obj = OBJ_NEW('IDLgrColorbar') or
Result = *Obj* -> [IDLgrColorbar::]Init([*aRed, aGreen, aBlue*] [, BLUE_VALUES{Get, Set}=*vector*]
[, COLOR{Get, Set}=*index or RGB vector*]
[, DIMENSIONS{Get, Set}=*[dx, dy]*]
[, GREEN_VALUES{Get, Set}=*vector*] [, /HIDE{Get, Set}] [, MAJOR{Get, Set}=*integer*] [, MINOR{Get, Set}=*integer*] [NAME{Get, Set}=*string*]
[, PALETTE{Get, Set}=*objref*] [, RED_VALUES{Get, Set}=*vector*] [, SHOW_AXIS{Get, Set}={0 | 1 | 2}]
[, /SHOW_OUTLINE{Get, Set}] [, SUBTICKLEN{Get, Set}=*minor_tick_length/major_tick_length*]
[, THICK{Get, Set}=*points*{1 to 10}] [, /THREED{Get}]
[, TICKFORMAT{Get, Set}=*string*]
[, TICKFRMTDATA{Get, Set}=*value*] [, TICKLEN{Get, Set}=*value*] [, TICKTEXT{Get, Set}=*objref(s)*]
[, TICKVALUES{Get, Set}=*vector*] [, TITLE{Get, Set}=*objref*] [, UVALUE{Get, Set}=*value*]
[, XCOORD_CONV{Get, Set}=*vector*]
[, YCOORD_CONV{Get, Set}=*vector*]
[, ZCOORD_CONV{Get, Set}=*vector*])

IDLgrColorbar::SetProperty - Sets the value of a property or group of properties for the colorbar.

Obj -> [IDLgrColorbar::]SetProperty
Note: See also the {Set} properties in IDLgrColorbar::Init

IDLgrContour - Draws a contour plot from data stored in a rectangular array or from a set of unstructured points. No superclasses. No subclasses.

IDLgrContour::Cleanup - Performs all cleanup on the object.

OBJ_DESTROY, *Obj* or *Obj* -> [IDLgrContour::]Cleanup

IDLgrContour::GetCTM - Returns the 4 x 4 graphics transform matrix from the current object

Result = *Obj* -> [IDLgrContour::]GetCTM(
[, DESTINATION=*objref*] [, PATH=*objref(s)*]
[, TOP=*objref*])

IDLgrContour::GetProperty - Retrieves the value of a property or group of properties for the contour.

Obj -> [IDLgrContour::]GetProperty [, ALL=*variable*]
[, GEOM=*variable*] [, PARENT=*variable*]
[, XRANGE=*variable*] [, YRANGE=*variable*]
[, ZRANGE=*variable*]
Note: See also the {Get} properties in IDLgrContour::Init

IDLgrContour::Init - Initializes the contour object.

Obj = OBJ_NEW('IDLgrContour') or
Result = *Obj* -> [IDLgrContour::]Init([*Values*]

[, ANISOTROPY{Get, Set}=*[x, y, z]*] [, C_COLOR{Get, Set}=*vector*] [, C_FILL_PATTERN{Get, Set}=*array of IDLgrPattern objects*] [, C_LINESTYLE{Get, Set}=*array of linestyles*] [, C_THICK{Get, Set}=*array of line thicknesses*] [, C_VALUE{Get, Set}=*vector*] [, COLOR{Get, Set}=*index or RGB vector*] [, DATA_VALUES{Get, Set}=*vector or 2D array*] [, /DOWNHILL{Get, Set}] [, /FILL{Get, Set}] [, GEOMX{Set}=*vector or 2D array*] [, GEOMY{Set}=*vector or 2D array*] [, GEOMZ{Set}=*scalar, vector, or 2D array*] [, /HIDE{Get, Set}] [, MAX_VALUE{Get, Set}=*value*] [, MIN_VALUE{Get, Set}=*value*] [, NAME{Get, Set}=*string*] [, N_LEVELS{Get, Set}=*value*] [, PALETTE{Get, Set}=*objref*] [, /PLANAR{Get, Set}] [, POLYGONS{Get, Set}=*array of polygon descriptions*] [, SHADE_RANGE{Get, Set}=*[min, max]*] [, SHADING{Get, Set}={0 |1}] [, TICKINTERVAL{Get, Set}=*value*] [, TICKLEN{Get, Set}=*value*] [, UVALUE{Get, Set}=*value*] [, XCOORD_CONV{Get, Set}=*vector*] [, YCOORD_CONV{Get, Set}=*vector*] [, ZCOORD_CONV{Get, Set}=*vector*])

IDLgrContour::SetProperty - Sets the value of a property or group of properties for the contour.

Obj -> [IDLgrContour::]SetProperty

Note: See also the {Set} properties in IDLgrContour::Init

IDLgrFont - Represents a typeface, style, weight, and point size that may be associated with text objects. No superclasses. No subclasses.

IDLgrFont::Cleanup - Performs all cleanup on the object.

OBJ_DESTROY, *Obj* or *Obj* -> [IDLgrFont::]Cleanup

IDLgrFont::GetProperty - Retrieves the value of a property or group of properties for the font.

Obj -> [IDLgrFont:]GetProperty [, ALL=*variable*]

Note: See also the {Get} properties in IDLgrFont::Init

IDLgrFont::Init - Initializes the font object.

Obj = OBJ_NEW('IDLgrFont' [, *Fontname*]) or *Result* = *Obj* -> [IDLgrFont::]Init(*[Fontname]* [, NAME{Get, Set}=*string*] [, SIZE{Get, Set}=*points*] [, SUBSTITUTE{Get, Set}={ 'Helvetica' | 'Courier' | 'Times' | 'Symbol' | 'Hershey'}] [, THICK{Get, Set}=*points*{1 to 10}] [, UVALUE{Get, Set}=*value*])

IDLgrFont::SetProperty - Sets the value of a property or group of properties for the font.

Obj -> [IDLgrFont:]SetProperty

Note: See also the {Set} properties in IDLgrFont::Init

IDLgrImage - Represents a mapping from a 2D array of data values to a 2D array of pixel colors, resulting in a flat 2D-scaled version of the image, drawn at Z = 0. No superclasses. No subclasses.

IDLgrImage::Cleanup - Performs all cleanup on the object.

OBJ_DESTROY, *Obj* or *Obj* -> [IDLgrImage::]Cleanup

IDLgrImage::GetCTM - Returns the 4 x 4 graphics transform matrix from the current object.

Result = *Obj* -> [IDLgrImage::]GetCTM([, DESTINATION=*objref*] [, PATH=*objref(s)*] [, TOP=*objref to IDLgrModel object*])

IDLgrImage::GetProperty - Retrieves the value of the property or group of properties for the image.

Obj -> [IDLgrImage::]GetProperty [, ALL=*variable*] [, PARENT=*variable*] [, XRANGE=*variable*] [, YRANGE=*variable*] [, ZRANGE=*variable*]

Note: See also the {Get} properties in IDLgrImage::Init

IDLgrImage::Init - Initializes the image object.

Obj = OBJ_NEW('IDLgrImage' [, *ImageData*]) or *Result* = *Obj* -> [IDLgrImage::]Init([*ImageData*] [, BLEND_FUNCTION{Get, Set}=*vector*] [, CHANNEL{Get, Set}=*hexadecimal bitmask*] [, DATA{Get, Set}=*nxm*, *2xnxm*, *3xnxm*, or *4xnxm array of image data*] [, DIMENSIONS{Get, Set}=*[width, height]*] [, /GREYSCALE{Get, Set}] [, /HIDE{Get, Set}] [, INTERLEAVE{Get, Set}={0 | 1 | 2}] [, /INTERPOLATE{Get, Set}] [LOCATION{Get, Set}=*[x, y] or [x, y, z]*] [, NAME{Get, Set}=*string*] [, /NO_COPY{Get, Set}] [, /ORDER{Get, Set}] [, PALETTE{Get, Set}=*objref*] [, /RESET_DATA{Set}] [, SHARE_DATA{Set}=*objref*] [, SUB_RECT{Get, Set}=*[x, y, xdim, ydim]*] [, UVALUE{Get, Set}=*value*] [, XCOORD_CONV{Get, Set}=*vector*] [YCOORD_CONV{Get, Set}=*vector*] [, ZCOORD_CONV{Get, Set}=*vector*])

IDLgrImage::SetProperty - Sets the value of the property or group of properties for the image.

Obj -> [IDLgrImage::]SetProperty

Note: See also the {Set} properties in IDLgrImage::Init

IDLgrLegend - Provides a simple interface for displaying a legend. Superclass: IDLgrModel. No subclasses.

IDLgrLegend::Cleanup - Performs all cleanup on the object.

OBJ_DESTROY, *Obj* or *Obj* -> [IDLgrLegend::]Cleanup

IDLgrLegend::ComputeDimensions - Retrieves the dimensions of a legend object for the given destination object.

Result = *Obj* ->[IDLgrLegend::]ComputeDimensions(*DestinationObj* [, PATH=*objref(s)*])

IDLgrLegend::GetProperty - Retrieves the value of a property or group of properties for the legend.

Obj -> [IDLgrLegend::]GetProperty [, ALL=*variable*] [, PARENT=*variable*] [, XRANGE=*variable*] [, YRANGE=*variable*] [, ZRANGE=*variable*]

Note: See also the {Get} properties in IDLgrLegend::Init

IDLgrLegend::Init - Initializes the legend object.

