



# IDL Quick Reference

IDL Version 7.0  
November 2007 Edition  
Copyright © ITT Visual Information Solutions  
All Rights Reserved

## Restricted Rights Notice

The IDL®, IDL Analyst™, ENVI®, and ENVI Zoom™ software programs 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. ITT Visual Information Solutions reserves the right to make changes to this document at any time and without notice.

## Limitation of Warranty

ITT Visual Information Solutions 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.

ITT Visual Information Solutions shall not be liable for any direct, consequential, or other damages suffered by the Licensee or any others resulting from use of the software packages or their documentation.

## Permission to Reproduce this Manual

If you are a licensed user of these products, ITT Visual Information Solutions 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.

## Export Control Information

This software and its associated documentation are subject to the controls of the Export Administration Regulations (EAR). It has been determined that this software is classified as EAR99 under U.S. Export Control laws and regulations, and may not be re-transferred to any destination expressly prohibited by U.S. laws and regulations. The recipient is responsible for ensuring compliance to all applicable U.S. Export Control laws and regulations.

## Acknowledgments

ENVI® and IDL® are registered trademarks of ITT Corporation, registered in the United States Patent and Trademark Office. ION™, ION Script™, ION Java™, and ENVI Zoom™ are trademarks of ITT Visual Information Solutions.

Numerical Recipes™ is a trademark of Numerical Recipes Software. Numerical Recipes routines are used by permission.

GRG2™ is a trademark of Windward Technologies, Inc. The GRG2 software for nonlinear optimization is used by permission.

NCSA Hierarchical Data Format (HDF) Software Library and Utilities. Copyright © 1988-2001, The Board of Trustees of the University of Illinois. All rights reserved.

NCSA HDF5 (Hierarchical Data Format 5) Software Library and Utilities. Copyright © 1998-2002, by the Board of Trustees of the University of Illinois. All rights reserved.

CDF Library. Copyright © 2002, National Space Science Data Center, NASA/Goddard Space Flight Center.

NetCDF Library. Copyright © 1993-1999, University Corporation for Atmospheric Research/Unidata.

HDF EOS Library. Copyright © 1996, Hughes and Applied Research Corporation.

SMACC. Copyright © 2000-2004, Spectral Sciences, Inc. and ITT Visual Information Solutions. All rights reserved.

This software is based in part on the work of the Independent JPEG Group.

Portions of this software are copyrighted by DataDirect Technologies, © 1991-2003.

BandMax®. Copyright © 2003, The Galileo Group Inc.

Portions of this computer program are copyright © 1995-1999, LizardTech, Inc. All rights reserved. MrSID is protected by U.S. Patent No. 5,710,835. Foreign Patents Pending.

Portions of this software were developed using Unisearch's Kakadu software, for which ITT has a commercial license. Kakadu Software. Copyright © 2001. The University of New South Wales, UNSW, Sydney NSW 2052, Australia, and Unisearch Ltd, Australia.

This product includes software developed by the Apache Software Foundation ([www.apache.org/](http://www.apache.org/)).

MODTRAN is licensed from the United States of America under U.S. Patent No. 5,315,513 and U.S. Patent No. 5,884,226.

FLAASH is licensed from Spectral Sciences, Inc. under a U.S. Patent Pending.

Portions of this software are copyrighted by Merge Technologies Incorporated.

Support Vector Machine (SVM) is based on the LIBSVM library written by Chih-Chung Chang and Chih-Jen Lin ([www.csie.ntu.edu.tw/~cjlin/libsvm/](http://www.csie.ntu.edu.tw/~cjlin/libsvm/)), adapted by ITT Visual Information Solutions for remote sensing image supervised classification purposes.

IDL Wavelet Toolkit Copyright © 2002, Christopher Torrence.

IMSL is a trademark of Visual Numerics, Inc. Copyright © 1970-2006 by Visual Numerics, Inc. All Rights Reserved.

Other trademarks and registered trademarks are the property of the respective trademark holders.



# Contents

<b>Chapter 1:</b> <b>Functional List of IDL Routines</b> .....	<b>5</b>
<b>Chapter 2:</b> <b>Alphabetical List of IDL Routines</b> .....	<b>27</b>
<b>Chapter 3:</b> <b>Scientific Data Formats</b> .....	<b>135</b>



# Functional List of IDL Routines

The following is a list of all routines included in IDL, categorized by functionality.

---

3D Visualization	6	Object Class Library	17
Animation	7	Operating System Access	19
Array Creation	7	Performance Testing	20
Array Manipulation	8	Plotting	20
Color Table Manipulation	8	Programming and IDL Control	20
Date and Time	8	Query Routines	21
Debugging	8	Saving/Restoring a Session	22
Dialog Routines	8	Scientific Data Formats	22
Direct Graphics	9	Scope Functions	22
Error Handling	9	Signal Processing	22
Executive Commands	10	Statements	22
External Linking	10	String Processing	23
Font Manipulation	10	Structures	23
Help Routines	10	Type Conversion	23
Image Processing	10	Utilities	23
Input/Output	12	Wavelet Toolkit	24
Language Catalogs	13	Widget Routines	24
Mapping	13	Widget Routines, Compound	24
Mathematics	14	Window Routines	25

## 3D Visualization

---

### 3D Transformations & Scene Setup

---

**CONVERT\_COORD** - Transforms coordinates to and from the coordinate systems supported by IDL.

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

**CREATE\_VIEW** - Sets up 3D transformations.

**CV\_COORD** - Converts 2D and 3D coordinates between coordinate systems.

**IDLgrLight** - Represents a source of illumination for three-dimensional graphic objects.

**IDLgrModel** - Represents a graphical item or group of items that can be transformed (rotated, scaled, and/or translated).

**IDLgrScene** - Represents the entire scene to be drawn and serves as a container of IDLgrView or IDLgrViewgroup objects.

**IDLgrView** - Represents a rectangular area in which graphics objects are drawn.

**IDLgrViewgroup** - Represents a simple container object, very similar to the IDLgrScene object, but can contain objects without a Draw method in addition to IDLgrView objects.

**IDLgrWindow** - Represents an on-screen area on a display device that serves as a graphics destination.

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

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

**SET\_SHADING** - Sets the light source shading parameters.

**SURFR** - Sets up 3D transformations by duplicating rotation, translation, and scaling of SURFACE.

**T3D** - Performs various 3D transformations.

**VERT\_T3D** - Transforms a 3D array by a 4x4 transformation matrix.

**VOXEL\_PROJ** - Generates volume visualizations using voxel technique.

### Polygonal Mesh Routines

---

**COMPUTE\_MESH\_NORMALS** - Computes normal vectors for a set of polygons described by the input array.

**IDLgrPolygon** - Represents one or more polygons that share a given set of vertices and rendering attributes.

**MESH\_CLIP** - Clips a polygonal mesh to an arbitrary plane in space and returns a polygonal mesh of the remaining portion.

**MESH\_DECIMATE** - Reduces the density of geometry while preserving as much of the original data as possible.

**MESH\_ISSOLID** - Computes various mesh properties and enables IDL to determine if a mesh encloses space (is a solid).

**MESH\_MERGE** - Merges two polygonal meshes.

**MESH\_NUMTRIANGLES** - Computes the number of triangles in a polygonal mesh.

**MESH\_OBJ** - Generates a polygon mesh for various simple objects.

**MESH\_SMOOTH** - Performs spatial smoothing on a polygon mesh.

**MESH\_SURFACEAREA** - Computes various mesh properties to determine the mesh surface area, including integration of other properties interpolated on the surface of the mesh.

**MESH\_VALIDATE** - Checks for NaN values in vertices, removes unused vertices, and combines close vertices.

**MESH\_VOLUME** - Computes the volume that the mesh encloses.

**POLYSHADE** - Creates a shaded surface representation from a set of polygons.

### Surfaces and Contours

---

**CONTOUR** - Draws a contour plot.

**ICONTOUR** - Creates an iTool and associated user interface (UI) configured to display and manipulate contour data.

**IDLgrContour** - Draws a contour plot from data stored in a rectangular array or from a set of unstructured points.

**IDLgrSurface** - Represents a shaded or vector representation of a mesh grid.

**IMAGE\_CONT** - Overlays an image with a contour plot.

**ISURFACE** - Creates an iTool and associated user interface (UI) configured to display and manipulate surface data.

**MIN\_CURVE\_SURF** - Interpolates points with a minimum curvature surface or a thin-plate-spline surface. Useful with CONTOUR.

**POLAR\_CONTOUR** - Draws a contour plot from data in polar coordinates.

**SHADE\_SURF** - Creates a shaded-surface representation of gridded data.

**SHADE\_SURF\_IRR** - Creates a shaded-surface representation of an irregularly gridded dataset.

**SHOW3** - Displays array as image, surface plot, and contour plot simultaneously.

**SURFACE** - Plots an array as a wireframe mesh surface.

**XSURFACE** - Provides GUI to SURFACE and SHADE\_SURF.

### Tetrahedral Mesh Routines

---

**IDLgrTessellator** - Decomposes a polygon description into a set of triangles to convert complex and/or concave polygons into a convex form suitable for drawing with the IDLgrPolygon object.

**TETRA\_CLIP** - Clips a tetrahedral mesh to an arbitrary plane in space and returns a tetrahedral mesh of the remaining portion.

**TETRA\_SURFACE** - Extracts a polygonal mesh as the exterior surface of a tetrahedral mesh.

**TETRA\_VOLUME** - Computes properties of tetrahedral mesh array.

### Vector Field Visualization

---

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

**INTERPOL** - Performs linear interpolation on vectors.

**IVECTOR** - Creates an iTool and associated user interface (UI) configured to display and manipulate vector data.

**PARTICLE\_TRACE** - Traces the path of a massless particle through a vector field.

**STREAMLINE** - Generates the visualization graphics from a path.

**VECTOR\_FIELD** - Places colored, oriented vectors of specified length at each vertex in an input vertex array.

**VEL** - Draws a velocity (flow) field with streamlines.

**VELOECT** - Draws a 2D velocity field plot.

## Volume Visualization

---

**EXTRACT\_SLICE** - Returns 2D planar slice extracted from volume.

**IDLgrVolume** - Represents a mapping from a 3D array of data to a 3D array of voxel colors, which, when drawn, are projected to two dimensions.

**IDLgrVRML** - Saves the contents of an Object Graphics hierarchy into a VRML 2.0 format file.

**INTERVAL\_VOLUME** - Generates a tetrahedral mesh from volumetric data.

**ISOSURFACE** - Returns topologically consistent triangles by using oriented tetrahedral decomposition.

**IVOLUME** - Creates an iTool and associated user interface (UI) configured to display and manipulate volume data.

**PROJECT\_VOL** - Returns a translucent rendering of a volume projected onto a plane.

**QGRID3** - Interpolates the dependent variable values to points in a regularly sampled volume.

**QHULL** - Constructs convex hulls, Delaunay triangulations, and Voronoi diagrams.

**RECON3** - Reconstructs a 3D representation of an object from 2D images.

**SEARCH3D** - Finds "objects" or regions of similar data values within a volume.

**SHADE\_VOLUME** - Contours a volume to create a list of vertices and polygons that can be displayed using POLYSHADE.

**SLICER3** - Interactive volume visualization tool.

**VOXEL\_PROJ** - Generates volume visualizations using voxel technique.

**XOBJVIEW** - Displays object viewer widget.

**XOBJVIEW\_ROTATE** - Programmatically rotate the object currently displayed in XOBJVIEW.

**XOBJVIEW\_WRITE\_IMAGE** - Write the object currently displayed in XOBJVIEW to an image file.

**XVOLUME** - Utility for viewing and interactively manipulating volumes and isosurfaces.

## Animation

---

**CW\_ANIMATE** - Creates a compound widget for animation.

**CW\_ANIMATE\_GETP** - Gets pixmap window IDs used by CW\_ANIMATE.

**CW\_ANIMATE\_LOAD** - Loads images into CW\_ANIMATE.

**CW\_ANIMATE\_RUN** - Displays images loaded into CW\_ANIMATE.

**IDLffJPEG2000** - provides the ability to read and write Motion JPEG2000 (MJ2) files.

**IDLgrModel** - Animates the display of objects by displaying a single object in the collection of objects contained in the model when the RENDER\_METHOD property is set.

**IDLgrMPEG** - Creates an MPEG movie file from an array of image frames.

**FLICK** - Causes the display to flicker between two images.

**MPEG\_CLOSE** - Closes an MPEG sequence.

**MPEG\_OPEN** - Opens an MPEG sequence.

**MPEG\_PUT** - Inserts an image array into an MPEG sequence.

**MPEG\_SAVE** - Saves an MPEG sequence to a file.

**XINTERANIMATE** - Displays animated sequence of images.

## Array Creation

---

**BINDGEN** - Returns byte array with each element set to its subscript.

**BYTARR** - Creates a byte vector or array.

**CINDGEN** - Returns a complex array with each element set to its subscript.

**COMPLEXARR** - Creates a complex, single-precision, floating-point vector or array.

**DBLARR** - Creates a double-precision array.

**DCINDGEN** - Returns a double-precision, complex array with each element set to its subscript.

**DCOMPLEXARR** - Creates a complex, double-precision vector or array.

**DINDGEN** - Returns a double-precision array with each element set to its subscript.

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

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

**IDENTITY** - Returns an identity array (an array with ones along the main diagonal and zeros elsewhere) of the specified dimensions.