Obj = OBJ_NEW('IDLgrLegend') or *Result* = *Obj* -> [IDLgrLegend::]Init([*aItemNames*] [, BORDER_GAP{Get, Set}=*value*] [, COLUMNS{Get, Set}=*integer*] [, FILL_COLOR{Get, Set}=*index or RGB vector*] [, FONT{Get, Set}=*objref*] [, GAP{Get,

Set}=*value*] [, GLYPH_WIDTH{Get, Set}=*value*] [, /HIDE{Get, Set}] [, ITEM_COLOR{Get, Set}=*array of colors*] [, ITEM_LINESTYLE{Get, Set}=*int array*] [, ITEM_NAME{Get, Set}=*string array*] [, ITEM_OBJECT{Get, Set}=*array of objrefs of type IDLgrSymbol or IDLgrPattern*] [, ITEM_THICK{Get, Set}=*int array*] [, ITEM_TYPE{Get, Set}=*int array*{each element 0 or 1}] [, NAME{Get, Set}=*string*] [, OUTLINE_COLOR{Get, Set}=*index or RGB vector*] [, OUTLINE_THICK{Get, Set}=*integer*] [, /SHOW_FILL{Get, Set}] [, /SHOW_OUTLINE{Get, Set}] [, TEXT_COLOR{Get, Set}=*index or RGB vector*] [, TITLE{Get, Set}=*objref*] [, UVALUE{Get, Set}=*value*] [, XCOORD_CONV{Get, Set}=*vector*] [, YCOORD_CONV{Get, Set}=*vector*] [, ZCOORD_CONV{Get, Set}=*vector*])

IDLgrLegend::SetProperty - Sets the value of a property or group of properties for the legend.

Obj -> [IDLgrLegend::]SetProperty [, RECOMPUTE={0 | 1}{0 prevents recompute, 1 is the default}]

Note: See also the {Set} properties in IDLgrLegend::Init

IDLgrLight - A light object represents a source of illumination for 3D graphic objects. No superclasses. No subclasses.

IDLgrLight::Cleanup - Performs all cleanup on the object.

OBJ_DESTROY, *Obj* or *Obj* -> [IDLgrLight::]Cleanup

IDLgrLight::GetCTM - Returns the 4 x 4 graphics transform matrix from the current object.

Result = *Obj* -> [IDLgrLight::]GetCTM([, DESTINATION=*objref*] [, PATH=*objref(s)*] [, TOP=*objref to IDLgrModel object*])

IDLgrLight::GetProperty - Retrieves the value of a property or group of properties for the light.

Obj -> [IDLgrLight::]GetProperty [, ALL=*variable*] [, PARENT=*variable*]

Note: See also the {Get} properties in IDLgrLight::Init

IDLgrLight::Init - Initializes the light object.

Obj = OBJ_NEW('IDLgrLight') or *Result* = *Obj* -> [IDLgrLight::]Init([, ATTENUATION{Get, Set}=*[constant, linear, quadratic]*] [, COLOR{Get, Set}=*[R, G, B]*] [, CONEANGLE{Get, Set}=*degrees*] [, DIRECTION{Get, Set}=*3-element vector*] [, FOCUS{Get, Set}=*value*] [, /HIDE{Get, Set}] [, INTENSITY{Get, Set}=*value*{0.0 to 1.0}] [, LOCATION{Get, Set}=*[x, y, z]*] [, NAME{Get, Set}=*string*] [, TYPE{Get, Set}={0 | 1 | 2 | 3}] [, UVALUE{Get, Set}=*value*] [, XCOORD_CONV{Get, Set}=*vector*] [, YCOORD_CONV{Get, Set}=*vector*] [, ZCOORD_CONV{Get, Set}=*vector*])

IDLgrLight::SetProperty - Sets the value of a property or group of properties for the light.

Obj -> [IDLgrLight::]SetProperty

Note: See also the {Set} properties in IDLgrLight::Init

IDLgrModel - Represents a graphical item or group of items that can be transformed (rotated, scaled, and/or translated). Superclass: IDL_Container. The following classes are subclassed from this class: IDLgrColorbar, IDLgrLegend.

IDLgrModel::Add - Adds a child to this Model.

Obj -> [IDLgrModel::]Add, *Object* [, /ALIAS] [, POSITION=*index*]

IDLgrModel::Cleanup - Performs all cleanup on the object.

OBJ_DESTROY, *Obj* or *Obj* -> [IDLgrModel::]Cleanup

IDLgrModel::Draw - Draws the specified picture to the specified graphics destination. *This method is provided for purposes of sub-classing only, and is intended to be called only from the Draw method of a subclass of IDLgrModel.*

Obj -> [IDLgrModel::]Draw, *Destination, Picture*

IDLgrModel::GetByName - Finds contained objects by name and returns the object reference to the named object.

Result = *Obj* -> [IDLgrModel::]GetByName(*Name*)

IDLgrModel::GetCTM - Returns the 4 x 4 graphics transform matrix from the current object

Result = *Obj* -> [IDLgrModel::]GetCTM([, DESTINATION=*objref*] [, PATH=*objref(s)*] [, TOP=*objref to IDLgrModel object*])

IDLgrModel::GetProperty - Retrieves the value of a property or group of properties for the model.

Obj -> [IDLgrModel::]GetProperty [, ALL=*variable*] [, PARENT=*variable*]

Note: See also the {Get} properties in IDLgrModel::Init

IDLgrModel::Init - Initializes the model object.

Obj = OBJ_NEW('IDLgrModel') or *Result* = *Obj* -> [IDLgrModel::]Init([, /HIDE{Get, Set}] [, LIGHTING{Get, Set}={0 | 1 | 2}] [, NAME{Get, Set}=*string*] [, /SELECT_TARGET{Get, Set}] [, TRANSFORM{Get, Set}=*4x4 transformation matrix*] [, UVALUE{Get, Set}=*value*])

IDLgrModel::Reset - Sets the current transform matrix for the model object to the identity matrix.

Obj -> [IDLgrModel::]Reset

IDLgrModel::Rotate - Rotates the model about the specified axis by the specified angle.

Obj -> [IDLgrModel::]Rotate, *Axis, Angle* [, /PREMULTIPLY]

IDLgrModel::Scale - Scales model by the specified scaling factors.

Obj -> [IDLgrModel::]Scale, *Sx, Sy, Sz* [, /PREMULTIPLY]

IDLgrModel::SetProperty - Sets the value of a property or group of properties for the model.

Obj -> [IDLgrModel::]SetProperty

Note: See also the {Set} properties in IDLgrModel::Init

IDLgrModel::Translate - Translates the model by the specified translation offsets.

Obj -> [IDLgrModel::]Translate, *Tx, Ty, Tz*

[, /PREMULTIPLY]

IDLgrMPEG - Creates an MPEG movie file from an array of image frames. No superclasses. No subclasses.

IDLgrMPEG::Cleanup - Performs all cleanup on the object.

OBJ_DESTROY, *Obj* or *Obj* -> [IDLgrMPEG::]Cleanup

IDLgrMPEG::GetProperty - Retrieves the value of a property or group of properties for the MPEG object.

Obj -> [IDLgrMPEG::]GetProperty [, ALL=*variable*]

Note: See also the {Get} properties in IDLgrMPEG::Init

IDLgrMPEG::Init - Initializes the MPEG object.

Obj = OBJ_NEW('IDLgrMPEG') or
Result = *Obj* -> [IDLgrMPEG::]Init(
[, DIMENSIONS{Get, Set}=*2-element array*]
[, FILENAME{Get, Set}=*string*] [, FORMAT{Get, Set}={0 | 1}] [, FRAME_RATE{Get, Set} ={1 | 2 | 3 | 4 | 5 | 6 | 7 | 8} [, /INTERLACED{Get, Set}] [, SCALE{Get, Set}=*[xscale, yscale]*] [, /STATISTICS{Get, Set}]
[, TEMP_DIRECTORY=*string*])

IDLgrMPEG::Put - Puts a given image into the MPEG sequence at the specified frame.

Obj -> [IDLgrMPEG::]Put, *Image*[, *Frame*]

IDLgrMPEG::Save - Encodes and saves an MPEG sequence to a file.

Obj -> [IDLgrMPEG::]Save [, FILENAME=*string*]
Macintosh Keywords: [, CREATOR_TYPE=*string*]

IDLgrMPEG::SetProperty - Sets the value of a property or group of properties for the MPEG object.

Obj -> [IDLgrMPEG::]SetProperty

Note: See also the {Set} properties in IDLgrMPEG::Init

IDLgrPalette - Represents a color lookup table that maps indices to red, green, and blue values. No superclasses. No subclasses.

IDLgrPalette::Cleanup - Performs all cleanup on the object.

OBJ_DESTROY, *Obj* or *Obj* -> [IDLgrPalette::]Cleanup

IDLgrPalette::GetRGB - Returns the RGB values contained in the palette at the given index.

Result = *Obj* -> [IDLgrPalette::]GetRGB(*Index*)

IDLgrPalette::GetProperty - Retrieves the value of a property or group of properties for the palette.

Obj -> [IDLgrPalette::]GetProperty [, ALL=*variable*]
[, N_COLORS=*variable*]

Note: See also the {Get} properties in IDLgrPalette::Init

IDLgrPalette::Init - Initializes a palette object.

Obj=OBJ_NEW('IDLgrPalette', [*aRed, aGreen, aBlue*]) or
Result=Obj -> [IDLgrPalette::]Init([*aRed, aGreen, aBlue*]
[, BLUE_VALUES{Get, Set}=*vector*]
[, BOTTOM_STRETCH{Get, Set}=*value*{0 to 100}]
[, GAMMA{Get, Set}=*value*{0.1 to 10.0}]
[, GREEN_VALUES{Get, Set}=*vector*] [, NAME{Get, Set}=*string*] [, RED_VALUES{Get, Set}=*vector*]
[, TOP_STRETCH{Get, Set}=*value*{0 to 100}]
[, UVALUE{Get, Set}=*value*])

IDLgrPalette::LoadCT - Loads one of the IDL predefined color tables into an IDLgrPalette object.

Obj -> [IDLgrPalette::]LoadCT, *TableNum*
[, FILENAME=*colortable filename*]

IDLgrPalette::NearestColor - Returns the index of the color in the palette that best matches the given RGB values.

Result = *Obj* -> [IDLgrPalette::]NearestColor(*Red, Green, Blue*)

IDLgrPalette::SetRGB - Sets the color values at a specified index in the palette to the specified Red, Green and Blue values.

Obj -> [IDLgrPalette::]SetRGB, *Index, Red, Green, Blue*

IDLgrPalette::SetProperty - Sets the value of a property or group of properties for the palette.

Obj -> [IDLgrPalette::]SetProperty

Note: See also the {Set} properties in IDLgrPalette::Init

IDLgrPattern - Describes which pixels are filled and which are left blank when an area is filled. No superclasses. No subclasses.

IDLgrPattern::Cleanup - Performs all cleanup on the object.

OBJ_DESTROY, *Obj* or *Obj* -> [IDLgrPattern::]Cleanup

IDLgrPattern::GetProperty - Retrieves the value of a property or group of properties for the pattern.

Obj -> [IDLgrPattern::]GetProperty [, ALL=*variable*]

Note: See also the {Get} properties in IDLgrPattern::Init

IDLgrPattern::Init - Initializes the pattern object.

Obj = OBJ_NEW('IDLgrPattern' [, *Style*]) or
Result = *Obj* -> [IDLgrPattern::]Init([*Style*]
[, ORIENTATION{Get, Set}=*ccw degrees from horiz*]
[, NAME{Get, Set}=*string*] [, PATTERN{Get, Set}=*32 x 32 bit array*] [, SPACING{Get, Set}=*pixels*]
[, STYLE*{Get, Set}*={0 | 1 | 2}] [, THICK=*integer*{1 to 10}] [, UVALUE{Get, Set}=*value*])

IDLgrPattern:SetProperty - Sets the value of a property or group of properties for the pattern.

Obj -> [IDLgrPattern::]SetProperty

Note: See also the {Set} properties in IDLgrPattern::Init

IDLgrPlot - Creates set of polylines connecting data points in 2D space. No superclasses. No subclasses.

IDLgrPlot::Cleanup - Performs all cleanup on the object.

OBJ_DESTROY, *Obj* or *Obj* -> [IDLgrPlot::]Cleanup

IDLgrPlot::GetCTM - Returns the 4 x 4 graphics transform matrix from the current object upward through the graphics tree.

Result = *Obj* -> [IDLgrPlot::]GetCTM(
[, DESTINATION=*objref*] [, PATH=*objref(s)*]
[, TOP=*objref to IDLgrModel object*])

IDLgrPlot::GetProperty - Retrieves the value of the property or group of properties for the plot.

Obj -> [IDLgrPlot::]GetProperty [, ALL=*variable*]
[, DATA=*variable*] [, PARENT=*variable*]
[, ZRANGE=*variable*]

Note: See also the {Get} properties in IDLgrPlot::Init

IDLgrPlot::Init - Initializes the plot object.

Obj = OBJ_NEW('IDLgrPlot' [, [*X*,] *Y*]) or
Result = *Obj* -> [IDLgrPlot::]Init([[*X*,] *Y*] [, COLOR{Get, Set}=*index or RGB vector* | , VERT_COLORS{Get, Set}=*vector*] [, DATAX {Set}=*vector*] [, DATAY{Set}=*vector*] [, /HIDE{Get, Set}] [, /HISTOGRAM{Get, Set}] [, LINESTYLE{Get, Set}=*integer or two-element vector*] [, MAX_VALUE{Get, Set}=*value*] [, MIN_VALUE{Get, Set}=*value*] [, NAME{Get, Set}=*string*] [, NSUM{Get, Set}=*value*] [, PALETTE{Get, Set}=*objref*] [, /POLAR{Get, Set}] [, /RESET_DATA{Set}] [, SHARE_DATA{Set}=*objref*] [, SYMBOL{Get, Set}=*objref(s)*] [, THICK{Get, Set}=*points*{1 to 10}] [, /USE_ZVALUE] [, UVALUE{Get, Set}=*value*] [, XCOORD_CONV{Get, Set}=*vector*] [, XRANGE{Get, Set}=*[xmin, xmax]*] [, YCOORD_CONV{Get, Set}=*vector*] [, YRANGE{Get, Set}=*[ymin, ymax]*] [, ZCOORD_CONV{Get, Set}=*vector*] [, ZVALUE{Get, Set}=*value*])

IDLgrPlot::SetProperty - Sets the value of the property or group of properties for the plot.

Obj -> [IDLgrPlot::]SetProperty

Note: See also the {Set} properties in IDLgrPlot::Init

IDLgrPolygon - Represents one or more polygons that share a set of vertices and rendering attributes. No superclasses. No subclasses.

IDLgrPolygon::Cleanup - Performs all cleanup on the object.

OBJ_DESTROY, *Obj* or *Obj* -> [IDLgrPolygon::]Cleanup

IDLgrPolygon::GetCTM - Returns the 4 x 4 graphics transform matrix from the current object upward through the graphics tree.

Result = *Obj* -> [IDLgrPolygon::]GetCTM([, DESTINATION=*objref*] [, PATH=*objref(s)*] [, TOP=*objref to IDLgrModel object*])

IDLgrPolygon::GetProperty - Retrieves the value of the property or group of properties for the polygons.

Obj -> [IDLgrPolygon::]GetProperty [, ALL=*variable*] [, PARENT=*variable*] [, XRANGE=*variable*] [, YRANGE=*variable*] [, ZRANGE=*variable*]

Note: See also the {Get} properties in IDLgrPolygon::Init

IDLgrPolygon::Init - Initializes the polygons object.

Obj = OBJ_NEW('IDLgrPolygon' [, *X* [, *Y*[, *Z*]]]) or
Result = *Obj* -> [IDLgrPolygon::]Init([*X*, [*Y*, [*Z*]]] [, BOTTOM{Get, Set}=*index or RGB vector*] [, COLOR{Get, Set}=*index or RGB vector* | , VERT_COLORS{Get, Set}=*vector*] [, DATA{Get, Set}=*array*] [, FILL_PATTERN{Get, Set}=*objref to IDLgrPattern object*] [, /HIDDEN_LINES] [, /HIDE{Get, Set}] [, LINESTYLE{Get, Set}=*value*] [, NAME{Get, Set}=*string*] [, NORMALS{Get, Set}=*array*] [, PALETTE=*objref*] [, POLYGONS{Get, Set}=*array of polygon descriptions*] [, REJECT{Get, Set}={0 | 1 | 2}] [, /RESET_DATA{Set}] [, SHADE_RANGE{Get, Set}=*array*] [, SHADING{Get, Set}={0 | 1}] [, SHARE_DATA{Set}=*objref*] [, STYLE{Get, Set}={0 | 1 | 2}] [, TEXTURE_COORD{Get, Set}=*array*] [, /TEXTURE_INTERP{Get, Set}] [, TEXTURE_MAP*{Get, Set}*=*objref to IDLgrImage object*] [, THICK{Get, Set}=*points*{1 to 10}] [, XCOORD_CONV{Get, Set}=*vector*] [, YCOORD_CONV{Get, Set}=*vector*] [, ZCOORD_CONV{Get, Set}=*vector*] [, ZERO_OPACITY_SKIP{Get, Set}={0 | 1}])

IDLgrPolygon::SetProperty - Sets the value of the property or group of properties for the polygons.

Obj -> [IDLgrPolygon::]SetProperty

Note: See also the {Set} properties in IDLgrPolygon::Init

IDLgrPolyline - Represents one or more polylines that share a set of vertices and rendering attributes. No superclasses. No subclasses.

IDLgrPolyline::Cleanup - Performs all cleanup on the object.

OBJ_DESTROY, *Obj* or *Obj* -> [IDLgrPolyline::]Cleanup

IDLgrPolyline::GetCTM - Returns the 4 x 4 graphics transform matrix from the current object upward through the graphics tree.

Result = *Obj* -> [IDLgrPolyline::]GetCTM([, DESTINATION=*objref*] [, PATH=*objref(s)*] [, TOP=*objref to IDLgrModel object*])

IDLgrPolyline::GetProperty - Retrieves the value of a property or group of properties for the polylines.

Obj -> [IDLgrPolyline::]GetProperty [, ALL=*variable*] [, PARENT=*variable*] [, XRANGE=*variable*] [, YRANGE=*variable*] [, ZRANGE=*variable*]

Note: See also the {Get} properties in IDLgrPolyline::Init

IDLgrPolyline::Init - Initializes the polylines object.

Obj = OBJ_NEW('IDLgrPolyline' [, *X* [, *Y*[, *Z*]]]) or
Result = *Obj* -> [IDLgrPolyline::]Init([*X*, [*Y*, [*Z*]]] [, COLOR{Get, Set}=*index or RGB vector* | , VERT_COLORS{Get, Set}=*vector*] [, DATA{Get, Set}=*array*] [, /HIDE{Get, Set}] [, LINESTYLE{Get, Set}=*value*] [, NAME{Get, Set}=*string*] [, PALETTE{Get, Set}=*objref*] [, POLYLINES{Get, Set}=*array of polyline descriptions*] [, /RESET_DATA{Set}] [, SHADING{Get, Set}={0 | 1}] [, SHARE_DATA{Set}=*objref*] [, SYMBOL{Get, Set}=*objref(s)*] [, THICK{Get, Set}=*points*{1 to 10}] [, UVALUE{Get, Set}=*value*] [, XCOORD_CONV{Get, Set}=*vector*] [, YCOORD_CONV{Get, Set}=*vector*] [, ZCOORD_CONV{Get, Set}=*vector*])

IDLgrPolyline::SetProperty - Sets the value of a property or group of properties for the polylines.

Obj -> [IDLgrPolylines::]SetProperty

Note: See also the {Set} properties in IDLgrPolyline::Init

IDLgrPrinter - Represents a hardcopy graphics destination. No superclasses. No subclasses.