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

**INTARR** - Creates an integer vector or array.

**L64INDGEN** - Returns a 64-bit integer array with each element set to its subscript.

**LINDGEN** - Returns a longword integer array with each element set to its subscript.

**LON64ARR** - Returns a 64-bit integer vector or array.

**LONARR** - Returns a longword integer vector or array.

**MAKE\_ARRAY** - Returns an array of the specified type, dimensions, and initialization.

**OBJARR** - Creates an array of object references.

**PTRARR** - Creates an array of pointers.

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

**SINDGEN** - Returns a string array with each element set to its subscript.

**STRARR** - Returns string array containing zero-length strings.

**TIMEGEN** - Returns an array of double-precision floating-point values that represent times in Julian dates.

**UINDGEN** - Returns unsigned integer array with each element set to its subscript.

**UINTARR** - Returns an unsigned integer vector or array.

**UL64INDGEN** - Returns an unsigned 64-bit integer array with each element set to its subscript.

**ULINDGEN** - Returns an unsigned longword array with each element set to its subscript.

**ULON64ARR** - Returns an unsigned 64-bit integer vector or array.

**ULONARR** - Returns an unsigned longword integer vector or array.

## Array Manipulation

---

**ARRAY\_EQUAL** - Provides fast test for data equality in cases where the positions of the differing data elements is not required.

**ARRAY\_INDICES** - Converts one-dimensional subscripts of an array into corresponding multi-dimensional subscripts.

**BLAS\_AXPY** - Updates existing array by adding a multiple of another array.

**INVERT** - Computes the inverse of a square array.

**MAX** - Returns the value of the largest element of Array.

**MEDIAN** - Returns the median value of Array or applies a median filter.

**MIN** - Returns the value of the smallest element of an array.

**REFORM** - Changes array dimensions without changing the total number of elements.

**REPLICATE\_INPLACE** - Updates an array by replacing all or selected parts of it with a specified value.

**REVERSE** - Reverses the order of one dimension of an array.

**ROT** - Rotates an image by any amount.

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

**SHIFT** - Shifts elements of vectors or arrays by a specified number of elements.

**SIZE** - Returns array size and type information.

**SORT** - Returns indices of an array sorted in ascending order.

**TOTAL** - Sums of the elements of an array.

**TRANSPOSE** - Transposes an array.

**UNIQ** - Returns subscripts of the unique elements in an array.

**WHERE** - Returns subscripts of nonzero array elements.

**XVAREEDIT** - Provides widget-based editor for IDL variables.

## Color Table Manipulation

---

**CMYK\_CONVERT** - Converts color triples to and from RGB and CMYK.

**COLOR\_QUAN** - Converts color triples to and from RGB, HLS, and HSV.

**COLOR\_QUAN** - Converts true-color (24-bit) image to pseudo-color (8-bit) image.

**COLORMAP\_APPLICABLE** - Determines whether the current visual class supports the use of a colormap.

**CT\_LUMINANCE** - Calculates the luminance of colors.

**CW\_PALETTE\_EDITOR** - Creates compound widget to display and edit color palettes.

**CW\_PALETTE\_EDITOR\_GET** - Gets CW\_PALETTE\_EDITOR properties.

**CW\_PALETTE\_EDITOR\_SET** - Sets CW\_PALETTE\_EDITOR properties.

**GAMMA\_CT** - Applies gamma correction to a color table.

**H\_EQ\_CT** - Histogram-equalizes the color tables for an image or a region of the display.

**H\_EQ\_INT** - Interactively histogram-equalizes the color tables of an image or a region of the display.

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

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

**IDLgrPalette** - Represents a color lookup table that maps indices to red, green, and blue values.

**LOADCT** - Loads one of the predefined IDL color tables.

**MODIFYCT** - Saves modified color tables in the IDL color table file.

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

**PSEUDO** - Creates pseudo-color table based on Lightness, Hue, and Brightness system.

**REDUCE\_COLORS** - Reduces the number of colors used in an image by eliminating unused pixel values.

**STRETCH** - Stretches color table for contrast enhancement.

**TEK\_COLOR** - Loads color table based on Tektronix printer.

**TVLCT** - Loads display color tables.

**XLOADCT** - Provides GUI to interactively select and load color tables.

**XPALETTE** - Displays widget used to create and modify color tables.

## Date and Time

---

**BIN\_DATE** - Converts ASCII date/time string to binary string.

**CALDAT** - Converts Julian date to month, day, year.

**CALENDAR** - Displays a calendar for a given month or year.

**JULDAY** - Returns Julian Day Number for given month, day, and year.

**SYSTEM** - Returns the current time as either a date/time string, as the number of seconds elapsed since 1 January 1970, or as a Julian date/time value.

**TIMEGEN** - Returns an array of double-precision floating-point values that represent date/times in terms of Julian values.

## Debugging

---

**.CONTINUE** - Continues execution of a stopped program.

**.SKIP** - Skips over the next n statements and then single steps.

**.STEP** - Executes one or n statements from the current position.

**.STEPOVER** - Executes a single statement if the statement doesn't call a routine.

**.TRACE** - Similar to .CONTINUE, but displays each line of code before execution.

**BREAKPOINT** - Sets and clears breakpoints for debugging.

**SHMDEBUG** - Print debugging information when a variable loses reference to an underlying shared memory segment.

**STOP** - Stops the execution of a running program or batch file.

## Dialog Routines

---

**DIALOG\_MESSAGE** - Creates modal message dialog.

**DIALOG\_PICKFILE** - Creates native file-selection dialog.



**DIALOG\_PRINTERSETUP** - Opens native dialog used to set properties for a printer.

**DIALOG\_PRINTJOB** - Opens native dialog used to set parameters for a print job.

**DIALOG\_READ\_IMAGE** - Presents GUI for reading image files.

**DIALOG\_WRITE\_IMAGE** - Presents GUI for writing image files.

## Direct Graphics

---

**ANNOTATE** - Starts IDL widget used to interactively annotate images and plots with text and drawings.

**ARROW** - Draws line with an arrow head.

**AXIS** - Draws an axis of the specified type and scale at a given position.

**BAR\_PLOT** - Creates a bar graph.

**BOX\_CURSOR** - Emulates operation of a variable-sized box cursor.

**CONVERT\_COORD** - Transforms coordinates to and from the coordinate systems supported by IDL.

**CONTOUR** - Draws a contour plot.

**CREATE\_VIEW** - Sets the various system variables required to define a coordinate system and a 3-D view.

**CURSOR** - Reads position of the interactive graphics cursor.

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

**DEFROI** - Defines an irregular region of interest of an image using the image display system and the cursor and mouse.

**DEVICE** - Sets to plot in device coordinates.

**DRAW\_ROI** - Draws a region or group of regions to the current Direct Graphics device.

**EMPTY** - Empties the graphics output buffer.

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

**ERRPLOT** - Plots error bars over a previously drawn plot.

**FLICK** - Causes the display to flicker between two output images at a given rate.

**FORMAT\_AXIS\_VALUES** - Formats numbers as strings for use as axis values.

**IMAGE\_CONT** - Overlays an image with a contour plot.

**LABEL\_REGION** - Consecutively labels all of the regions, or blobs, of a bi-level image with a unique region index.

**LOADCT** - Loads a pre-defined color table.

**OPLLOT** - Plots vector data over a previously drawn plot.

**OPLOTERR** - Plots error bars over a previously drawn plot.

**PLOT** - Draws a graph of vector arguments.

**PLOT\_3DBOX** - Plots a function of two variables (e.g.,  $Z=f(X, Y)$ ) inside a 3-D box.

**PLOT\_FIELD** - Plots a 2-D field. N random points are picked, and from each point a path is traced along the field.

**PLOTERR** - plots individual data points with error bars.

**PLOTS** - Plots vectors and points.

**POLAR\_CONTOUR** - Draws a contour plot from data in polar coordinates.

**POLAR\_SURFACE** - Interpolates a surface from polar coordinates (R, Theta, Z) to rectangular coordinates (X, Y, Z).

**POLYFILL** - Fills the interior of a polygon.

**POLYSHADE** - Creates a shaded-surface representation of one or more solids described by a set of polygons.

**PROFILE** - Extracts a profile from an image.

**PROFILES** - Interactively examines image profiles.

**PROJECT\_VOL** - Returns a two-dimensional image that is the projection of a 3-D volume of data onto a plane (similar to an X-ray).

**RDPIX** - Interactively displays the X position, Y position, and pixel value at the cursor.

**SCALE3** - Sets up transformation and scaling parameters for basic 3-D viewing.

**SCALE3D** - Scales the 3-D unit cube (a cube with the length of each side equal to 1) into the viewing area.

**SET\_PLOT** - Sets the output device used by the IDL direct graphics procedures.

**SET\_SHADING** - Modifies the light source shading parameters that affect the output of SHADE\_SURF and POLYSHADE.

**SHADE\_SURF** - Creates a shaded-surface representation of a regular or nearly-regular gridded surface.

**SHADE\_SURF\_IRR** - Creates a shaded-surface representation of an irregularly gridded elevation dataset. Given a 3-D volume and a contour value, SHADE\_VOLUME produces a list of vertices and polygons describing the contour surface.

**SHADE\_VOLUME** - Given a 3-D volume and a contour value, produces a list of vertices and polygons describing the contour surface.

**SHOW3** - Combines an image, a surface plot of the image data, and a contour plot of the images data in a single tri-level display.

**SURFACE** - draws a wire-mesh representation of a two-dimensional array projected into two dimensions, with hidden lines removed.

**THREED** - Plots a 2D array as a pseudo 3D plot.

**TV** - Displays an image.

**TVCRS** - Manipulates the image display cursor.

**TVLCT** - Loads a predefined color table or a color table from specified variables.

**TVRD** - Returns the contents of the specified rectangular portion of the current graphics window or device.

**TVSCL** - Scales and displays an image.

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

**WSET** - Selects the current window.

**WSHOW** - Exposes or hides the designated window.

**XYOUTS** - Draws text on currently-selected graphics device.

**ZOOM** - Zooms portions of the display.

**ZOOM\_24** - Zooms portions of true-color (24-bit) display.

## Error Handling

---

**CATCH** - Intercepts and processes error messages, and continues program execution.

**MESSAGE** - Issues error and informational messages.

**ON\_ERROR** - Designates the error recovery method.

**ON\_IOERROR** - Declares I/O error exception handler.

**STRMESSAGE** - Returns the text of a given error number.

## Executive Commands

---

**.COMPILE** - Compiles programs without running.

**.CONTINUE** - Continues execution of a stopped program.

**.EDIT** - Opens files in editor windows of the IDL Workbench.

**.FULL\_RESET\_SESSION** - Does everything .RESET\_SESSION does, plus additional reset tasks such as unloading sharable libraries.

**.GO** - Executes a previously compiled \$MAIN\$ program.

**.OUT** - Continues execution until the current routine returns.

**.RESET\_SESSION** - Resets much of the state of an IDL session without requiring the user to exit and restart the IDL session.

**.RETURN** - Continues execution until RETURN statement.

**.RNEW** - Erases \$MAIN\$ program variables and then executes .RUN.

**.RUN** - Compiles and executes IDL commands from files or keyboard.

**.SKIP** - Skips over the next n statements and then single steps.

**.STEP** - Executes one or n statements from the current position.

**.STEPOVER** - Executes a single statement if the statement does not call a routine.

**.TRACE** - Similar to .CONTINUE, but displays each line of code before execution.

## External Linking

---

**CALL\_EXTERNAL** - Calls a function in an external sharable object and returns a scalar value.

**DLM\_LOAD** - Explicitly causes a DLM to be loaded.

**IDLcomActiveX** - Creates an IDL object that encapsulates an ActiveX control.

**IDLcomIDispatch** - Creates an IDL object that encapsulates a COM object.

**IDLEXBR\_ASSISTANT** - Launches the Export Bridge Assistant.

**IDLjavaObject** - An IDL object encapsulating a Java object. IDL provides data type and other translation services, allowing IDL programs to access the Java object's methods and properties using standard IDL syntax.

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

**MAKE\_DLL** - Compiles and links sharable libraries (DLLs).

## Font Manipulation

---

**EFONT** - Interactive vector font editor and display tool.

**IDLgrFont** - represents a typeface, style, weight, and point size that may be associated with text objects.

**PS\_SHOW\_FONTS** - Displays all the PostScript fonts that IDL knows about.

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

**SHOWFONT** - Displays a TrueType or vector font

**XFONT** - Creates modal widget to select and view an X Windows font.

## Help Routines

---

**?** - Invokes the IDL Online Help facility when entered at the IDL command line.

**DOC\_LIBRARY** - Extracts documentation headers from IDL programs.

**HELP** - Provides information about the current IDL session.

**MEMORY** - Returns information about dynamic memory currently in use by the IDL session.

**MK\_HTML\_HELP** - Converts text documentation headers to HTML files.

**ONLINE\_HELP** - Invokes online help viewer from programs.

**STRUCT\_HIDE** - Prevents the IDL HELP procedure from displaying information about structures or objects.

## Image Processing

---

### Contrast Enhancement and Filtering

---

**ADAPT\_HIST\_EQUAL** - Performs adaptive histogram equalization

**BUTTERWORTH** - Returns the absolute value of the low-pass Butterworth kernel.

**BYTSCAL** - Scales all values of an array into range of bytes.

**CANNY** - Implements the Canny edge-detection algorithm.

**CONVOL** - Convolves two vectors or arrays.

**DIGITAL\_FILTER** - Calculates coefficients of a non-recursive, digital filter.

**FFT** - Returns the Fast Fourier Transform of an array.

**HILBERT** - Constructs a Hilbert transform.

**HIST\_EQUAL** - Histogram-equalizes an image.

**IR\_FILTER** - Performs the infinite or finite impulse response filter on data.

**LEEFILT** - Performs the Lee filter algorithm on an image array.

**MEDIAN** - Returns the median value of Array or applies a median filter.

**ROBERTS** - Returns an approximation of Roberts edge enhancement.

**SMOOTH** - Smooths with a boxcar average.

**SOBEL** - Returns an approximation of Sobel edge enhancement.

**UNSHARP\_MASK** - Performs an unsharp-mask sharpening filter on a two-dimensional array or a truecolor image.

**See Also** - [Wavelet Toolkit](#)

### Feature Extraction/Image Segmentation

---

**CONTOUR** - Draws a contour plot.

**DEFROI** - Defines an irregular region of interest of an image.

**HISTOGRAM** - Computes the density function of an array.

**HOUGH** - Returns the Hough transform of a two-dimensional image.

**IMAGE\_STATISTICS** - Computes sample statistics for a given array of values.

**ISOCOUNTOUR** - Interprets the contouring algorithm found in the IDLgrContour object.

**ISOSURFACE** - Returns topologically consistent triangles by using oriented tetrahedral decomposition.

**LABEL\_REGION** - Labels regions (blobs) of a bi-level image.

**MAX** - Returns the value of the largest element of Array.

**MEDIAN** - Returns the median value of Array or applies a median filter.

**MIN** - Returns the value of the smallest element of an array.

**PROFILES** - Interactively examines image profiles.

**RADON** - Returns the Radon transform of a two-dimensional image.

**REGION\_GROW** - Perform region growing.

**SEARCH2D** - Finds "objects" or regions of similar data within a 2D array.

**THIN** - Returns the "skeleton" of a bi-level image.

**UNIQ** - Returns subscripts of the unique elements in an array.

**WATERSHED** - Applies the morphological watershed operator to a grayscale image.

**WHERE** - Returns subscripts of nonzero array elements.

## Image Display

---

**DISSOLVE** - Provides a digital "dissolve" effect for images.

**IDLgrImage** - Creates an image object that represents a mapping from a 2D array of data values to a 2D array of pixel colors.

**IDLgrPalette** - Represents a color lookup table that maps indices to red, green, and blue values.

**IMAGE** - Creates an iTool and associated user interface (UI) configured to display and manipulate image data.

**RDPIX** - Interactively displays image pixel values.

**SLIDE\_IMAGE** - Creates a scrolling graphics window for examining large images.

**TV** - Displays an image. To scale and display the image, use TVSCL.

**TVCRS** - Manipulates the image display cursor.

**TVLCT** - Loads display color tables.

**TVSCL** - Scales and displays an image.

**XOBJVIEW** - Displays object viewer widget.

**XOBJVIEW\_ROTATE** - Programmatically rotate the object currently displayed in XOBJVIEW.

**XOBJVIEW\_WRITE\_IMAGE** - Write the object currently displayed in XOBJVIEW to an image file.

**ZOOM** - Zooms portions of the display.

**ZOOM\_24** - Zooms portions of true-color (24-bit) display.

## Image Geometry Transformations

---

**CONGRID** - Resamples an image to any dimensions.

**EXPAND** - Shrinks/expands image using bilinear interpolation.

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

**INTERPOLATE** - Returns an array of interpolates.

**INVERT** - Computes the inverse of a square array.

**POLY\_2D** - Performs polynomial warping of images.

**POLYWARP** - Performs polynomial spatial warping.

**REBIN** - Resizes a vector or array by integer multiples.

**REFORM** - Changes array dimensions without changing the total number of elements.

**REVERSE** - Reverses the order of one dimension of an array.

**ROT** - Rotates an image by any amount.

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

**SHIFT** - Shifts elements of vectors or arrays by a specified number of elements.

**TRANPOSE** - Transposes an array.

**WARP\_TRI** - Warps an image using control points.

## Morphological Image Operators

---

**DILATE** - Implements morphologic dilation operator on binary and grayscale images.

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

**LABEL\_REGION** - Labels regions (blobs) of a bi-level image.

**MORPH\_CLOSE** - Applies closing operator to binary or grayscale image.

**MORPH\_DISTANCE** - Estimates N-dimensional distance maps, which contain for each foreground pixel the distance to the nearest background pixel, using a given norm.

**MORPH\_GRADIENT** - Applies the morphological gradient operator to a grayscale image.

**MORPH\_HITORMISS** - Applies the hit-or-miss operator to a binary image.

**MORPH\_OPEN** - Applies the opening operator to a binary or grayscale image.

**MORPH\_THIN** - Performs a thinning operation on binary images.

**MORPH\_TOPHAT** - Applies top-hat operator to a grayscale image.

**WATERSHED** - Applies the morphological watershed operator to a grayscale image.

## Regions of Interest

---

**CW\_DEFROI** - Creates compound widget used to define region of interest.

**DEFROI** - Defines an irregular region of interest of an image.

**DRAW\_ROI** - Draws region or group of regions to current Direct Graphics device.

**IDLanROI** - Represents a region of interest used for analysis.

**IDLanROIGroup** - Analytical representation of a group of regions of interest.

**IDLgrROI** - Object graphics representation of a region of interest.

**IDLgrROIGroup** - Object Graphics representation of a group of regions of interest.

**LABEL\_REGION** - Labels regions (blobs) of a bi-level image.

**REGION\_GROW** - Grows an initial region to include all areas that match specified constraints.

**XROI** - Utility for defining regions of interest, and obtaining geometry and statistical data about these ROIs.

## Input/Output

---

### General File Access

---

**APP\_USER\_DIR** - Provides access to the *application user directory*.

**APP\_USER\_DIR\_QUERY** - Locates existing *application user directories*.

**DIALOG\_PICKFILE** - Creates native file-selection dialog.

**FILE\_BASENAME** - Returns the final segment of a file path.

**FILE\_CHMOD** - Changes the current access permission associated with a file or directory.

**FILE\_COPY** - Copies files or directories to a new location.

**FILE\_DELETE** - Removes a file or empty directory.

**FILE\_DIRNAME** - Returns the directory name of a file path consisting of everything except the final segment of the file path.

**FILE\_EXPAND\_PATH** - Returns a fully qualified path regardless of the current working directory.

**FILE\_INFO** - Returns status information about a file.

**FILE\_LINES** - Returns the number of lines of text in a file.

**FILE\_LINK** - Creates Unix file links.

**FILE\_MKDIR** - Creates a new directory or directories.

**FILE\_MOVE** - Renames files and directories.

**FILE\_POLL\_INPUT** - Blocks processing until it detects that a read operation on a specified file will succeed.

**FILE\_READLINK** - Returns the path pointed to by a Unix symbolic link.

**FILE\_SAME** - Determines if two different file names refer to the same underlying file.

**FILE\_SEARCH** - Returns a string array containing the names of all files matching the input path specification.

**FILE\_TEST** - Test a file or directory for existence and other specific attributes.

**FILE\_WHICH** - Returns the path for the first file for the given name found by searching the specified path.

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

**FSTAT** - Returns information about a specified file unit.

**PATH\_SEP** - Returns the proper file path segment separator character for the current operating system.

### Image Data Formats

---

NOTE: Also see "Query Routines" on page 21.

**DIALOG\_READ\_IMAGE** - Presents GUI for reading image files.

**DIALOG\_WRITE\_IMAGE** - Presents GUI for writing image files.

**IDLffDICOM** - Contains the data for one or more images embedded in a DICOM part 10 file.

**IDLffDicomEx** - Allows you to read and write data contained in a DICOM file. See the *Medical Imaging in IDL* manual.

**IDLffJPEG2000** - Contains the data for one or more JPEG2000 files.

**IDLffMJPEG2000** - Allows you to create and play Motion JPEG2000 animations contained in MJ2 files.

**IDLffMrSID** - Allows you to query and load image data from a MrSID image file.

**MPEG\_CLOSE** - Closes an MPEG sequence.

**MPEG\_OPEN** - Opens an MPEG sequence.

**MPEG\_PUT** - Inserts an image array into an MPEG sequence.

**MPEG\_SAVE** - Saves an MPEG sequence to a file.

**READ\_BMP** - Reads Microsoft Windows bitmap file (.BMP).

**READ\_DICOM** - Reads an image from a DICOM file.

**READ\_GIF** - Reads GIF file (.GIF)

**READ\_IMAGE** - Reads the image contents of a file and returns the image in an IDL variable.

**READ\_INTERFILE** - Reads Interfile (v3.3) file.

**READ\_JPEG** - Reads JPEG file.

**READ\_JPEG2000** - Reads JPEG 2000 file.

**READ\_MRSID** - Reads MrSID file.

**READ\_PICT** - Reads Macintosh PICT (version 2) bitmap file.

**READ\_PNG** - Reads Portable Network Graphics (PNG) file.

**READ\_PPM** - Reads PGM (gray scale) or PPM (portable pixmap for color) file.

**READ\_SRF** - Reads Sun Raster Format file.

**READ\_TIFF** - Reads TIFF format file.

**READ\_X11\_BITMAP** - Reads X11 bitmap file.

**READ\_XWD** - Reads X Windows Dump file.

**TVRD** - Reads an image from a window into a variable.

**WRITE\_BMP** - Writes Microsoft Windows Version 3 device independent bitmap file (.BMP).

**WRITE\_GIF** - Writes GIF file (.GIF).

**WRITE\_IMAGE** - Writes an image and its color table vectors, if any, to a file of a specified type.

**WRITE\_JPEG** - Writes JPEG file.

**WRITE\_JPEG2000** - Writes JPEG 2000 file.

**WRITE\_NRIF** - Writes NCAR Raster Interchange Format rasterfile.

**WRITE\_PICT** - Writes Macintosh PICT (version 2) bitmap file.

**WRITE\_PNG** - Writes Portable Network Graphics (PNG) file.

**WRITE\_PPM** - Writes PPM (true-color) or PGM (gray scale) file.

**WRITE\_SRF** - Writes Sun Raster File (SRF).

**WRITE\_TIFF** - Writes TIFF file with 1 to 3 channels.

**XOBJVIEW\_WRITE\_IMAGE** - Write the object currently displayed in XOBJVIEW to an image file.

## Scientific Data Formats

---

- CDF Routines** - see the [Scientific Data Formats](#) chapter of this manual in [quickref.pdf](#), located on your IDL CD-ROM.
- EOS Routines** - see the [Scientific Data Formats](#) chapter of this manual in [quickref.pdf](#), located on your IDL CD-ROM.
- H5\_BROWSER** - Opens a GUI to view contents of HDF5 files.
- HDF Routines** - see the [Scientific Data Formats](#) chapter of this manual in [quickref.pdf](#), located on your IDL CD-ROM.
- HDF5 Routines** - see the [Scientific Data Formats](#) chapter of this manual in [quickref.pdf](#), located on your IDL CD-ROM.
- HDF\_BROWSER** - Opens a GUI to view contents of HDF, HDF-EOS, or NetCDF file.
- HDF\_READ** - Extracts HDF, HDF-EOS, and NetCDF data and metadata into an output structure.
- NetCDF Routines** - see the [Scientific Data Formats](#) chapter of this manual in [quickref.pdf](#), located on your IDL CD-ROM.

## Other Data Formats

---

- ASCII\_TEMPLATE** - Presents a GUI that generates a template defining an ASCII file format.
- BINARY\_TEMPLATE** - Presents a GUI for interactively generating a template structure for use with `READ_BINARY`.
- IDLffDXF** - Object that contains geometry, connectivity, and attributes for graphics primitives.
- IDLffShape** - Contains geometry, connectivity and attributes for graphics primitives accessed from ESRI Shapefiles.
- IDLffXMLDOM\*** - Represents classes that provide support for IDL's XML Document Object Model (DOM). See "[IDLffXMLDOM Classes](#)" on page 83.
- IDLffXMLSAX** - Represents an XML SAX level 2 parser.
- IDLgrVRML** - Saves the contents of an Object Graphics hierarchy into a VRML 2.0 format file.
- READ\_ASCII** - Reads data from an ASCII file.
- READ\_BINARY** - Reads the contents of a binary file using a passed template or basic command line keywords.
- READ\_SYLK** - Reads Symbolic Link format spreadsheet file.
- READ\_WAV** - Reads the audio stream from the named .WAV file.
- READ\_WAVE** - Reads Wavefront Advanced Visualizer file.
- WRITE\_SYLK** - Writes SYLK (Symbolic Link) spreadsheet file.
- WRITE\_WAV** - Writes the audio stream to the named .WAV file.
- WRITE\_WAVE** - Writes Wavefront Advanced Visualizer (.WAV) file.

## General Input/Output

---

- ASSOC** - Associates an array structure with a file.
- CLOSE** - Closes the specified files.
- COPY\_LUN** - Copies data between two open files.
- EOF** - Tests the specified file for the end-of-file condition.
- FLUSH** - Flushes file unit buffers.
- FREE\_LUN** - Frees previously-reserved file units.
- GET\_KBRD** - Gets one input IDL character.