IDLgrPrinter::Cleanup - Performs all cleanup on the object.

OBJ_DESTROY, *Obj* or *Obj* -> [IDLgrPrinter::]Cleanup

IDLgrPrinter::Draw - Draws a picture to this graphics destination.

Obj -> [IDLgrPrinter::]Draw [, *Picture*]
[, VECTOR={ 0 | 1 }]

IDLgrPrinter::GetContiguousPixels - Returns an array of long integers whose length is equal to the number of colors available in the index color mode (value of N_COLORS property).

Return = *Obj* -> [IDLgrPrinter::]GetContiguousPixels( )

IDLgrPrinter::GetFontnames - Returns the list of available fonts that can be used in IDLgrFont objects.

Return = *Obj* -> [IDLgrPrinter::]GetFontnames(
FamilyName [, IDL_FONTS={0 | 1 | 2}]
[, STYLES=*string*])

IDLgrPrinter::GetProperty - Retrieves the value of a property or group of properties for the printer.

Obj -> [IDLgrPrinter::]GetProperty [, ALL=*variable*]
[, DIMENSIONS=*variable*] [, NAME=*string*]
[, RESOLUTION=*variable*]
Note: See also the {Get} properties in IDLgrPrinter::Init

IDLgrPrinter::GetTextDimensions - Retrieves the dimensions of a text object that will be rendered on the printer.

Result = *Obj* ->[IDLgrPrinter::]GetTextDimensions(
TextObj [, DESCENT=*variable*] [, PATH=*objref(s)*])

IDLgrPrinter::Init - Initializes the printer object.

Obj = OBJ_NEW('IDLgrPrinter') or
Result = *Obj* -> [IDLgrPrinter::]Init(
[, COLOR_MODEL{Get}=0 | 1}]
[, GRAPHICS_TREE{Get, Set}=*objref of type IDLgrScene, IDLgrViewgroup, or IDLgrView*]
[, /LANDSCAPE{Get, Set}]
[, N_COLORS{Get}=*integer*{2 to 256}]
[, N_COPIES{Get, Set}=*integer*] [, PALETTE{Get, Set}=*objref*] [, PRINT_QUALITY{Get, Set}={0 | 1 | 2}]
[, QUALITY{Get, Set}={0 | 1 | 2}] [, UNITS{Get, Set}={0 | 1 | 2 | 3}] [, UVALUE{Get, Set}=*value*])

IDLgrPrinter::NewDocument - Closes current document (page or group of pages), which causes pending output to be sent to the printer, finishing the printer job.

Obj -> [IDLgrPrinter::]NewDocument

IDLgrPrinter::NewPage - Issues new page command to printer.

Obj -> [IDLgrPrinter::]NewPage

IDLgrPrinter::SetProperty - Sets the value of a property or group of properties for the printer.

Obj -> [IDLgrPrinter::]SetProperty
Note: See also the {Set} properties in IDLgrPrinter::Init

IDLgrROI - Object graphics representation of a region of interest. Subclass of IDLanROI.

IDLgrROI::Cleanup - Performs all cleanup for the object.

Obj–>[IDLgrROI::]Cleanup
or OBJ_DESTROY, *Obj*

IDLgrROI::GetProperty - Retrieves the value of a property or group of properties for the Object Graphics region.

Obj–>[IDLgrROI::]GetProperty [, ALL=*variable*]

IDLgrROI::Init - Initializes an Object Graphics region of interest.

Result = IDLgrROI::Init([*X*[, *Y*[, *Z*]]]
[, COLOR{Get, Set}=*vector*] [, /HIDE{Get, Set}]
[, LINESTYLE{Get, Set}=*value*]
[, NAME{Get, Set}=*string*]
[, PALETTE{Get, Set}=*objref*]
[, STYLE{Get, Set}={ 0 | 1 | 2 }]
[, SYMBOL{Get, Set}=*objref*]
[, THICK{Get, Set}=*points* {1.0 to 10.0}]
[, UVALUE{Get, Set}=*uvalue*]
[, XCOORD_CONV{Get, Set}=*[s_0, s_1]*]
[, YCOORD_CONV{Get, Set}=*[s_0, s_1]*]
[, ZCOORD_CONV{Get, Set}=*[s_0, s_1]*])
or
Obj = OBJ_NEW('IDLgrROI'[, *X*[, *Y*[, *Z*]]])

IDLgrROI::PickVertex - Picks a vertex of the region that, when projected onto the given destination device, is nearest to the given 2D device coordinate.

Result = *Obj*–>[IDLgrROI::]PickVertex(*Dest*, *View*, *Point*
[, PATH=*objref*])

IDLgrROI::SetProperty - Sets the value of a property or group of properties for the Object Graphics region.

Obj–>[IDLgrROI::]SetProperty

IDLgrROIGroup - Object Graphics representation of a group of regions of interest. Subclass of IDLanROIGroup.

IDLgrROIGroup::Add - Adds a region to the region group.

Obj–>[IDLgrROIGroup::]Add, *ROI*

IDLgrROIGroup::Cleanup - Performs all cleanup for the object.

Obj–>[IDLgrROIGroup::]Cleanup
or OBJ_DESTROY, *Obj*

IDLgrROIGroup::Init - Initializes an Object Graphics region of interest group object.

Result = IDLgrROIGroup::Init()
or *Obj* = OBJ_NEW('IDLgrROIGroup')

IDLgrROIGroup::PickRegion - Picks a region within the group that, when projected onto the given destination device, is nearest to the given 2D device coordinate.

Result = *Obj*–>[IDLgrROIGroup::]PickRegion(*Dest*, *View*, *Point* [, PATH=*objref*])

IDLgrScene - Represents the entire scene to be drawn and serves as a container of IDLgrView or IDLgrViewgroup objects. Superclass: IDL_Container. No subclasses.

IDLgrScene::Add - Verifies that the added item is an instance of an IDLgrView or IDLgrViewgroup object.

Obj -> [IDLgrScene::]Add, *View* [, POSITION=*index*]

IDLgrScene::Cleanup - Performs all cleanup on the object.

OBJ_DESTROY, *Obj* or *Obj* -> [IDLgrScene::]Cleanup

IDLgrScene::GetByName - Finds contained objects by name and returns the object reference to the named object.

Result = *Obj* -> [IDLgrScene::]GetByName(*Name*)

IDLgrScene::GetProperty - Retrieves the value of a property or group of properties for the contour.

Obj -> [IDLgrScene::]GetProperty [, ALL=*variable*]

Note: See also the {Get} properties in IDLgrScene::Init

IDLgrScene::Init - Initializes the scene object.

Obj = OBJ_NEW('IDLgrScene') or
Result = *Obj* -> [IDLgrScene::]Init([, COLOR{Get, Set}=*index or RGB vector*] [, /HIDE{Get, Set}] [, NAME{Get, Set}=*string*] [, /TRANSPARENT{Get, Set}] [, UVALUE{Get, Set}=*value*])

IDLgrScene::SetProperty - Sets the value of one or more properties for the scene.

Obj -> [IDLgrScene::]SetProperty

Note: See also the {Set} properties in IDLgrScene::Init

IDLgrSurface - A shaded or vector representation of a mesh grid. No superclasses. No subclasses.

IDLgrSurface::Cleanup - Performs all cleanup on the object.

OBJ_DESTROY, *Obj* or *Obj* -> [IDLgrSurface::]Cleanup

IDLgrSurface::GetCTM - Returns the 4 x 4 graphics transform matrix from the current object upward through the graphics tree.

Result = *Obj* -> [IDLgrSurface::]GetCTM([, DESTINATION=*objref*] [, PATH=*objref(s)*] [, TOP=*objref to IDLgrModel object*])

IDLgrSurface::GetProperty - Retrieves the value of a property or group of properties for the surface.

Obj -> [IDLgrSurface::]GetProperty [, ALL=*variable*] [, DATA=*variable*] [, PARENT=*variable*] [, XRANGE=*variable*] [, YRANGE=*variable*] [, ZRANGE=*variable*]

Note: See also the {Get} properties in IDLgrSurface::Init

IDLgrSurface::Init - Initializes the surface object.

Obj = OBJ_NEW('IDLgrSurface' [, *Z* [, *X*, *Y*]]) or
Result = *Obj* -> [IDLgrSurface::]Init([*Z*, [*X*, *Y*]] [, BOTTOM{Get, Set}=*index or RGB vector*] [, COLOR{Get, Set}=*index or RGB vector*] [, DATAX{Set}=*vector or 2D array*] [, DATAY{Set}=*vector or 2D array*] [, DATAZ{Set}=*2D array*] [, /EXTENDED_LEGO{Get, Set}] [, /HIDDEN_LINES{Get, Set}] [, /HIDE{Get, Set}] [, LINESTYLE{Get, Set}=*value*] [, MAX_VALUE{Get, Set}=*value*] [, MIN_VALUE{Get, Set}=*value*] [, NAME{Get, Set}=*string*] [, PALETTE{Get, Set}=*objref*] [, /RESET_DATA{Set}] [, SHADE_RANGE{Get, Set}=*[index of darkest pixel, index of brightest pixel]*] [, SHADING{Get, Set}={0 | 1}] [, SHARE_DATA{Set}=*objref*] [, /SHOW_SKIRT{Get, Set}] [, SKIRT{Get, Set}=*Z value*] [, STYLE{Get, Set}={0 | 1 | 2 | 3 | 4 | 5 | 6}] [, TEXTURE_COORD{Get, Set}=*array*] [, /TEXTURE_INTERP{Get, Set}] [, TEXTURE_MAP{Get, Set}=*objref to IDLgrImage*] [, THICK{Get, Set}=*points*{1 to 10}] [, UVALUE{Get, Set}=*value*] [, /USE_TRIANGLES{Get, Set}] [, VERT_COLORS{Get, Set}=*vector*] [, XCOORD_CONV{Get, Set}=*vector*] [, YCOORD_CONV{Get, Set}=*vector*] [, ZCOORD_CONV{Get, Set}=*vector*] [, ZERO_OPACITY_SKIP{Get, Set}={0 | 1}])

IDLgrSurface::SetProperty - Sets the value of a property or group of properties for the surface.