- GET\_LUN** - Reserves a logical unit number (file unit).
- IOCTL** - Performs special functions on UNIX files.
- OPENR/OPENU/OPENW** - Opens files for reading, updating, or writing.
- POINT\_LUN** - Sets or gets current position of the file pointer.
- PRINT/PRINTF** - Writes formatted output to screen or file.
- READ/READF** - Reads formatted input from keyboard or file.
- READS** - Reads formatted input from a string variable.
- READU** - Reads unformatted binary data from a file.
- SEM\_CREATE** - Creates a reference to a semaphore in the current IDL process. Creates the semaphore itself if necessary.
- SEM\_DELETE** - Destroys a reference to the specified semaphore in the current IDL process. Optionally destroys the semaphore itself.
- SEM\_LOCK** - Attempts to gain the lock on an existing semaphore (created by a call to the `SEM_CREATE` function) for the current IDL process.
- SEM\_RELEASE** - Releases the lock on the specified semaphore held by the current IDL process.
- SHMMAP** - Maps anonymous shared memory, or local disk files, into the memory address space of the currently executing IDL process.
- SHMUNMAP** - Removes a memory segment previously created by `SHMMAP` from the system.
- SHMVAR** - Creates an IDL array variable that uses the memory from a current mapped memory segment created by the `SHMMAP` procedure.
- SKIP\_LUN** - Reads data in an open file and moves the file pointer.
- SOCKET** - Opens a client-side TCP/IP Internet socket as an IDL file unit.
- TRUNCATE\_LUN** - Truncates an open file at the location of the current file pointer.
- WRITEU** - Writes unformatted binary data to a file.

## Language Catalogs

---

- IDLffLangCat** - Finds and loads an XML language catalog.
- LOCALE\_GET** - Returns the current locale of the operating platform.
- MULTI** - Returns a catalog object for the given parameters if found.

## Mapping

---

- IDLffShape** - Contains geometry, connectivity and attributes for graphics primitives accessed from ESRI Shapefiles.
- IMAP** - Displays georeferenced data in an iTool.
- LL\_ARC\_DISTANCE** - Returns the longitude and latitude of a point given arc distance and azimuth.
- MAP\_2POINTS** - Returns distance, azimuth, and path relating to the great circle or rhumb line connecting two points on a sphere.
- MAP\_CONTINENTS** - Draws continental boundaries, filled continents, political boundaries, coastlines, and/or rivers, over an existing map projection established by `MAP_SET`.
- MAP\_GRID** - Draws parallels and meridians over a map projection.
- MAP\_IMAGE** - Returns an image warped to fit the current map projection. (Use when map data is larger than the display).

**MAP\_PATCH** - Returns an image warped to fit the current map projection. (Use when map data is smaller than the display).

**MAP\_PROJ\_FORWARD** - Transforms map coordinates from longitude/latitude to Cartesian (X, Y) coordinates

**MAP\_PROJ\_IMAGE** - Warps an image from geographic coordinates to a specified map projection.

**MAP\_PROJ\_INFO** - Returns information about current map and/or the available projections.

**MAP\_PROJ\_INIT** - Initializes a mapping projection, using either IDL's own map projections or the General Cartographic Transformation Package (GCTP) map projections.

**MAP\_PROJ\_INVERSE** - Transforms map coordinates from Cartesian (X, Y) coordinates to longitude/latitude.

**MAP\_SET** - Establishes map projection type and limits.

## Mathematics

---

### Complex Numbers

---

**COMPLEX** - Converts argument to complex type.

**CONJ** - Returns the complex conjugate of X.

**DCOMPLEX** - Converts argument to double-precision complex type.

**IMAGINARY** - Returns the imaginary part of a complex value.

**REAL\_PART** - Returns the real part of a complex-valued argument.

### Correlation Analysis

---

**A\_CORRELATE** - Computes autocorrelation.

**C\_CORRELATE** - Computes cross correlation.

**CORRELATE** - Computes the linear Pearson correlation.

**M\_CORRELATE** - Computes multiple correlation coefficient.

**P\_CORRELATE** - Computes partial correlation coefficient.

**R\_CORRELATE** - Computes rank correlation.

### Curve and Surface Fitting

---

**COMFIT** - Fits paired data using one of six common filtering functions.

**CRVLENGTH** - Computes the length of a curve.

**CURVEFIT** - Fits multivariate data with a user-supplied function.

**GAUSS2DFIT** - Fits a 2D elliptical Gaussian equation to rectilinearly gridded data.

**GAUSSFIT** - Fits the sum of a Gaussian and a quadratic.

**GRID\_TPS** - Uses thin plate splines to interpolate a set of values over a regular 2D grid, from irregularly sampled data values.

**KRIG2D** - Interpolates set of points using kriging.

**LADFIT** - Fits paired data using least absolute deviation method.

**LINFIT** - Fits by minimizing the Chi-square error statistic.

**LMFIT** - Does a non-linear least squares fit.

**MIN\_CURVE\_SURF** - Interpolates points with a minimum curvature surface or a thin-plate-spline surface. Useful with CONTOUR.

**POLY\_FIT** - Performs a least-square polynomial fit.

**REGRESS** - Computes fit using multiple linear regression.

**SFIT** - Performs polynomial fit to a surface.

**SVDFIT** - Multivariate least squares fit using SVD method.

**TRIGRID** - Interpolates irregularly-gridded data to a regular grid from a triangulation.

## Differentiation and Integration

---

**CRVLENGTH** - Computes the length of a curve.

**DERIV** - Performs differentiation using 3-point Lagrangian interpolation.

**DERIVSIG** - Computes standard deviation of derivative found by DERIV.

**INT\_2D** - Computes the double integral of a bivariate function.

**INT\_3D** - Computes the triple integral of a trivariate function.

**INT\_TABULATED** - Integrates a tabulated set of data.

**LSODE** - Advances a solution to a system of ordinary differential equations one time-step H.

**QROMB** - Evaluates integral over a closed interval.

**QROMO** - Evaluates integral over an open interval.

**QSIMP** - Evaluates integral using Simpson's rule.

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

## Eigenvalues and Eigenvectors

---

**EIGENQL** - Computes eigenvalues and eigenvectors of a real, symmetric array.

**EIGENVEC** - Computes eigenvectors of a real, non-symmetric array.

**ELMHES** - Reduces nonsymmetric array to upper Hessenberg form.

**HQR** - Returns all eigenvalues of an upper Hessenberg array.

**TRIQL** - Determines eigenvalues and eigenvectors of tridiagonal array.

**TRIRED** - Reduces a real, symmetric array to tridiagonal form.

## Gridding and Interpolation

---

**BILINEAR** - Computes array using bilinear interpolation.

**CONGRID** - Shrinks or expands the size of an array by an arbitrary amount.

**GRID\_INPUT** - Preprocesses and sorts two-dimensional scattered data points, and removes duplicate values.

**GRID\_TPS** - Uses thin plate splines to interpolate a set of values over a regular 2D grid, from irregularly sampled data values.

**GRID3** - Creates a regularly-gridded 3D dataset from a set of scattered 3D nodes.

**GRIDDATA** - Interpolates scattered data values and locations sampled on a plane or a sphere to a regular grid.

**INTERPOL** - Performs linear interpolation on vectors.

**INTERPOLATE** - Returns an array of interpolates.

**KRIG2D** - Interpolates set of points using kriging.

**MIN\_CURVE\_SURF** - Interpolates points with a minimum curvature surface or a thin-plate-spline surface. Useful with CONTOUR.

**POLAR\_SURFACE** - Interpolates a surface from polar coordinates to rectangular coordinates.

**SPH\_SCAT** - Performs spherical gridding.

**SPL\_INIT** - Establishes the type of interpolating spline.

**SPL\_INTERP** - Performs cubic spline interpolation (Numerical Recipes).

**REBIN** - Resizes a vector or an array to a set of given dimensions.

**SPLINE** - Performs cubic spline interpolation.

**SPLINE\_P** - Performs parametric cubic spline interpolation.

**TRI\_SURF** - Interpolates gridded set of points with a smooth quintic surface.

**TRIANGULATE** - Constructs Delaunay triangulation of a planar set of points.

**TRIGRID** - Interpolates irregularly-gridded data to a regular grid from a triangulation.

**VALUE\_LOCATE** - Finds the intervals within a given monotonic vector that brackets a given set of one or more search values.

**VORONOI** - Computes Voronoi polygon given Delaunay triangulation.

## Hypothesis Testing

---

**CTI\_TEST** - Performs chi-square goodness-of-fit test.

**FV\_TEST** - Performs the F-variance test.

**KW\_TEST** - Performs Kruskal-Wallis H-test.

**LNP\_TEST** - Computes the Lomb Normalized Periodogram.

**MD\_TEST** - Performs the Median Delta test.

**R\_TEST** - Runs test for randomness.

**RS\_TEST** - Performs the Wilcoxon Rank-Sum test.

**S\_TEST** - Performs the Sign test.

**TM\_TEST** - Performs t-means test.

**XSQ\_TEST** - Computes Chi-square goodness-of-fit test.

## LAPACK Routines

---

**LA\_CHOLDC** - Computes the Cholesky factorization of an  $n$ -by- $n$  symmetric positive-definite array.

**LA\_CHOLMPROVE** - Uses Cholesky factorization to improve the solution to a system of linear equations.

**LA\_CHOLSOL** - Used in conjunction with LA\_CHOLDC to solve a set of linear equations.

**LA\_DETERM** - Uses LU decomposition to compute the determinant of a square array.

**LA\_EIGENPROBLEM** - Uses the QR algorithm to compute eigenvalues and eigenvectors of an array.

**LA\_EIGENQL** - Computes selected eigenvalues and eigenvectors.

**LA\_EIGENVEC** - Uses the QR algorithm to compute all of some eigenvectors of an array.

**LA\_ELMHES** - Reduces a real nonsymmetric or complex array to upper Hessenberg form.

**LA\_GM\_LINEAR\_MODEL** - Used to solve a general Gauss-Markov linear model problem.

**LA\_HQR** - Uses the multishift QR algorithm to compute all eigenvalues of an array.

**LA\_INVERT** - Uses LU decomposition to compute the inverse of a square array.

**LA\_LEAST\_SQUARE\_EQUALITY** - Used to solve linear least-squares problems.

**LA\_LEAST\_SQUARES** - Used to solve linear least-squares problems.

**LA\_LINEAR\_EQUATION** - Uses LU decomposition to solve a system of linear equations.

**LA\_LUDC** - Computes the LU decomposition of an array.

**LA\_LUMPROVE** - Uses LU decomposition to improve the solution to a system of linear equations.

**LA\_LUSOL** - Used in conjunction with LA\_LUDC to solve a set of linear equations.

**LA\_SVD** - Computes the singular value decomposition of an array.

**LA\_TRIDC** - Computes the LU decomposition of a tridiagonal array.

**LA\_TRIMPROVE** - Improves the solution to a system of linear equations with a tridiagonal array.

**LA\_TRIQL** - Uses the QL and QR variants of the implicitly-shifted QR algorithm to compute the eigenvalues and eigenvectors of an array.

**LA\_TRIRED** - Reduces a real symmetric or complex Hermitian array to real tridiagonal form.

**LA\_TRISOL** - Used in conjunction with LA\_TRIDC to solve a set of linear equations.

## Linear Systems

---

**CHOLDC** - Constructs Cholesky decomposition of a matrix.

**CHOLSOL** - Solves set of linear equations (use with CHOLDC).

**COND** - Computes the condition number of a square matrix.

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

**CROSSP** - Computes vector cross product.

**DETERM** - Computes the determinant of a square matrix.

**GS\_ITER** - Solves linear system using Gauss-Seidel iteration.

**IDENTITY** - Returns an identity array.

**INVERT** - Computes the inverse of a square array.

**LINBCG** - Solves a set of sparse linear equations using the iterative biconjugate gradient method.

**LU\_COMPLEX** - Solves complex linear system using LU decomposition.

**LUDC** - Replaces array with the LU decomposition.

**LUMPROVE** - Uses LU decomposition to iteratively improve an approximate solution.

**LUSOL** - Solves a set of linear equations. Use with LUDC.

**NORM** - Computes Euclidean norm of vector or Infinity norm of array.

**SVDC** - Computes Singular Value Decomposition of an array.

**SVSOL** - Solves set of linear equations using back-substitution.

**TRACE** - Computes the trace of an array.

**TRISOL** - Solves tridiagonal systems of linear equations.

## Mathematical Error Assessment

---

**CHECK\_MATH** - Returns and clears accumulated math error status.

**FINITE** - Returns True if its argument is finite.

**MACHAR** - Determines and returns machine-specific parameters affecting floating-point arithmetic.

## Miscellaneous Math Routines

---

**ABS** - Returns the absolute value of X.

**CEIL** - Returns the closest integer greater than or equal to X.

**CIR\_3PNT** - Returns radius and center of circle, given 3 points.

**COMPLEXROUND** - Rounds a complex array.

**DIAG\_MATRIX** - Constructs a diagonal matrix from an input vector, or if given a matrix, then extracts a diagonal vector.

**DIST** - Creates array with each element proportional to its frequency.

**EXP** - Returns the natural exponential function of given expression.

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

**IMAGINARY** - Returns the imaginary part of a complex value.

**ISHFT** - Performs integer bit shift.

**LEEFILT** - Performs the Lee filter algorithm on an image array.

**MATRIX\_MULTIPLY** - Calculates the IDL matrix-multiply operator (#) of two (possibly transposed) arrays.

**MATRIX\_POWER** - Computes the product of a matrix with itself.

**PNT\_LINE** - Returns the perpendicular distance between a point and a line.

**POLY\_AREA** - Returns the area of a polygon given the coordinates of its vertices.

**PRIMES** - Computes the first K prime numbers.

**PRODUCT** - Returns the product of elements within an array.

**ROUND** - Returns the integer closest to its argument.

**SPH\_4PNT** - Returns center and radius of a sphere given 4 points.

**SQRT** - Returns the square root of X.

**TOTAL** - Sums of the elements of an array.

**VOIGT** - Calculates intensity of atomic absorption line (Voight) profile.

## Multivariate Analysis

---

**CLUST\_WTS** - Computes cluster weights of array for cluster analysis.

**CLUSTER** - Performs cluster analysis.

**CLUSTER\_TREE** - Computes the hierarchical clustering for a set of  $m$  items in an  $n$ -dimensional space.

**CTI\_TEST** - Performs chi-square goodness-of-fit test.

**DENDRO\_PLOT** - Draws a two-dimensional dendrite plot on the current direct graphics device if given a hierarchical tree cluster, as created by CLUSTER\_TREE.

**DENDROGRAM** - Constructs a dendrogram and returns a set of vertices and connectivity that can be used to visualize the dendrite plot if given a hierarchical tree cluster, as created by CLUSTER\_TREE.

**DISTANCE\_MEASURE** - Computes the pairwise distance between a set of items or observations.

**KW\_TEST** - Performs Kruskal-Wallis H-test.

**M\_CORRELATE** - Computes multiple correlation coefficient.

**P\_CORRELATE** - Computes partial correlation coefficient.

**PCOMP** - Computes principal components/derived variables.

**STANDARDIZE** - Computes standardized variables.

## Nonlinear Equations

---

**BROYDEN** - Solves nonlinear equations using Broyden's method.

**FX\_ROOT** - Computes real and complex roots of a univariate nonlinear function using an optimal Müller's method.

**FZ\_ROOTS** - Finds the roots of a complex polynomial using Laguerre's method.

**NEWTON** - Solves nonlinear equations using Newton's method.

## Optimization

---

**AMOEBEA** - Minimizes a function using downhill simplex method.

**CONSTRAINED\_MIN** - Minimizes a function using Generalized Reduced Gradient Method.

**DFFMIN** - Minimizes a function using Davidon-Fletcher-Powell method.

**POWELL** - Minimizes a function using the Powell method.

**SIMPLEX** - Use the simplex method to solve linear programming problems.

## Probability

---

**BINOMIAL** - Computes binomial distribution function.

**CHISQR\_CVF** - Computes cutoff value in a Chi-square distribution.

**CHISQR\_PDF** - Computes Chi-square distribution function.

**F\_CVF** - Computes the cutoff value in an F distribution.

**F\_PDF** - Computes the F distribution function.

**GAUSS\_CVF** - Computes cutoff value in Gaussian distribution.

**GAUSS\_PDF** - Computes Gaussian distribution function.

**GAUSSINT** - Returns integral of Gaussian probability function.

**T\_CVF** - Computes the cutoff value in a Student's t distribution.

**T\_PDF** - Computes Student's t distribution.

## Sparse Arrays

---

NOTE: SPRSIN must be used to convert to sparse storage format before the other routines can be used.

**FULSTR** - Restores a sparse matrix to full storage mode.

**LINBCG** - Solves a set of sparse linear equations using the iterative biconjugate gradient method.

**READ\_SPR** - Reads a row-indexed sparse matrix from a file.

**SPRSAB** - Performs matrix multiplication on sparse matrices.

**SPRSAX** - Multiplies sparse matrix by a vector.

**SPRSIN** - Converts matrix to row-index sparse matrix.

**SPRSTP** - Constructs the transpose of a sparse matrix.

**WRITE\_SPR** - Writes row-indexed sparse array structure to a file.



## Special Math Functions

---

- BESELI** - Returns the I Bessel function of order N for X.  
**BESELJ** - Returns the J Bessel function of order N for X.  
**BESELK** - Returns the K Bessel function of order N for X.  
**BESELY** - Returns the Y Bessel function of order N for X.  
**BETA** - Returns the value of the beta function.  
**ERF** - Returns the value of an error function.  
**ERFC** - Returns the value of a complementary error function.  
**ERFCX** - Returns the value of a scaled complementary error function.  
**EXPINT** - Returns the value of the exponential integral.  
**GAMMA** - Returns the gamma function of X.  
**IBETA** - Computes the incomplete beta function.  
**IGAMMA** - Computes the incomplete gamma function.  
**LAGUERRE** - Returns value of the associated Laguerre polynomial.  
**LEGENDRE** - Returns value of the associated Legendre polynomial.  
**LNGAMMA** - Returns logarithm of the gamma function of X.  
**POLY** - Evaluates polynomial function of a variable.  
**SPHER\_HARM** - Returns value of the spherical harmonic function.

## Statistical Fitting

---

- COMFIT** - Fits paired data using one of six common filtering functions.  
**CURVEFIT** - Fits multivariate data with a user-supplied function.  
**FUNCT** - Evaluates the sum of a Gaussian and a 2nd-order polynomial and optionally returns the value of its partial derivatives.  
**LADFIT** - Fits paired data using least absolute deviation method.  
**LINFIT** - Fits by minimizing the Chi-square error statistic.  
**REGRESS** - Multiple linear regression.  
**SVDFIT** - Multivariate least squares fit using SVD method.

## Statistical Tools

---

- FACTORIAL** - Computes the factorial N!.  
**HIST\_2D** - Returns histogram of two variables.  
**HISTOGRAM** - Computes the density function of an array.  
**KURTOSIS** - Computes statistical kurtosis of n-element vector.  
**MAX** - Returns the value of the largest element of an array.  
**MEAN** - Computes the mean of a numeric vector.  
**MEANABSDEV** - Computes the mean absolute deviation of a vector.  
**MEDIAN** - Returns the median value of Array or applies a median filter.  
**MIN** - Returns the value of the smallest element of an array.  
**MOMENT** - Computes mean, variance, skewness, and kurtosis.  
**RANDOMN** - Returns normally-distributed pseudo-random numbers.  
**RANDOMU** - Returns uniformly-distributed pseudo-random numbers.  
**RANKS** - Computes magnitude-based ranks.  
**SKEWNESS** - Computes statistical skewness of an n-element vector.

- SORT** - Returns the indices of an array sorted in ascending order.  
**STDDEV** - Computes the standard deviation of an n-element vector.  
**TOTAL** - Sums of the elements of an array.  
**VARIANCE** - Computes the statistical variance of an n-element vector.

## Time-Series Analysis

---

- A\_CORRELATE** - Computes autocorrelation.  
**C\_CORRELATE** - Computes cross correlation.  
**SMOOTH** - Smooths with a boxcar average.  
**TS\_COEF** - Computes the coefficients for autoregressive time-series.  
**TS\_DIFF** - Computes the forward differences of a time-series.  
**TS\_FCAST** - Computes future or past values of stationary time-series.  
**TS\_SMOOTH** - Computes moving averages of a time-series.

## Transcendental Functions

---

- ACOS** - Returns the arc-cosine of X.  
**ALOG** - Returns the natural logarithm of X.  
**ALOG10** - Returns the logarithm to the base 10 of X.  
**ASIN** - Returns the arc-sine of X.  
**ATAN** - Returns the arc-tangent of X.  
**COS** - Returns the cosine of X.  
**COSH** - Returns the hyperbolic cosine of X.  
**EXP** - Returns the natural exponential function of a given expression.  
**SIN** - Returns the trigonometric sine of X.  
**SINH** - Returns the hyperbolic sine of X.  
**TAN** - Returns the tangent of X.  
**TANH** - Returns the hyperbolic tangent of X.

## Transforms

---

- BLK\_CON** - Convolve input signal with impulse-response sequence.  
**CHEBYSHEV** - Returns the forward or reverse Chebyshev polynomial expansion.  
**CONVOL** - Convolve two vectors or arrays.  
**FFT** - Returns the Fast Fourier Transform of an array.  
**HILBERT** - Constructs a Hilbert transform.  
**HOUGH** - Returns the Hough transform of a two-dimensional image.  
**RADON** - Returns the Radon transform of a two-dimensional image.  
**WTN** - Returns wavelet transform of the input array.  
 See Also - [Wavelet Toolkit](#)

## Object Class Library

---

### Analysis Objects

---

- IDLanROI** - Represents a region of interest.

**IDLanROIGroup** - Analytical representation of a group of regions of interest.

## File Format Objects

---

**IDLffDICOM** - Contains the data for one or more images embedded in a DICOM Part 10 file.

**IDLffDicomEx** - Allows you to read and write data contained in a DICOM file. See the *Medical Imaging in IDL* manual.

**IDLffDXF** - Contains geometry, connectivity and attributes for graphics primitives.

**IDLffJPEG2000** - Contains the data for one or more images embedded in a JPEG-2000 file as well as functionality for reading and writing JPEG-2000 files.

**IDLffLangCat** - Finds and loads an XML language catalog.

**IDLffMrSID** - Used to query information about and load image data from a MrSID (.sid) image file.

**IDLffShape** - Contains geometry, connectivity and attributes for graphics primitives.

**IDLffXMLDOMAttr** - Represents an attribute that is part of an element object in an XML document.

**IDLffXMLDOMCDATASection** - Used to escape blocks of text in an XML document containing characters that would otherwise be regarded as markup.

**IDLffXMLDOMCharacterData** - Extension of the IDLffXML-DOMNode class that supplies a set of methods for accessing character data in the XML DOM tree.

**IDLffXMLDOMComment** - Represents the content of an XML comment (characters between "`<!--`" and "`-->`").

**IDLffXMLDOMDocument** - Represents the entire XML document as the root of the XML document tree and by providing the primary access to the document's data.

**IDLffXMLDOMDocumentFragment** - Represents a document fragment in an XML document.

**IDLffXMLDOMDocumentType** - References a DocumentType node in an XML document.

**IDLffXMLDOMElement** - References an element node in an XML document.

**IDLffXMLDOMEntity** - References an entity, either parsed or unparsed, in an XML document.

**IDLffXMLDOMEntityReference** - References an entity reference node in an XML document.

**IDLffXMLDOMNamedNodeMap** - Container for IDLffXML-DOM nodes that uses node names to access the nodes.

**IDLffXMLDOMNode** - Abstract class used as a superclass for other IDLffXMLDOM node classes.

**IDLffXMLDOMNodeIterator** - Allows iterative navigation of the XML DOM tree.

**IDLffXMLDOMNodeList** - Container for IDLffXMLDOM nodes.

**IDLffXMLDOMNotation** - Represents a notation in the XML DTD.

**IDLffXMLDOMProcessingInstruction** - Represents a processing instruction in an XML document.

**IDLffXMLDOMText** - References a text node in the XML document.

**IDLffXMLDOMTreeWalker** - Allows tree-walking navigation of the XML DOM tree.

**IDLffXMLSAX** - Represents an XML SAX level 2 parser.

**IDLgrMPEG** - Creates an MPEG movie file from an array of image frames.

**IDLgrVRML** - Saves the contents of an Object Graphics hierarchy into a VRML 2.0 format file.

## Graphic Objects—Destination

---

**IDL\_Container** - A container for other objects.

**IDLgrBuffer** - An in-memory, off-screen destination object.

**IDLgrClipboard** - A destination object representing the native clipboard.

**IDLgrPrinter** - Represents a hardcopy graphics destination.

**IDLgrWindow** - Represents an on-screen area on a display device that serves as a graphics destination.

## Graphic Objects—Display

---

**IDL\_Container** - Object that holds other objects.

**IDLgrModel** - Represents a graphical item or group of items that can be transformed (rotated, scaled, and/or translated).

**IDLgrScene** - Represents the entire scene to be drawn and serves as a container of IDLgrView or IDLgrViewgroup objects.

**IDLgrView** - Represents a rectangular area in which graphics objects are drawn. It is a container for objects of the IDLgrModel class.

**IDLgrViewgroup** - A simple container object that contains one or more IDLgrView objects. An IDLgrScene can contain one or more of these objects.

## Graphic Objects—Visualization

---

**IDLgrAxis** - Represents a single vector that may include a set of tick marks, tick labels, and a title.

**IDLgrColorbar** - Consists of a color-ramp with an optional framing box and annotation axis.

**IDLgrContour** - Draws a contour plot from data stored in a rectangular array or from a set of unstructured points.

**IDLgrFont** - Represents a typeface, style, weight, and point size that may be associated with text objects.

**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.

**IDLgrLegend** - Provides a simple interface for displaying a legend.

**IDLgrLight** - Represents a source of illumination for 3D graphic objects.

**IDLgrPalette** - Represents a color lookup table that maps indices to red, green, and blue values.

**IDLgrPattern** - Describes which pixels are filled and which are left blank when an area is filled.

**IDLgrPlot** - Creates set of polylines connecting data points in 2D space.

**IDLgrPolygon** - Represents one or more polygons that share a set of vertices and rendering attributes.

**IDLgrPolyline** - Represents one or more polylines that share a set of vertices and rendering attributes.

**IDLgrROI** - Object graphics representation of a region of interest.

**IDLgrROIGroup** - Object Graphics representation of a group of regions of interest.

**IDLgrSurface** - A shaded or vector representation of a mesh grid. No superclasses.

**IDLgrSymbol** - Represents a graphical element that is plotted relative to a particular position.

**IDLgrTessellator** - Converts a simple concave polygon (or a simple polygon with “holes”) into a number of simple convex polygons (general triangles).