Obj -> [IDLgrSurface::]SetProperty

Note: See also the {Set} properties in IDLgrSurface::Init

IDLgrSymbol - Represents a graphical element that is plotted relative to a particular position. No superclasses. No subclasses.

IDLgrSymbol::Cleanup - Performs all cleanup on the object.

OBJ_DESTROY, *Obj* or *Obj* -> [IDLgrSymbol::]Cleanup

IDLgrSymbol::GetProperty - Retrieves the value of a property or group of properties for the symbol.

Obj -> [IDLgrSymbol::]GetProperty [, ALL=*variable*]

Note: See also the {Get} properties in IDLgrSymbol::Init

IDLgrSymbol::Init - Initializes the plot symbol.

Obj = OBJ_NEW('IDLgrSymbol' [*Data*]) or
Result = *Obj* -> [IDLgrSymbol::]Init([*Data*] [, COLOR{Get, Set}=*index or RGB vector*] [, DATA{Get, Set}=*integer or objref*] [, NAME{Get, Set}=*string*] [, SIZE{Get, Set}=*vector*] [, THICK{Get, Set}=*points*{1 to 10}] [, UVALUE{Get, Set}=*value*])

IDLgrSymbol::SetProperty - Sets the value of a property or group of properties for the symbol.

Obj -> [IDLgrSymbol::]SetProperty

Note: See also the {Set} properties in IDLgrSymbol::Init

IDLgrTessellator - Converts a simple concave polygon (or a simple polygon with “holes”) into a number of simple convex polygons (general triangles). No superclasses. No subclasses.

IDLgrTessellator::AddPolygon - Adds a polygon to the tessellator object.

Obj -> [IDLgrTessellator::]AddPolygon, *X* [, *Y*[, *Z*]] [, POLYGON{Get, Set}=*array of polygon descriptions*] [, /INTERIOR]

IDLgrTessellator::Cleanup - Performs all cleanup on the object.

OBJ_DESTROY,*Obj* or
Obj -> [IDLgrTessellator::]Cleanup

IDLgrTessellator::Init - Initializes the tessellator object.

Obj = OBJ_NEW('IDLgrTesselator') or
Result = *Obj* -> [IDLgrTessellator::]Init( )

IDLgrTessellator::Reset - Resets the object’s internal state.

Obj -> [IDLgrTessellator::]Reset

IDLgrTessellator::Tessellate - Performs the actual tessellation.

Result = *Obj* -> [IDLgrTessellator::]Tessellate(*Vertices*, *Poly* [, /QUIET])

IDLgrText - Represents one or more text strings that share common rendering attributes. No superclasses. No subclasses.

IDLgrText::Cleanup - Performs all cleanup on the object.

OBJ_DESTROY, *Obj* or *Obj* -> [IDLgrText::]Cleanup

IDLgrText::GetCTM - Returns the 4 x 4 graphics transform matrix from the current object upward through the graphics tree.

Result = *Obj* -> [IDLgrText::]GetCTM([, DESTINATION=*objref*] [, PATH=*objref(s)*] [, TOP=*objref to IDLgrModel object*])

IDLgrText::GetProperty - Retrieves the value of a property or group of properties for the text.

Obj -> [IDLgrText::]GetProperty [, ALL=*variable*] [, PARENT=*variable*] [, XRANGE=*variable*] [, YRANGE=*variable*] [, ZRANGE=*variable*]

Note: See also the {Get} properties in IDLgrText::Init

IDLgrText::Init - Initializes the text object.

Obj = OBJ_NEW('IDLgrText' [, *String or vector of strings*]) or

Result = *Obj* -> [IDLgrText::]Init([, *String or vector of strings*] [, ALIGNMENT{Get, Set}=*value*{0.0 to 1.0}] [, BASELINE{Get, Set}=*vector*] [, CHAR_DIMENSIONS{Get, Set}=*[width, height]*] [, COLOR{Get, Set}=*index or RGB vector*] [, /ENABLE_FORMATTING{Get, Set}] [, FONT{Get, Set}=*objref*] [, /HIDE{Get, Set}] [, LOCATIONS{Get, Set}=*array*] [, NAME{Get, Set}=*string*] [, /ONGLASS{Get, Set}] [, PALETTE{Get, Set}=*objref*] [, RECOMPUTE_DIMENSIONS{Get, Set}={0 | 1 | 2}] [, STRINGS{Get, Set}=*string or vector of strings*] [, UPDIR{Get, Set}=*vector*] [, UVALUE{Get, Set}=*value*] [, VERTICAL_ALIGNMENT{Get, Set}=*value*{0.0 to 1.0}] [, XCOORD_CONV{Get, Set}=*vector*] [, YCOORD_CONV{Get, Set}=*vector*] [, ZCOORD_CONV{Get, Set}=*vector*])

IDLgrText::SetProperty - Sets the value of a property or group of properties for the text.

Obj -> [IDLgrText::]SetProperty

Note: See also the {Set} properties in IDLgrText::Init

IDLgrView - Represents a rectangular area in which graphics objects are drawn. It is a container for objects of the IDLgrModel class. Superclass: IDL_Container. No subclasses.

IDLgrView::Add - Adds a child to this view.

Obj -> [IDLgrView::]Add, *Model* [, POSITION=*index*]

IDLgrView::Cleanup - Performs all cleanup on the object.

OBJ_DESTROY, *Obj* or *Obj* -> [IDLgrView::]Cleanup

IDLgrView::GetByName - Finds contained objects by name.

Result = *Obj* -> [IDLgrView::]GetByName(*Name*)

IDLgrView::GetProperty - Retrieves the value of the property or group of properties for the view.

Obj -> [IDLgrView::]GetProperty [, ALL=*variable*] [, PARENT=*variable*]

Note: See also the {Get} properties in IDLgrView::Init

IDLgrView::Init - Initializes the view object.

Obj = OBJ_NEW('IDLgrView') or

Result = *Obj* -> [IDLgrView::]Init([, COLOR{Get, Set}=*index or RGB vector*] [, DEPTH_CUE{Get, Set}=*[zbright, zdim]*] [, DIMENSIONS{Get, Set}=*[width, height]*] [, EYE{Get, Set}=*distance*] [, LOCATION{Get, Set}=*[x, y]*] [, PROJECTION{Get, Set}={1 | 2}] [, /TRANSPARENT{Get, Set}] [, UNITS{Get, Set}={0 | 1 | 2 | 3}] [, UVALUE{Get, Set}=*value*] [, VIEWPLANE_RECT{Get, Set}=*[x, y, width, height]*] [, ZCLIP{Get, Set}=*[near, far]*])

IDLgrView::SetProperty - Sets the value of the property or group of properties for the view.

Obj -> [IDLgrView::]SetProperty

Note: See also the {Set} properties in IDLgrView::Init

IDLgrViewgroup - A simple container object that contains one or more IDLgrView objects. An IDLgrScene can contain one or more of these objects. Superclass: IDL_Container. No subclasses.

IDLgrViewgroup::Add - Verifies that the added item is not an instance of the IDLgrScene or IDLgrViewgroup object.

Obj -> [IDLgrViewgroup::]Add, *Object* [, POSITION=*index*]

IDLgrViewgroup::Cleanup - Performs all cleanup on the object.

OBJ_DESTROY,*Obj*

or

Obj -> [IDLgrViewgroup::]Cleanup

IDLgrViewgroup::GetByName - Finds contained objects by name.

Result = *Obj* -> [IDLgrViewgroup::]GetByName(*Name*)

IDLgrViewgroup::GetProperty - Retrieves the value of a property or group of properties for the viewgroup object.

Obj -> [IDLgrViewgroup::]GetProperty [, ALL=*variable*] [, PARENT=*variable*]

Note: See also the {Get} properties in IDLgrViewgroup::Init

IDLgrViewgroup::Init - Initializes the viewgroup object.

Obj = OBJ_NEW('IDLgrViewgroup') or

Result = *Obj* -> [IDLgrViewgroup::]Init([, /HIDE{Get, Set}] [, NAME{Get, Set}=*string*] [, UVALUE{Get, Set}=*value*])

IDLgrViewgroup::SetProperty - Sets the value of a property or group of properties for the viewgroup.

Obj -> [IDLgrViewgroup::]SetProperty

Note: See also the {Set} properties in IDLgrViewgroup::Init

IDLgrVolume - Represents mapping from a 3D array of data to a 3D array of voxel colors, which, when drawn, are projected to two dimensions. No superclasses. No subclasses.

IDLgrVolume::Cleanup - Performs all cleanup on the object.

OBJ_DESTROY, *Obj* or *Obj* -> [IDLgrVolume::]Cleanup

IDLgrVolume::ComputeBounds - Computes the smallest bounding box that contains all voxels whose opacity lookup is greater than a given opacity value.

Obj -> [IDLgrVolume::]ComputeBounds
[, OPACITY=*value*] [, /RESET] [, VOLUMES=*int array*]

IDLgrVolume::GetCTM - Returns the 4 x 4 graphics transform matrix from the current object upward through the graphics tree.

Result = *Obj* -> [IDLgrVolume::]GetCTM(
[, DESTINATION=*objref*] [, PATH=*objref(s)*]
[, TOP=*objref to IDLgrModel object*])

IDLgrVolume::GetProperty - Retrieves the value of a property or group of properties for the volume.