**IDLgrText** - Represents one or more text strings that share common rendering attributes.

**IDLgrView** - Represents a rectangular area in which graphics objects are drawn. It is a container for objects of the IDLgrModel class.

**IDLgrViewgroup** - A simple container object that contains one or more IDLgrView objects. An IDLgrScene can contain one or more of these objects.

**IDLgrVolume** - Represents mapping from a 3D array of data to a 3D array of voxel colors, which, when drawn, are projected to two dimensions.

## iTools System Objects

---

**IDLitCommand** - The base functionality for the iTools command buffer system.

**IDLitCommandSet** - A container for IDLitCommand objects, which allows a group of commands to be managed as a single item.

**IDLitComponent** - A core or base component, from which all other components subclass.

**IDLitContainer** - A specialization of the IDL\_Container class that manages a collection of IDLitComponents and provides methods for working with the Identifier system of the iTools framework.

**IDLitData** - A generic data storage object that can hold any IDL data type available. It provides typing, metadata, and data change notification functionality. When coupled with IDLitDataContainer, it forms the element for the construction of composite data types.

**IDLitDataContainer** - A container for IDLitData and IDLitDataContainer objects. This container is used to form hierarchical data structures. Data and DataContainer objects can be added and removed to/from a DataContainer during program execution, allowing for dynamic creation of composite data types.

**IDLitDataOperation** - A subclass to IDLitOperation that automates data access and automatically records information for the undo-redo system.

**IDLitMessaging** - An interface providing common methods to send or trigger messaging and error actions, which may occur during execution.

**IDLitManipulator** - The base functionality of the iTools manipulator system.

**IDLitManipulatorContainer** - A container for IDLitManipulator objects, which allows for the construction of manipulator hierarchies. This container implements the concept of a current manipulator for the items it contains.

**IDLitManipulatorManager** - A specialization of the manipulator container (IDLitManipulatorContainer), which acts as the root of the manipulator hierarchy.

**IDLitManipulatorVisual** - The means for iTool developers to create visual elements associated with an interactive manipulator.

**IDLitOperation** - The basis for all iTool operations. It defines how an operation is executed and how information about the operation is recorded for the command transaction (undo-redo) system.

**IDLitParameter** - An interface providing parameter management methods to associate parameter names with IDLitData objects.

**IDLitParameterSet** - A specialized subclass of the IDLitDataContainer class. This class provides the ability to associate names with contained IDLitData objects.

**IDLitReader** - The definition of the interface and the process used to construct file readers for the iTools framework. When a new file reader is constructed for the iTools system, a new class is subclassed from this IDLitReader class.

**IDLitTool** - All the functionality provided by a particular instance of an IDL Intelligent Tool (iTool). This object provides the management systems for the underlying tool functionality.

**IDLitUI** - A link between the underlying functionality of an iTool and the IDL widget interface.

**IDLitVisualization** - The basis for all iTool visualizations. All visualization components subclass from this class.

**IDLitWindow** - The basis for all iTool windows. All iTool windows subclass from this class.

**IDLitWriter** - The definition of the interface and the process used to construct file writers for the iTools framework. When a new file writer is constructed for the iTools system, a new class is subclassed from this IDLitWriter class.

## Miscellaneous Objects

---

**IDL\_Container** - A container for other objects.

**IDL\_IDLBridge** - Creates and controls a synchronous or an asynchronous slave IDL process that can operate sequentially or in parallel with the main IDL process.

**IDL\_Savefile** - Provides complete query and restore capabilities for IDL SAVE files.

**IDLcomActiveX** - Creates an IDL object that encapsulates an ActiveX control.

**IDLcomDispatch** - Creates an IDL object that encapsulates a COM object.

**IDLjavaObject** - An IDL object encapsulating a Java object. IDL provides data type and other translation services, allowing IDL programs to access the Java object's methods and properties using standard IDL syntax.

**IDLsysMonitorInfo** - Contains information about the display monitor or monitors attached to your system. Use this object to obtain information about the monitor(s) when building applications that use multiple monitors.

**TrackBall** - Translates widget events from a draw widget into transformations that emulate a virtual trackball (for transforming object graphics in three dimensions).

## Network Objects

---

**IDLnetOGCWCS** - Provides the ability to query and return data from and Open Geospatial Consortium Web Coverage Service server.

**IDLnetOGCWMS** - Provides the ability to query and return data from and Open Geospatial Consortium Web Map Service server.

## Operating System Access

---

**APP\_USER\_DIR** - Provides access to the *application user directory*.

**APP\_USER\_DIR\_QUERY** - Locates existing *application user directories*.

**CALL\_EXTERNAL** - Calls a function in an external sharable object and returns a scalar value.

**CD** - Sets and/or changes the current working directory.

**FILE\_BASENAME** - Returns the *basename* of a *file path*.

**FILE\_CHMOD** - Changes file access permissions.

**FILE\_DELETE** - Deletes files and empty directories.

**FILE\_DIRNAME** - Returns the *dirname* of a *file path*.

**FILE\_EXPAND\_PATH** - Fully qualifies file and directory paths.

**FILE\_INFO** - Returns status information about a file.

**FILE\_MKDIR** - Creates directories.

**FILE\_SAME** - Determines whether two different file names refer to the same underlying file.

**FILE\_SEARCH** - Returns a string array containing the names of all files matching the input path specification.

**FILE\_TEST** - Test a file or directory for existence and other specific attributes.

**FILE\_WHICH** - Searches for a specified file in a directory search path.

**GET\_DRIVE\_LIST** (Windows only) - Returns string array of the names of valid drives/volumes for the file system.

**GET\_SCREEN\_SIZE** - Returns dimensions of the screen.

**GETENV** - Returns the value of an environment variable.

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

**PATH\_SEP** - Returns the proper file path segment separator character for the current operating system.

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

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

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

**SETENV** - Adds or changes an environment variable.

**SPAWN** - Spawns child process for access to operating system.

## Performance Testing

---

**MEMORY** - Provides information about the amount of dynamic memory currently in use by the IDL session.

**PROFILER** - Accesses the IDL Code Profiler used to analyze performance of applications.

**TIME\_TEST2** - Performs speed benchmarks for IDL.

## Plotting

---

**AXIS** - Draws an axis of the specified type and scale.

**BAR\_PLOT** - Creates a bar graph.

**ERRPLOT** - Plots error bars over a previously drawn plot.

**IDLgrAxis** - Displays a plot axis object that may include tick marks, text and a title.

**IDLgrContour** - Draws a contour plot from data stored in a rectangular array or from a set of unstructured points.

**IDLgrLegend** - Provides a simple interface for displaying a legend.

**IDLgrPlot** - Creates a set of polylines connecting data points in two-dimensional space.

**IDLgrSymbol** - Represents a graphical element that is plotted relative to a particular position.

**IPLLOT** - Creates an iTool and associated user interface (UI) configured to display and manipulate plot data.

**LABEL\_DATE** - Labels axes with dates. Use with [XYZ]TICKFORMAT keyword.

**OPLLOT** - Plots vector data over a previously-drawn plot.

**OPLOTERR** - Draws error bars over a previously drawn plot.

**PLOT** - Plots vector arguments as X versus Y graphs.

**PLOT\_3DBOX** - Plots function of two variables inside 3D box.

**PLOT\_FIELD** - Plots a 2D field using arrows.

**PLOTERR** - Plots individual data points with error bars.

**PLOTS** - Plots vectors and points.

**POLYFILL** - Fills the interior of a polygon.

**POLYFILLV** - Returns subscripts of pixels inside a polygon.

**PROFILE** - Extracts a profile from an image.

**PROFILES** - Interactively examines image profiles.

**THREED** - Plots a 2D array as a pseudo 3D plot.

**TRIANGULATE** - Constructs Delaunay triangulation of a planar set of points.

**TRIGRID** - Interpolates irregularly-gridded data to a regular grid from a triangulation.

**USERSYM** - Defines a new plotting symbol.

**VEL** - Draws a velocity (flow) field with streamlines.

**VELOVECT** - Draws a 2D velocity field plot.

**WF\_DRAW** - Draws weather fronts with smoothing.

**XPLOT3D** - Utility for creating and interactively manipulating 3D plots.

**XYOUTS** - Draws text on currently-selected graphics device.

## Programming and IDL Control

---

**APP\_USER\_DIR** - Provides access to the *application user directory*.

**APP\_USER\_DIR\_QUERY** - Locates existing *application user directories*.

**ARG\_PRESENT** - Returns TRUE if the value of the specified variable can be passed back to the caller.

**BIT\_FFS** - Returns the index of the first bit set (non-zero) in an integer.

**BIT\_POPULATION** - Returns the number of set (non-zero) bits in an integer.

**BREAKPOINT** - Sets and clears breakpoints for debugging.

**BYTEORDER** - Converts between host and network byte ordering.

**CALL\_FUNCTION** - Calls an IDL function.

**CALL\_METHOD** - Calls an IDL object method.

**CALL\_PROCEDURE** - Calls an IDL procedure.

**CATCH** - Declares and clears exception handlers.

**COMMAND\_LINE\_ARGS** - Returns string values supplied when the user starts IDL with the -arg or -args command line options.

- CPU** - Changes the values stored in the read-only !CPU system variable.
- CREATE\_STRUCT** - Creates and concatenates structures.
- COMPILE\_OPT** - Change default rules for compiling routines.
- DEFINE\_KEY** - Programs keyboard function keys.
- DEFINE\_MSGBLK** - Defines and loads a new message block into the current IDL session.
- DEFINE\_MSGBLK\_FROM\_FILE** - Reads the definition of a message block from a file, and loads it into the current IDL session.
- DEFSYSV** - Creates a new system variable.
- EXECUTE** - Compiles, executes IDL statements contained in a string.
- EXIT** - Quits IDL and exits back to the operating system.
- EXPAND\_PATH** - Expands path-definition string into full path name for use with the !PATH system variable.
- HEAP\_FREE** - Recursively frees all heap variables referenced by its input argument.
- HEAP\_GC** - Performs “garbage collection” on heap variables.
- HEAP\_NOSAVE** - Used to clear the *save attribute* of pointer or object heap variables.
- HEAP\_SAVE** - Used to query whether a pointer or object heap variable is savable. It can also be used to change the heap variable save attribute.
- IDL\_VALIDNAME** - Determines whether a string may be used as a valid IDL variable name or structure tag name.
- IDLITSYS\_CREATETOOL** - Creates an instance of the specified tool registered within the iTools system.
- ITCURRENT** - Set the current tool in the iTools system.
- ITDELETE** - Deletes a tool in the iTools system.
- ITGETCURRENT** - Gets the identifier of the current tool in the iTools system.
- ITREGISTER** - Registers tool object classes with the iTools system.
- ITRESET** - Resets the iTools session.
- ITRESOLVE** - Resolves all IDL code within the iTools directory, as well as all other IDL code required for the iTools framework.
- KEYWORD\_SET** - Returns True if given expression is defined and nonzero or an array.
- LMGR** - Determines the type of license used by the current IDL session.
- LOGICAL\_AND** - Performs a logical AND operation on its arguments.
- LOGICAL\_OR** - Performs a logical OR operation on its arguments.
- LOGICAL\_TRUE** - Determines whether its arguments are non-zero (or non-NULL).
- MESSAGE** - Issues error and informational messages.
- N\_ELEMENTS** - Returns the number of elements contained in an expression or variable.
- N\_PARAMS** - Returns the number of non-keyword parameters used in calling an IDL procedure or function.
- N\_TAGS** - Returns the number of tags in a structure.
- OBJ\_CLASS** - Determines the class name of an object.
- OBJ\_DESTROY** - Destroys an object reference.
- OBJ\_HASMETHOD** - Determines whether an object class implements or inherits a specified method or methods.
- OBJ\_ISA** - Determines inheritance relationship of an object.
- OBJ\_NEW** - Creates an object reference.
- OBJ\_VALID** - Verifies validity of object references.
- ON\_ERROR** - Designates the error recovery method.
- ON\_IOERROR** - Declares I/O error exception handler.
- PATH\_CACHE** - Controls IDL’s path caching mechanism.
- PREF\_COMMIT** - Commits IDL preferences in the pending state.
- PREF\_GET** - Returns information about IDL preferences.
- PREF\_MIGRATE** - Imports IDL preferences from other versions of IDL for use by the currently running version.
- PREF\_SET** - Sets new values for IDL preferences.
- PTR\_FREE** - Destroys a pointer.
- PTR\_NEW** - Creates a pointer.
- PTR\_VALID** - Verifies the validity of pointers.
- PTRARR** - Creates an array of pointers.
- RECALL\_COMMANDS** - Returns entries in IDL’s command recall buffer.
- REGISTER\_CURSOR** - Associates a given name with cursor information, used in conjunction with IDLgrWindow::SetCurrentCursor.
- RESOLVE\_ALL** - Compiles any uncompiled routines.
- RESOLVE\_ROUTINE** - Compiles a routine.
- RETALL** - Returns control to the main program level.
- RETURN** - Returns control to the next-higher program level.
- ROUTINE\_INFO** - Provides information about compiled procedures and functions.
- STOP** - Stops the execution of a running program or batch file.
- STRMESSAGE** - Returns the text of a given error number.
- STRUCT\_ASSIGN** - Uses “Relaxed Structure Assignment” to copy structures.
- STRUCT\_HIDE** - Prevents the IDL HELP procedure from displaying information about structures or objects.
- SWAP\_ENDIAN** - Reverses the byte ordering of scalars, arrays or structures.
- SWAP\_ENDIAN\_INPLACE** - Reverses the byte ordering of scalars, arrays or structures. Differs from the SWAP\_ENDIAN function in that it alters the input data in place rather than making a copy.
- TAG\_NAMES** - Returns the names of tags in a structure.
- TEMPORARY** - Returns a temporary copy of a variable, and sets the original variable to “undefined”.
- WAIT** - Suspends execution of an IDL program for a specified period.