Obj -> [IDLgrVolume::]GetProperty [, ALL=*variable*]
[, PARENT=*variable*] [, VALID_DATA=*variable*]
[, XRANGE=*variable*] [, YRANGE=*variable*]
[, ZRANGE=*variable*]
Note: See also the {Get} properties in IDLgrVolume::Init

IDLgrVolume::Init - Initializes the volume object.

Obj = OBJ_NEW('IDLgrVolume' [, *vol0* [, *vol1* [, *vol2* [, *vol3*]]]]) or
Result = *Obj* -> [IDLgrVolume::]Init([*vol0* [, *vol1* [, *vol2* [, *vol3*]]]] [, AMBIENT{Get, Set}=*RGB vector*] [, BOUNDS{Get, Set}=*[xmin, ymin, zmin, xmax, ymax, zmax]*] [, COMPOSITE_FUNCTION{Get, Set}={0 | 1 | 2 | 3}] [, CUTTING_PLANES{Get, Set}=*array*] [, DATA0{Get, Set}=*[d_x, d_y, d_z]*] [, DATA1{Get, Set}=*[d_x, d_y, d_z]*] [, DATA2{Get, Set}=*[d_x, d_y, d_z]*] [, DATA3{Get, Set}=*[d_x, d_y, d_z]*] [, DEPTH_CUE{Get, Set}=*[zbright, zdim]*] , /HIDE{Get, Set}] [, HINTS{Get, Set}={0 | 1 | 2 | 3}] [, /INTERPOLATE{Get, Set}] [, /LIGHTING_MODEL{Get, Set}] [, NAME{Get, Set}=*string*] [, /NO_COPY{Get, Set}] [, OPACITY_TABLE0{Get, Set}=*256-element byte array*] [, OPACITY_TABLE1{Get, Set}=*256-element byte array*] [, RENDER_STEP{Get, Set}=*[x, y, z]*] [, RGB_TABLE0{Get, Set}=*256 x 3-element byte array*] [, RGB_TABLE1{Get, Set}=*256 x 3-element byte array*] [, /TWO_SIDED{Get, Set}] [, UVALUE{Get, Set}=*value*] [, VOLUME_SELECT{Get, Set}={0 | 1 | 2}] [, XCOORD_CONV{Get, Set}=*vector*] [, YCOORD_CONV{Get, Set}=*vector*] [, /ZBUFFER{Get, Set}] [, ZCOORD_CONV{Get, Set}=*vector*] [, ZERO_OPACITY_SKIP{Get, Set}={0 | 1}])

IDLgrVolume::PickVoxel - Computes the coordinates of the voxel projected to a location specified by the 2D device coordinates point, [x_i, y_i], and the current Z-buffer.

Result = *Obj* -> [IDLgrVolume::]PickVoxel (*Win*, *View*, *Point* [, PATH=*objref(s)*])

IDLgrVolume::SetProperty - Sets the value of a property or group of properties for the volume.

Obj -> [IDLgrVolume::]SetProperty

Note: See also the {Set} properties in IDLgrVolume::Init

IDLgrVRML - Saves the contents of an Object Graphics hierarchy into a VRML 2.0 format file. No superclasses. No subclasses.

IDLgrVRML::Cleanup - Performs all cleanup on the object.

OBJ_DESTROY, *Obj* or *Obj* -> [IDLgrVRML::]Cleanup

IDLgrVRML::Draw - Draws a picture to this graphics destination.

Obj -> [IDLgrVRML::]Draw [, *Picture*]

IDLgrVRML::GetDeviceInfo - Returns information that allows IDL applications to make decisions for optimal performance.

Result = *Obj*–>[IDLgrVRML::]GetDeviceInfo(
[, ALL=*variable*]
[, MAX_TEXTURE_DIMENSIONS=*variable*]
[, MAX_VIEWPORT_DIMENSIONS=*variable*]
[, NAME=*variable*] [, NUM_CPUS=*variable*]
[, VENDOR=*variable*] [, VERSION=*variable*])

IDLgrVRML::GetFontnames - Returns the list of available fonts that can be used in IDLgrFont objects.

Return = *Obj* ->[IDLgrVRML::]GetFontnames(
FamilyName [, IDL_FONTS={0 | 1 | 2}]
[, STYLES=*string*])

IDLgrVRML::GetProperty - Retrieves the value of a property or group of properties for the VRML object.

Obj -> [IDLgrVRML::]GetProperty [, ALL=*variable*]
[, SCREEN_DIMENSIONS=*variable*]
Note: See also the {Get} properties in IDLgrVRML::Init

IDLgrVRML::GetTextDimensions - Retrieves the dimensions of a text object that will be rendered in the clipboard buffer.

Result = *Obj* ->[IDLgrVRML::]GetTextDimensions(
TextObj [, DESCENT=*variable*] [, PATH=*objref(s)*])

IDLgrVRML::Init - Initializes the VRML object.

Obj = OBJ_NEW('IDLgrVRML') or
Result = *Obj* -> [IDLgrVRML::]Init(
[, COLOR_MODEL{Get}={0 | 1}] [, DIMENSIONS{Get, Set}=*[width, height]*] [, FILENAME{Get, Set}=*string*]
[, GRAPHICS_TREE{Get, Set}=*objref*]
[, N_COLORS{Get}=*integer*(2 to 256)] [, PALETTE{Get, Set}=*objref*] [, QUALITY{Get, Set}={0 | 1 | 2}]
[, RESOLUTION{Get, Set}=*[xres, yres]*] [, UNITS{Get, Set}={0 | 1 | 2 | 3}] [, UVALUE{Get, Set}=*value*]
[, WORLDINFO=*string array*] [, WOLRDTITLE=*string*]
)

IDLgrVRML::SetProperty - Sets the value of a property or group of properties for the VRML world.

Obj -> [IDLgrVRML::]SetProperty
Note: See also the {Set} properties in IDLgrVRML::Init

IDLgrWindow - Represents an on-screen area on a display device that serves as a graphics destination. No superclasses. No subclasses.

IDLgrWindow::Cleanup - Performs all cleanup on the object.

OBJ_DESTROY, *Obj* or *Obj* -> [IDLgrWindow::]Cleanup

IDLgrWindow::Draw - Draws the specified scene or view object to this graphics destination.

Obj -> [IDLgrWindow::]Draw [, *Picture*] [, CREATE_INSTANCE={1 | 2}] [, /DRAW_INSTANCE]

IDLgrWindow::Erase - Erases the entire contents of the window.

Obj -> [IDLgrWindow::]Erase [, COLOR=*index or RGB vector*]

IDLgrWindow::GetContiguousPixels - Returns an array of long integers whose length is equal to the number of colors available in the index color mode (value of N_COLORS property).

Return = *Obj* -> [IDLgrWindow::]GetContiguousPixels( )

IDLgrWindow::GetDeviceInfo - Returns information that allows IDL applications to make decisions for optimal performance.

Obj–>[IDLgrWindow::]GetDeviceInfo([, ALL=*variable*] [, MAX_TEXTURE_DIMENSIONS=*variable*] [, MAX_VIEWPORT_DIMENSIONS=*variable*] [, NAME=*variable*] [, NUM_CPUS=*variable*] [, VENDOR=*variable*] [, VERSION=*variable*])

IDLgrWindow::GetFontnames - Returns the list of available fonts that can be used in IDLgrFont objects.

Return = *Obj* -> [IDLgrWindow::]GetFontnames(*FamilyName* [, IDL_FONTS={0 | 1 | 2}] [, STYLES=*string*])

IDLgrWindow::GetProperty - Retrieves the value of a property or group of properties for the window.

Obj -> [IDLgrWindow::]GetProperty [, ALL=*variable*] [, IMAGE_DATA=*variable*] [, RESOLUTION=*variable*] [, SCREEN_DIMENSIONS=*variable*] [, ZBUFFER_DATA=*variable*]

Note: See also the {Get} properties in IDLgrWindow::Init

IDLgrWindow::GetTextDimensions - Retrieves the dimensions of a text object that will be rendered in the window.

Result = *Obj* ->[IDLgrWindow::]GetTextDimensions(*TextObj* [, DESCENT=*variable*] [, PATH=*objref(s)*])

IDLgrWindow::Iconify - Iconifies or de-iconifies the window.

Obj -> [IDLgrWindow::]Iconify, *IconFlag*

IDLgrWindow::Init - Initializes the window object.

Obj = OBJ_NEW('IDLgrWindow') or
Result = *Obj* -> [IDLgrWindow::]Init([, COLOR_MODEL{Get}={0 | 1}] [, DIMENSIONS{Get, Set} =*[width, height]*] [, GRAPHICS_TREE{Get, Set}=*objref of type IDLgrScene, IDLgrViewgroup, or IDLgrView*] [, LOCATION{Get, Set}=*[x, y]*] [, N_COLORS{Get}=*integer*{2 to 256}] [, PALETTE{Get, Set}=*objref*] [, QUALITY{Get, Set}={0 | 1 | 2}] [, RENDERER{Get}={0 | 1}] [, RETAIN{Get}={0 | 1 | 2}] [, TITLE{Get, Set}=*string*] [, UNITS{Get, Set}={0 | 1 | 2 | 3}] [, UVALUE{Get, Set}=*value*])

X Windows Keywords:
[, DISPLAY_NAME{Get}=*string*]

IDLgrWindow::Pickdata - Maps a point in the 2D device space of the window to a point in the 3D data space of an object tree.

Result = *Obj* -> [IDLgrWindow::]Pickdata(*View, Object, Location, XYZLocation* [, PATH=*objref(s)*])

IDLgrWindow::Read - Reads an image from a window.

Result = *Obj* -> [IDLgrWindow::]Read( )

IDLgrWindow::Select - Returns a list of objects selected at a specified location.

Result = *Obj* -> [IDLgrWindow::]Select(*Picture, XY* [, DIMENSIONS=*[width, height]*] [, UNITS={0 | 1 | 2 | 3}])

IDLgrWindow::SetCurrentCursor - Sets the current cursor image to be used while positioned over a drawing area.