## Query Routines

---

- GET\_LOGIN\_INFO** - Returns the name of the computer running the IDL process, as well as the user’s login name.
- QUERY\_ASCII** - Obtains information about an ASCII file.
- QUERY\_BMP** - Obtains information about a BMP image file.
- QUERY\_DICOM** - Tests file for compatibility with READ\_DICOM.
- QUERY\_GIF** - Obtains information about a GIF image file.
- QUERY\_IMAGE** - Determines if a file is recognized as an image file.

- QUERY\_JPEG** - Obtains information about a JPEG image file.
- QUERY\_JPEG2000** - Obtains information about a JPEG 2000 image file.
- QUERY\_MRSID** - Obtains information about a MrSID image file.
- QUERY\_PICT** - Obtains information about a PICT image file.
- QUERY\_PNG** - Obtains information about a PNG image file.
- QUERY\_PPM** - Obtains information about a PPM image file.
- QUERY\_SRF** - Obtains information about an SRF image file.
- QUERY\_TIFF** - Obtains information about a TIFF image file.
- QUERY\_WAV** - Checks that the file is actually a .WAV file and that the READ\_WAV function can read the data in the file.
- ROUTINE\_FILEPATH** - Returns the full path to a currently-compiled procedure or function (or a list of procedures and functions).

## Saving/Restoring a Session

---

- HEAP\_NOSAVE** - Used to clear the *save attribute* of pointer or object heap variables.
- HEAP\_SAVE** - Used to query whether a pointer or object heap variable is savable. It can also be used to change the heap variable save attribute.
- IDL\_Savefile** - Object that provides complete query and restore capabilities for IDL SAVE files.
- JOURNAL** - Logs IDL commands to a file.IDL.
- RESTORE** - Restores IDL variables and routines in an IDL SAVE file.
- SAVE** - Saves variables, system variables, and IDL routines in a file for later use.

## Scientific Data Formats

---

- CDF Routines** - see the [Scientific Data Formats](#) chapter of this manual in quickref.pdf, located on your IDL CD-ROM.
- EOS Routines** - see the [Scientific Data Formats](#) chapter of this manual in quickref.pdf, located on your IDL CD-ROM.
- H5\_BROWSER** - Opens a GUI to view contents of HDF5 files.
- HDF Routines** - see the [Scientific Data Formats](#) chapter of this manual in quickref.pdf, located on your IDL CD-ROM.
- HDF5 Routines** - see the [Scientific Data Formats](#) chapter of this manual in quickref.pdf, located on your IDL CD-ROM.
- HDF\_BROWSER** - Opens a GUI to view contents of HDF, HDF-EOS, or NetCDF file.
- HDF\_READ** - Extracts HDF, HDF-EOS, and NetCDF data and metadata into an output structure.
- NCDF Routines** - see the [Scientific Data Formats](#) chapter of this manual in quickref.pdf, located on your IDL CD-ROM.

## Scope Functions

---

- SCOPE\_LEVEL** - Returns the current routine's scope level.
- SCOPE\_TRACEBACK** - Returns the current interpreter call stack (the sequence of routine calls to the present point).

- SCOPE\_VARFETCH** - Returns variables outside the current routine's local scope.
- SCOPE\_VARNAME** - Returns the names of variables outside current routine's local scope.

## Signal Processing

---

- A\_CORRELATE** - Computes autocorrelation.
- BLK\_CON** - Convolves input signal with impulse-response sequence.
- BUTTERWORTH** - Returns the absolute value of the low-pass Butterworth kernel.
- CANNY** - Implements the Canny edge-detection algorithm.
- C\_CORRELATE** - Computes cross correlation.
- CONVOL** - Convolves two vectors or arrays.
- CORRELATE** - Computes the linear Pearson correlation.
- DIGITAL\_FILTER** - Calculates coefficients of a non-recursive, digital filter.
- FFT** - Returns the Fast Fourier Transform of an array.
- HANNING** - Creates Hanning and Hamming windows.
- HILBERT** - Constructs a Hilbert transform.
- INTERPOL** - Performs linear interpolation on vectors.
- IR\_FILTER** - Performs the infinite or finite impulse response filter on data.
- LEEFILT** - Performs the Lee filter algorithm on an image array.
- M\_CORRELATE** - Computes multiple correlation coefficient.
- MEDIAN** - Returns median value of an array or applies a median filter.
- P\_CORRELATE** - Computes partial correlation coefficient.
- R\_CORRELATE** - Computes rank correlation.
- SAVGOL** - Returns coefficients of Savitzky-Golay smoothing filter.
- SMOOTH** - Smooths with a boxcar average.
- TS\_COEF** - Computes the coefficients for autoregressive time-series.
- TS\_DIFF** - Computes the forward differences of a time-series.
- TS\_FCAST** - Computes future or past values of stationary time-series.
- TS\_SMOOTH** - Computes moving averages of a time-series.
- WTN** - Returns wavelet transform of the input array.

See Also - [Wavelet Toolkit](#)

## Statements

---

### Compound Statements

---

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

### Conditional Statements

---

- IF...THEN...ELSE** - Conditionally executes a statement or block of statements.
- CASE** - Selects one statement for execution, depending on the value of an expression.

**SWITCH** - Selects one statement for execution, depending upon the value of an expression.

## Loop Statements

---

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

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

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

## Jump Statements

---

**BREAK** - Exits from a loop (FOR, WHILE, REPEAT), CASE, or SWITCH statement.

**CONTINUE** - Starts the next iteration of the enclosing FOR, WHILE, or REPEAT loop.

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

## Functions and Procedures

---

**COMPILE\_OPT** - Gives IDL compiler information that changes the default rules for compiling functions or procedures.

**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).

**FUNCTION** - Defines a function.

**PRO** - Defines a procedure.

## Variable Scope

---

**COMMON** - Creates a common block.

## String Processing

---

**FILE\_BASENAME** - Returns the *basename* of a *file path*.

**FILE\_DIRNAME** - Returns the *dirname* of a *file path*.

**STRCMP** - Compares two strings.

**STRCOMPRESS** - Removes whitespace from a string.

**STREGEX** - Performs regular expression matching.

**STRING** - Converts arguments to string type.

**STRJOIN** - Collapses a string scalar or array into merged strings.

**STRLEN** - Returns the length of a string.

**STRLOWCASE** - Converts a string to lower case.

**STRMATCH** - Compares search string against input string expression.

**STRMID** - Extracts a substring from a string.

**STRPOS** - Finds first occurrence of a substring within a string.

**STRPUT** - Inserts the contents of one string into another.

**STRSPLIT** - Splits its input string argument into separate substrings, according to the specified pattern.

**STRTRIM** - Removes leading and/or trailing blanks from string.

**STRUPCASE** - Converts a string to upper case.

## Structures

---

**IDL\_VALIDNAME** - Determines whether a string may be used as a valid IDL variable name or structure tag name.

**N\_TAGS** - Returns the number of tags in a structure.

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

**STRUCT\_ASSIGN** - Uses "Relaxed Structure Assignment" to copy structures.

**STRUCT\_HIDE** - Prevents the IDL HELP procedure from displaying information about structures or objects.

**TAG\_NAMES** - Returns the names of tags in a structure.

## Type Conversion

---

**BYTE** - Converts argument to byte type.

**COMPLEX** - Converts argument to complex type.

**DCOMPLEX** - Converts argument to double-precision complex type.

**DOUBLE** - Converts argument to double-precision type.

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

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

**LONG** - Converts argument to longword integer type.

**LONG64** - Converts argument to 64-bit integer type.

**STRING** - Converts argument to string type.

**UINT** - Converts argument to unsigned integer type.

**ULONG** - Converts argument to unsigned longword integer type.

**ULONG64** - Converts argument to unsigned 64-bit integer type.

## Utilities

---

**EFONT** - Interactive vector font editor and display tool.

**IDLEXBR\_ASSISTANT** - Launches the Export Bridge Assistant.

**SLIDE\_IMAGE** - Creates a scrolling graphics window for examining large images.

**MAKE\_RT** - Creates a stand-alone IDL runtime distribution for one or more platforms.

**XBM\_EDIT** - Creates, edits bitmap icons for IDL widget button labels.

**XDISPLAYFILE** - Displays ASCII text file in scrolling text widget.

**XDXF** - Utility to display and interactively manipulate DXF objects.

**XFONT** - Creates modal widget to select and view an X Windows font.

**XINTERANIMATE** - Displays animated sequence of images.

**XLOADCT** - Displays a tool for selecting and setting the current color table.

**XMTOOL** - Displays a tool for viewing XMANAGER widgets.

**XOBJVIEW** - Displays object viewer widget.

**XOBJVIEW\_ROTATE** - Programmatically rotate the object currently displayed in XOBJVIEW.

**XOBJVIEW\_WRITE\_IMAGE** - Write the object currently displayed in XOBJVIEW to an image file.

**XPALETTE** - Displays a tool for creating and modifying color tables.

**XPCOLOR** - Adjusts the value of the current foreground plotting color, !P.COLOR.

**XPLOT3D** - Utility for creating and interactively manipulating 3D plots.

**XROI** - Utility for interactively defining and obtaining information about regions of interest.

**XSURFACE** - Provides a graphical interface to the SURFACE and SHADE\_SURF commands.

**XVAREEDIT** - Utility for editing any IDL variable.

**XVOLUME** - Utility for viewing and interactively manipulating volumes and isosurfaces.

**XVOLUME\_ROTATE** - Utility for rotating a volume displayed in XVOLUME.

**XVOLUME\_WRITE\_IMAGE** - Utility for writing a volume displayed in XVOLUME to an image file.

## Wavelet Toolkit

---

### Widget Commands and Visualization Tools

---

**WV\_APPLET** - Runs the IDL Wavelet Toolkit GUI.

**WV\_CW\_WAVELET** - Compound widget used to select and display wavelet functions.

**WV\_IMPORT\_DATA** - Allows user to add a variable to the currently active WV\_APPLET widget from the IDL> command prompt.

**WV\_IMPORT\_WAVELET** - Allows user to add wavelet functions to the IDL Wavelet Toolkit.

**WV\_PLOT3D\_WPS** - Runs the GUI for 3D visualization of the wavelet power spectrum.

**WV\_PLOT\_MULTIRESES** - Runs GUI for multiresolution analysis.

**WV\_TOOL\_DENOISE** - Runs the GUI for wavelet filtering and denoising.

### Wavelet Transform

---

**WV\_CWT** - Returns the one-dimensional continuous wavelet transform of the input array.

**WV\_DENOISE** - Uses the wavelet transform to filter (or de-noise) a multi-dimensional array.

**WV\_DWT** - Returns the multi-dimensional discrete wavelet transform of the input array.

**WV\_PWT** - Returns the partial wavelet transform of the input vector.

### Wavelet Functions

---

**WV\_FN\_COIFLET** - Constructs wavelet coefficients for the coiflet wavelet function.

**WV\_FN\_DAUBECHIES** - Constructs wavelet coefficients for the Daubechies wavelet function.

**WV\_FN\_GAUSSIAN** - Constructs wavelet coefficients for the Gaussian wavelet function.

**WV\_FN\_HAAR** - Constructs wavelet coefficients for the Haar wavelet function.

**WV\_FN\_MORLET** - Constructs wavelet coefficients for the Morlet wavelet function.

**WV\_FN\_PAUL** - Constructs wavelet coefficients for the Paul wavelet function.

**WV\_FN\_SYMLET** - Constructs wavelet coefficients for the symlet wavelet function.

## Widget Routines

---

**WIDGET\_ACTIVEX** - Create an ActiveX control and place it into an IDL widget hierarchy.

**WIDGET\_BASE** - Creates base widget (containers for other widgets).

**WIDGET\_BUTTON** - Creates button widgets.

**WIDGET\_COMBOBOX** - Creates editable droplist widgets.

**WIDGET\_CONTROL** - Realizes, manages, and destroys widgets.

**WIDGET\_DISPLAYCONTEXTMENU** - Displays a context-sensitive menu.

**WIDGET\_DRAW** - Creates drawable widgets.

**WIDGET\_DROPLIST** - Creates droplist widgets.

**WIDGET\_EVENT** - Returns events for the widget hierarchy.

**WIDGET\_INFO** - Obtains information about widgets.

**WIDGET\_LABEL** - Creates label widgets.

**WIDGET\_LIST** - Creates list widgets.

**WIDGET\_PROPERTY SHEET** - Creates a property sheet widget, which exposes the properties of an IDL object in a graphical interface. This widget transparently handles property value changes.