Obj -> [IDLgrWindow::]SetCurrentCursor [, *CursorName*] [, IMAGE=*16 x 16 bitmap*] [, MASK=*16 x 16 bitmap*] [, HOTSPOT=*[x, y]*]

X Windows Only Keywords: [, STANDARD=*index*]

IDLgrWindow::SetProperty - Sets the value of a property or group of properties for the window.

Obj -> [IDLgrWindow::]SetProperty

Note: See also the {Set} properties in IDLgrWindow::Init

IDLgrWindow::Show - Exposes or hides a window.

Obj -> [IDLgrWindow::]Show, *Position*

TrackBall - Translates widget events from a draw widget (created with the WIDGET_DRAW function) into transformations that emulate a virtual trackball (for transforming object graphics in three dimensions). No superclasses. No subclasses.

TrackBall::Init - Initializes the TrackBall object.

Obj = OBJ_NEW('TrackBall', *Center*, *Radius*)
or
Result = *Obj* -> [TrackBall::]Init(*Center*, *Radius* [, AXIS={0 | 1 | 2}] [, /CONSTRAIN] [, MOUSE=*bitmask*])

Trackball::Reset - Resets the state of the TrackBall object.

Obj -> [TrackBall::]Reset(*Center, Radius* [, AXIS={0 | 1 | 2}] [, /CONSTRAIN] [, MOUSE=*bitmask*])

TrackBall::Update - Updates the state of the TrackBall object based on the information contained in the input widget event structure.

Result = *Obj* -> [TrackBall::]Update(*sEvent* [, MOUSE=*bitmask*] [, TRANSFORM=*variable*] [, /TRANSLATE])

Statements

Assignment

***variable = expression* -** Assigns a value to a variable.

Flow Control

BEGIN...END - Defines a block of statements.

BEGIN
statements
END

CASE...ENDCASE - Selects one statement for execution, depending on the value of an expression.

CASE *expression* OF
expression: *statement*
...
expression: *statement*
[ELSE: *statement*]
ENDCASE

COMMON - Creates a common block.

COMMON *Block_Name*, $Variable_1$, ..., $Variable_n$

FOR - Executes one or more statements repeatedly, while incrementing or decrementing a variable with each repetition, until a condition is met.

FOR *Variable* = *Init*, *Limit* [, *Increment*] DO *statement*
or
FOR *Variable* = *Init*, *Limit* [, *Increment*] DO BEGIN
statements
ENDFOR

GOTO - Transfers program control to point specified by *label*.

GOTO, *label*

IF...THEN...ELSE - Conditionally executes a statement or block of statements.

IF *expression* THEN *statement* [ELSE *statement*]
or
IF *expression* THEN BEGIN
statements
ENDIF [ELSE BEGIN
statements
ENDELSE]

REPEAT...UNTIL - Repeats statement(s) until expression evaluates to true. Subject is always executed at least once.

REPEAT *statement* UNTIL *expression*
or
REPEAT BEGIN
statements
ENDREP UNTIL *expression*

WHILE...DO - Performs statement(s) as long as expression evaluates to true. Subject is never executed if condition is initially false.

WHILE *expression* DO *statement*
or
WHILE *expression* DO BEGIN
statements
ENDWHILE

Functions and Procedures

COMPILE_OPT - Gives IDL compiler information that changes the default rules for compiling functions or procedures.

COMPILE_OPT opt_1 [, opt_2, ..., opt_n]
Note: opt_n can be IDL2, DEFINT32, HIDDEN, OBSOLETE, or STRICTARR

FORWARD_FUNCTION - Causes argument(s) to be interpreted as functions rather than variables (versions of IDL prior to 5.0 used parentheses to declare arrays).

FORWARD_FUNCTION $Name_1$, $Name_2$, ..., $Name_n$

FUNCTION - Defines a function.

FUNCTION *Function_Name*, $parameter_1$, ..., $parameter_n$

PRO - Defines a procedure.

PRO *Procedure_Name*, $argument_1$, ..., $argument_n$

***Procedure_Name* -** Calls a procedure.

Procedure_Name, $argument_1$, ..., $argument_n$

***Result = FUNCTION*(arg_1, ..., arg_n) -** Calls a function.

Executive Commands

Executive commands must be entered at the IDL command prompt. They cannot be used in programs.

.COMPILE - Compiles programs without running.

.COMPILE [$File_1$,..., $File_n$]

To compile from a temporary file: .COMPILE -f *File TempFile*

.CONTINUE - Continues execution of a stopped program.

.CONTINUE

.EDIT - Opens files in editor windows of the IDLDE (Windows and Motif only). Note that filenames are separated by spaces, not commas.

.EDIT $File_1$ [$File_2$ $File_n$]

.FULL_RESET_SESSION - Does everything .RESET_SESSION does, plus additional reset tasks such as unloading sharable libraries.

.FULL_RESET_SESSION

.GO - Executes previously-compiled main program.

.GO

.OUT - Continues execution until the current routine returns.

.OUT

.RESET_SESSION - Resets much of the state of an IDL session without requiring the user to exit and restart the IDL session.

.RESET_SESSION

.RETURN - Continues execution until RETURN statement.

.RETURN

.RNEW - Erases main program variables and then does .RUN.

.RNEW [$File_1$,...,$File_n$]

To save listing in a file:.RNEW -L *ListFile.lis* $File_1$ [, $File_2$,..., $File_n$]

To display listing on screen: .RNEW -T $File_1$ [, $File_2$,..., $File_n$]

.RUN - Compiles and executes IDL commands from files or keyboard.

.RUN [$File_1$,..., $File_n$]

To save listing in a file:.RUN -L *ListFile.lis* $File_1$ [, $File_2$,..., $File_n$]

To display listing on screen: .RUN -T $File_1$ [, $File_2$,..., $File_n$]

.SKIP - Skips over the next *n* statements and then single steps.

.SKIP [*n*]

.STEP - Executes one or *n* statements from the current position.

.STEP [*n*] or .S [*n*]

.STEPOVER - Executes a single statement if the statement doesn't call a routine.

.STEPOVER [*n*] or .SO [*n*]

.TRACE - Similar to .CONTINUE, but displays each line of code before execution.

.TRACE

Special Characters

Ampersand (&) - Separates multiple commands on a single line.

Apostrophe (') - Delimits strings or indicates octal or hex.

Asterisk (*) - Designates an ending subscript range equal to the size of the dimension. Also the multiplication operator and the pointer dereference operator.

At Sign (@) - Include character: Used at beginning of a line to cause the IDL compiler to substitute the contents of the file whose name appears after the @ symbol for the line. In interactive mode, the @ symbol is used to execute a batch file.

Colon (:) - Ends label identifiers. Also separates start and end subscript ranges.

Dollar Sign ($) - Continuation character (at end of line) or spawn operating system command (at start of line).

Exclamation Point (!) - First character of system variable names and font-positioning commands.

Period (.) - First character of executive commands. Also indicates floating-point numbers.

Question Mark (?) - Invokes the online help facility.

Quotation Mark (") - String delimiter or indicates octal number.

Semicolon (;) - First character of comment field. Everything after the semicolon is ignored by IDL. Semicolon can be used as the first character or after an IDL command:

```
; This is a comment
COUNT = 5 ; Set variable COUNT to 5
```

Operators

Mathematical Operators

+ Addition, String Concatenation
- Subtraction and Negation
***** Multiplication, Pointer dereference
/ Division
^ Exponentiation
MOD Modulo

Minimum/Maximum Operators

< The Minimum Operator
> The Maximum Operator

Matrix Operators

 # and ## Matrix Multiplication

Boolean Operators

AND - Boolean AND
NOT - Boolean complement
OR - Boolean OR
XOR - Boolean exclusive OR

Relational Operators

EQ - Equal to
GE - Greater than or equal to
GT - Greater than
LE - Less than or equal to
LT - Less than
NE - Not equal to

Other Operators

[ ] Array concatenation, enclose array subscripts
( ) Group expressions to control order of evaluation
= Assignment
?: Conditional expression

Operator Precedence

The following table lists IDL's operator precedence. Operators with the highest precedence are evaluated first. Operators with equal precedence are evaluated from left to right.

Priority	Operator
First (highest)	( ) (parentheses, to group expressions)
Second	* (pointer dereference) ^ (exponentiation)
Third	* (multiplication) # and ## (matrix multiplication) / (division) MOD (modulus)
Fourth	+ (addition) - (subtraction and negation) < (minimum) > (maximum) NOT (Boolean negation)
Fifth	EQ (equality) NE (not equal) LE (less than or equal) LT (less than) GE (greater than or equal) GT (greater than)
Sixth	AND (Boolean AND) OR (Boolean OR) XOR (Boolean exclusive OR)
Seventh	?: (conditional expression)

Subscripts

Subscripts are used to designate array elements to receive new values, and to retrieve the value of one or more array elements. IDL arrays are zero-based, meaning the first element is element 0.

Vector[**i**] **-** Element i + 1 of a vector. Vector[12] denotes the value of the 13th element of Vector.

Array[**i**, **j**] **-** The element stored at column i, row j of an array.

Vector[**i:j**] **-** Elements i through j of a vector.

Vector[**i:***] **-** Elements from i through the end of a vector.

Array[**i**, ***] **-** Column i of a two-dimensional array.

Array[***, **j**] **-** The jth row of a two-dimensional array.

Array[**i:j**, **m:n**] **-** Subarray of columns i though j, rows m through n.

Array[**Array2**] **-** The elements of Array whose subscripts are the values of Array2.

(Array_Expression)[**i**] **-** Element i of an array-valued expression.

System Variables

IDL system variables contain useful constants, control plotting defaults, and store information about the current IDL session.

Constant System Variables

!DPI - Double-precision pi (p).

!DTOR - Degrees to radians, pi/180 ≈ 0.01745.

!MAP - Read-only system variable used by MAP_SET.

!PI - Single-precision pi (p).

!RADEG - Radians to degrees, 180/pi ≈ 57.2958.

!VALUES - Single- and double-precision NaN and Infinity values.

Graphics System Variables

!D - Information about current graphics device.

Fields: FILL_DIST - line interval, in device coordinates
FLAGS - longword of flags
N_COLORS - number of simultaneously available colors
NAME - string containing name of device
ORIGIN - pan/scroll offset (*pan, scroll*)
TABLE_SIZE - number of color table indices
UNIT - logical number of file open for output
WINDOW - index of currently open window
X_CH_SIZE, Y_CHAR_SIZE - width/height of rectangle that encloses the average character in current font, in device units (usually pixels)
X_PX_CM, Y_PX_CM - approx. number of pixels/cm
X_SIZE, Y_SIZE - total size of the display or window, in device units
X_VSIZE, Y_VSIZE - size of visible area of display or window
ZOOM - X and Y zoom factors

!ORDER - Direction of image transfer: 0=bottom up, 1=top down.

!P - Information for plotting procedures.

Fields: BACKGROUND - background color index
CHANNEL - default source or destination channel
CHARSIZE - character size of annotation when Hershey fonts are selected
CHARTHICK - integer specifying thickness of vector fonts
CLIP - device coords of clipping window ([x_0, y_0, z_0), (x_1, y_1, z_1)]
COLOR - default color index
FONT - integer specifying graphics text font system to use (-1 for Hershey, 0 for output device font, 1 for TrueType)
LINESTYLE - style of lines that connect points (see "Line Styles" on page 72)
MULTI - integer array: [*plots remaining on page, columns per page, rows per page, plots in Z direction, 0 for left to right or 1 for top to bottom*]
NOCLIP - if set, inhibits clipping of graphic vectors
NOERASE - set to nonzero value to prevent erasing
NSUM - number of adjacent points to average
POSITION - normalized coords of plot window (*x0, y0, x1, y1*)
PSYM - plotting symbol index (see "Plotting Symbols" on page 72)
REGION - normalized coords of plot region (*x0, y0, x1, y1*)
SUBTITLE - plot subtitle (under X axis label)
T - homogeneous 4 x 4 transformation matrix
T3D - enables 3D to 2D transformation
THICK - thickness of lines connecting points
TITLE - main plot title
TICKLEN - tick mark length (0.0 to 1.0)

!X, !Y, !Z - Axis structures for X, Y, and Z axes.

Fields: CHARSIZE - character size of annotation when Hershey fonts are selected
CRANGE - output axis range
GRIDSTYLE - linestyle for tick marks/grids (see "Line Styles" on page 72)
MARGIN - 2-element array specifying plot window margins, in units of char size ([*left or bottom, right or top*])
MINOR - number of minor tick marks
OMARGIN - 2-element array specifying plot window outer margins, in units of char size ([*left or bottom, right or top*])
RANGE - 2-element vector specifying input axis range (*min, max*)
REGION - normalized coords of region (2-element floating-point array)
S - 2-element array specifying scaling factors for conversion between data and normalized coords
STYLE - style of the axis encoded as bits in a longword. 1=exact, 2=extend, 4=no axis, 8=no box, 16=inhibit setting Y axis min to 0 when data are all greater than 0 (add values together for multiple effects)
THICK - thickness of axis line
TICKFORMAT - format string or string containing name of function that returns format string used to format axis tick mark labels
TICKLEN - tick mark length, in normal coords
TICKNAME - annotation for each tick (string array of up to 30 elements)

TICKS - number of major tick intervals
TICKV - data values for each tick mark (array of up to 30 elements)
TITLE - string containing axis title
TYPE - type of axis (0 for linear, 1 for logarithmic)
WINDOW - normalized coords of axis end points (2-element floating-point array

Error Handling/Informational System Variables

!ERROR_STATE - Structure containing all error information.

Fields: NAME - string containing error name of IDL-generated component of last error message (read-only)
BLOCK - string containing name of message block for IDL-generated component of last error message (read-only)
CODE - long-integer containing error code of IDL-generated component of last error message
SYS_CODE - long-integer containing error code of operating system component of last error message
MSG - string containing text of IDL-generated component of last error message (read-only)
MSG_PREFIX - string containing prefix string used for error messages
SYS_MSG - string containing text of operating system generated component of last error message (read-only)

!EXCEPT - Controls when IDL checks for math error conditions (0=never report exceptions, 1=report exceptions when interpreter is returning to interactive prompt, 2=report exceptions at end of each IDL statement).

!MOUSE - Status from the last cursor read operation.

Fields: X, Y - location (in device coords) of cursor when mouse button was pressed
BUTTON - specifies which mouse button was pressed (1 if left, 2 if middle, 4 if right)
TIME - number of milliseconds since a base time

!WARN - Report use of obsolete routines.

Fields: OBS_ROUTINES - if set to 1, IDL generates warnings when it encounters use of obsolete routines
OBS_SYSVARS - if set to 1, IDL generates warnings when it encounters use of obsolete system variables
PARENS - if set to 1, IDL generates warnings when it encounters use parentheses to index array
TRUNCATED_FILENAME - if set to 1, IDL generates warnings when a file can only be found by truncating its full name

IDL Environment System Variables

!DIR - Location of the main IDL directory.

!DLM_PATH - Indicates where IDL looks for Dynamically Loadable Modules when started. Read-only.

!EDIT_INPUT - Enables/disables keyboard line editing.

!HELP_PATH - Lists directories in which IDL will search for online help files

!JOURNAL - Logical unit number of journal output, or 0.

!MORE - Set to 0 to prevent paginating help text.

!PATH - Search path for IDL routines.

UNIX: colon-separated list of directories.
VMS: comma-separated list of directories/text libraries.
Windows: semicolon-separated list of directories.
Macintosh: comma-separated list of folders.

!PROMPT - String to be used for IDL prompt.

!QUIET - Suppresses informational messages if set to nonzero.

!VERSION - Type, architecture, and version of IDL.

Graphics Information

Direct Graphics Devices

CGM - The CGM Device
HP - The HP-GL Device
LJ - The LJ Device
MAC - The Macintosh Display Device
NULL - The Null Display Device
PCL - The PCL Device
PRINTER - The Printer Device
PS - The PostScript Device
REGIS - The Regis Terminal Device
TEK - The Tektronix Device
WIN - The Microsoft Windows Device
X - The X Windows Device
Z - The Z-Buffer Device

Graphics Keywords

The following keywords are used with IDL plotting routines (AXIS, CONTOUR, PLOT, OPLOT, SHADE_SURF, and SURFACE) and graphics routines (CURSOR, ERASE, PLOTS, POLYFILL, TV, TVCRS, TVRD, and XYOUTS). Many have system variable equivalents. Not all keywords work with all routines. Listings such as {XYZ}KEYWORD indicate that there are 3 keywords, one for each axis (e.g., XCHARSIZE, YCHARSIZE, ZCHARSIZE).

BACKGROUND - Background color index when erasing.
CHANNEL - Channel index or mask for multi-channel displays.
CHARSIZE - Overall character size.
{XYZ}CHARSIZE - Character size for axes.
CHARTHICK - Overall thickness for vector fonts.
CLIP - Coordinates of clipping window.
COLOR - Color index for data, text, line, or polygon fill.
DATA - Set to plot in data coordinates.
DEVICE - Set to plot in device coordinates.
FONT - Text font index: -1 for vector, 0 for hardware fonts.
{XYZ}GRIDSTYLE - Linestyle index for tickmarks and grids.
LINESTYLE - LInestyle used to connect data points.
{XYZ}MARGIN - Margin of plot window in character units.
{XYZ}MINOR - number of minor tick marks.
NOCLIP - Set to disable clipping of plot.
NODATA - Set to plot only axes, titles, and annotation w/o data.
NOERASE - Set to inhibit erasing before new plot.
NORMAL - Set to plot in normal coordinates.
ORIENTATION - Angle (in degrees counter-clockwise) for text.
POSITION - Position of plot window.
PSYM - Use plotting symbols to plot data points.
{XYZ}RANGE - Axis range.
{XYZ}STYLE - Axis type.
SUBTITLE - String for subtitle.
SYMSIZE - Size of PSYM plotting symbols.
T3D - Set to use 3D transformation store in !P.T.
THICK - Overall line thickness.
{XYZ}THICK - Thickness of axis and tickmark lines.
{XYZ}TICKFORMAT - Allows advanced formatting of tick labels.
TICKLEN - Length of tickmarks in normal coordinates. 1.0 produces a grid. Negative values extend outside window.
{XYZ}TICKLEN - Tickmark lengths for individual axes.
{XYZ}TICKNAME - String array of up to 30 labels for tickmark annotation.
{XYZ}TICKS - Number of major tick intervals for axes.
{XYZ}TICKV - Array of up to 30 elements for tick mark values.
{XYZ}TICK_GET - Variable in which to return values of tick marks.
TITLE - String for plot title.
{XYZ}TITLE - String for specified axis title.
ZVALUE - The Z coordinate for a 2D plot in 3D space.
Z - Z coordinate if Z argument not specified in 3D plot call.

Line Styles

The LINESTYLE keyword to the Direct Graphics plotting routines OPLOT, PLOT, PLOTS, and SURFACE accepts the following values:

Index	Linestyle
0	Solid
1	Dotted
2	Dashed
3	Dash Dot
4	Dash Dot Dot Dot
5	Long Dashes

Plotting Symbols

The PSYM keyword to Direct Graphics plotting routines OPLOT, PLOT, and PLOTS accepts the following values:

PSYM Value	Plotting Symbol
1	Plus sign (+)
2	Asterisk (*)
3	Period (.)
4	Diamond
5	Triangle
6	Square
7	X
8	User-defined. See USERSYM procedure.
9	Undefined
10	Histogram mode.