**WIDGET\_SLIDER** - Creates slider widgets.

**WIDGET\_TAB** - Creates tab widgets.

**WIDGET\_TABLE** - Creates table widgets.

**WIDGET\_TEXT** - Creates text widgets.

**WIDGET\_TREE** - Creates tree widgets.

**WIDGET\_TREE\_MOVE** - Copies and moves tree widget nodes from one tree to another or within the same tree.

**XMANAGER** - Provides event loop manager for IDL widgets.

**XMNG\_TMPL** - Template for creating widgets.

**XMTOOL** - Displays tool for viewing XMANAGER widgets.

**XREGISTERED** - Returns registration status of a given widget.

## Widget Routines, Compound

---

**CW\_TMPL** - Template for compound widgets that use XMANAGER.



## Animation

---

**CW\_ANIMATE** - Creates a compound widget for animation.

**CW\_ANIMATE\_GETP** - Gets pixmap window IDs used by CW\_ANIMATE.

**CW\_ANIMATE\_LOAD** - Loads images into CW\_ANIMATE.

**CW\_ANIMATE\_RUN** - Displays images loaded into CW\_ANIMATE.

## Color Manipulation

---

**CW\_CLR\_INDEX** - Creates compound widget for the selection of a color index.

**CW\_COLORSEL** - Creates compound widget that displays all colors in current colormap.

**CW\_PALETTE\_EDITOR\_SET** - Sets the CW\_LIGHT\_EDITOR properties.

**CW\_PALETTE\_EDITOR** - Creates compound widget to display and edit color palettes.

**CW\_PALETTE\_EDITOR\_GET** - Gets the CW\_PALETTE\_EDITOR properties.

**CW\_PALETTE\_EDITOR\_SET** - Sets the CW\_PALETTE\_EDITOR properties.

**CW\_RGBSLIDER** - Creates compound widget with sliders for adjusting RGB color values.

## Data Entry and Display

---

**CW\_FIELD** - Creates a widget data entry field.

**CW\_FILESEL** - Creates compound widget for file selection.

**CW\_FORM** - Creates compound widget for creating forms.

## Image/Data Manipulation

---

**CW\_DEFROI** - Creates compound widget used to define region of interest.

**CW\_LIGHT\_EDITOR** - Creates compound widget to edit properties of existing IDLgrLight objects in a view.

**CW\_LIGHT\_EDITOR\_GET** - Gets the CW\_LIGHT\_EDITOR properties.

**CW\_ZOOM** - Creates widget for displaying zoomed images.

## Orientation

---

**CW\_ARCBALL** - Creates compound widget for intuitively specifying 3D orientations.

**CW\_ORIENT** - Creates compound widget used to interactively adjust the 3D drawing transformation.

## User Interface

---

**CW\_BGROUP** - Creates button group for use as a menu.

**CW\_FSLIDER** - Creates slider that selects floating-point values.

**CW\_PDMENU** - Creates widget pulldown menus.

## Window Routines

---

**IDLgrWindow** - Represents an on-screen area on a display device that serves as a graphics destination.

**WDELETE** - Deletes IDL graphics windows.

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

**WSET** - Selects the current window.

**WSHOW** - Exposes or hides the designated window.





# Alphabetical List of IDL Routines

This quick reference guide contains an alphabetical listing of all IDL routines. The alphabetical listing contains all functions, procedures, statements, and objects, including the syntax of each.

# IDL Syntax Conventions

**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.
<i>Italics</i>	Indicates arguments, expressions, or statements for which you must provide values.
{ } (Braces)	<ul style="list-style-type: none"> <li>• Indicates that you must choose one of the values they contain</li> <li>• Encloses a list of possible values, separated by vertical lines (   )</li> <li>• Encloses useful information about a keyword</li> <li>• Defines an IDL structure (this is the only case in which the braces are included in the statement).</li> </ul>
(Vertical lines)	Separates multiple values or keywords.
[, <i>Value</i> <sub>1</sub> , ... , <i>Value</i> <sub><i>n</i></sub> ]	Indicates that any number of values can be specified.
[, <i>Value</i> <sub>1</sub> , ... , <i>Value</i> <sub>8</sub> ]	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*<sub>1</sub>, ... , *Value*<sub>*n*</sub>] is listed, you can specify any number of arguments. When an explicit number is listed, as in [, *Value*<sub>1</sub>, ... , *Value*<sub>8</sub>], 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:

```
READ_JPEG [, TRUE={1 | 2 | 3}]
```

In this example, you must choose either 1, 2, or 3. An example of a valid statement is:

```
READ_JPEG, TRUE=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 nth 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. Note, however that when a keyword calls for a named variable, only a named variable can be used—sending an expression causes an error.

For example, the WIDGET\_CONTROL procedure can return the user values of widgets in a named variable using the GET\_UVALUE keyword. To return the user value for a widget ID (contained in the variable mywidget) in the variable userval, you would use the command:

```
WIDGET_CONTROL, mywidget, GET_UVALUE = userval
```

Upon return from the procedure, userval contains the user value. Note that userval did not have to be defined before the call to WIDGET\_CONTROL.

- 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 6.3.

## A

---

**A\_CORRELATE** - Computes autocorrelation.

*Result* = A\_CORRELATE(*X*, *Lag* [, /COVARIANCE] [, /DOUBLE] )

**ABS** - Returns the absolute value of *X*.

*Result* = ABS(*X* [, *Thread pool keywords*])

**ACOS** - Returns the arc-cosine of *X*.

*Result* = ACOS(*X* [, *Thread pool keywords*])

**ADAPT\_HIST\_EQUAL** - Performs adaptive histogram equalization

*Result* = ADAPT\_HIST\_EQUAL (*Image* [, CLIP=*value*] [, FCN=*vector*] [, NREGIONS=*nregions*] [, TOP=*value*] )

**ALOG** - Returns the natural logarithm of *X*.

*Result* = ALOG(*X* [, *Thread pool keywords*])

**ALOG10** - Returns the logarithm to the base 10 of *X*.

*Result* = ALOG10(*X* [, *Thread pool keywords*])

**AMOEBAS** - Minimizes a function using downhill simplex method.

*Result* = AMOEBAS( *Ftol* [, FUNCTION\_NAME=*string*] [, FUNCTION\_VALUE=*variable*] [, NCALLS=*value*] [, NMAX=*value*] [, PO=*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]

**APP\_USER\_DIR** - Provides access to the *application user directory*.

*Result* = APP\_USER\_DIR(*AuthorDirname*, *AuthorDesc*, *AppDirname*, *AppDesc*, *AppReadmeText*, *AppReadmeVersion* [, AUTHOR\_README\_TEXT=*string array*] [, AUTHOR\_README\_VERSION=*string*] [, RESTRICT\_APPVERSION=*string*] [, /RESTRICT\_ARCH] [, /RESTRICT\_FAMILY | /RESTRICT\_OS] [, /RESTRICT\_FILE\_OFFSET\_BITS] [, /RESTRICT\_IDL\_RELEASE] [, /RESTRICT\_MEMORY\_BITS] )

**APP\_USER\_DIR\_QUERY** - Allows searches for *application user directories*.

*Result* = APP\_USER\_DIR\_QUERY(*AuthorDirname*, *AppDirname* [, COUNT=*variable*] [, /EXCLUDE\_CURRENT] [RESTRICT keywords] [QUERY keywords] )

**ARG\_PRESENT** - Returns TRUE if the value of the specified variable can be passed back to the caller.

*Result* = ARG\_PRESENT(*Variable*)

**ARRAY\_EQUAL** - Provides a fast way to compare data for equality in situations where the index of the elements that differ are not of interest.

*Result* = ARRAY\_EQUAL( *Op1* , *Op2* [, /NO\_TYPECONV ] )

**ARRAY\_INDICES** - Converts one-dimensional subscripts of an array into corresponding multi-dimensional subscripts.

*Result* = ARRAY\_INDICES(*Array*, *Index* [, /DIMENSIONS] )

**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* [, *Thread pool keywords*])

**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* [, /PHASE] [, *Thread pool keywords*])  
or  
*Result* = ATAN(*Y*, *X*) [, *Thread pool keywords*])

**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 or a vector of*  
*strings*] [ , {X | Y | Z}TICKINTERVAL=*value*]  
 [ , {X | Y | Z}TICKLAYOUT=*scalar*]  
 [ , {X | Y | Z}TICKLEN=*value*]  
 [ , {X | Y | Z}TICKNAME=*string\_array*]  
 [ , {X | Y | Z}TICKS=*integer*]  
 [ , {X | Y | Z}TICKUNITS=*string or a vector of strings*]  
 [ , {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 | ENDIF | ENDELSE | ENDFOR | ENDREP |  

  ENDWHILE
```

**BESELI** - Returns the I Bessel function of order *N* for *X*.

```
Result = BESELI(X, N [, /DOUBLE] [, ITER=variable])
```

**BESELJ** - Returns the J Bessel function of order *N* for *X*.

```
Result = BESELJ(X, N [, /DOUBLE] [, ITER=variable])
```

**BESELK** - Returns the K Bessel function of order *N* for the *X*.

```
Result = BESELK(X, N [, /DOUBLE] [, ITER=variable])
```

**BESELY** - Returns the Y Bessel function of order *N* for *X*.

```
Result = BESELY(X, N [, /DOUBLE] [, ITER=variable])
```

**BETA** - Returns the value of the beta function.

```
Result = BETA( Z, W [, /DOUBLE] )
```

**BILINEAR** - Computes array using bilinear interpolation.

```
Result = BILINEAR(P, IX, JY [, MISSING=value] )
```

**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=variable] )
```

**BINDGEN** - Returns byte array with each element set to its subscript.

```
Result = BINDGEN(D1 [, ..., Dg] [, Thread pool keywords])
```

**BINOMIAL** - Computes binomial distribution function.

```
Result = BINOMIAL(V, N, P [, /DOUBLE]  

  [ , /GAUSSIAN] )
```

**BIT\_FFS** - Returns the index of the first bit set (non-zero) in an integer.

```
Result = BIT_FFS( Value )
```

**BIT\_POPULATION** - Returns the number of set (non-zero) bits in an integer.

```
Result = BIT_POPULATION( Value )
```

**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]  

  [ , /DOUBLE] )
```

**BOX\_CURSOR** - Emulates the operation of a variable-sized box cursor.

```
BOX_CURSOR, [ X0, Y0, NX, NY [ , /INIT]  

  [ , /FIXED_SIZE]] [ , /MESSAGE]
```

**BREAK** - Immediately exits from a loop (FOR, WHILE, REPEAT), CASE, or SWITCH statement.

```
BREAK
```

**BREAKPOINT** - Sets and clears breakpoints for debugging.

```
BREAKPOINT [ , File], Index [ , AFTER=integer]  

  [ , /CLEAR] [ , CONDITION='expression'] [ , /DISABLE]  

  [ , /ENABLE] [ , /ON_RECOMPILE] [ , /ONCE] [ , /SET]
```

**BROYDEN** - Solves nonlinear equations using Broyden's method.

```
Result = BROYDEN( X, Vecfunc [ , CHECK=variable]  

  [ , /DOUBLE] [ , EPS=value] [ , ITMAX=value]  

  [ , STEPMAX=value] [ , TOLF=value]  

  [ , TOLMIN=value] [ , TOLX=value] )
```



**BUTTERWORTH** - Returns an array that contains the absolute value of the low-pass Butterworth kernel.

*Result* = BUTTERWORTH(*X* [, *Y*] [, *Z*]  
[, CUTOFF=*value*] [, ORDER=*value*] [, /ORIGIN]  
[, XDIM=*value*] [, YDIM=*value*] [, ZDIM=*value*]

**BYTARR** - Creates a byte vector or array.

*Result* = BYTARR(*D<sub>1</sub>* [, ..., *D<sub>g</sub>*] [, /NOZERO])

**BYTE** - Converts argument to byte type.

*Result* = BYTE(*Expression* [, *Offset*] [, *D<sub>1</sub>* [, ..., *D<sub>g</sub>*]]  
[, *Thread pool keywords*])

**BYTEORDER** - Converts between host and network byte ordering.

BYTEORDER, *Variable<sub>1</sub>*, ..., *Variable<sub>n</sub>* [, /D<sub>TOVAX</sub>]  
[, /D<sub>TOXDR</sub>] [, /F<sub>TOVAX</sub>] [, /F<sub>TOXDR</sub>] [, /H<sub>TONL</sub>]  
[, /H<sub>TONS</sub>] [, /L<sub>64SWAP</sub>] [, /L<sub>SWAP</sub>] [, /N<sub>TOHL</sub>]  
[, /N<sub>TOHS</sub>] [, /S<sub>SWAP</sub>] [, /S<sub>WAP\_IF\_BIG\_ENDIAN</sub>]  
[, /S<sub>WAP\_IF\_LITTLE\_ENDIAN</sub>] [, /V<sub>AXTOD</sub>]  
[, /V<sub>AXTOF</sub>] [, /X<sub>DRTOD</sub>] [, /X<sub>DRTOF</sub>] [, *Thread pool keywords*]

**BYTSCL** - Scales all values of an array into range of bytes.

*Result* = BYTSCL(*Array* [, *MAX=**value*] [, *MIN=**value*]  
[, /NAN] [, *TOP=**value*] [, *Thread pool keywords*])

## 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<sub>0</sub>*, ..., *P<sub>N-1</sub>*]  
[, /ALL\_VALUE] [, /B\_VALUE] [, /D\_VALUE]  
[, /F\_VALUE] [, /I\_VALUE] [, /L<sub>64</sub>\_VALUE] [, /S\_VALUE]  
[, /UL\_VALUE] [, /UL\_VALUE] [, /UL<sub>64</sub>\_VALUE]  
[, /CDECL] [, RETURN\_TYPE=*value*] [, /UNLOAD]  
[, VALUE=*byte\_array*]  
[, WRITE\_WRAPPER=*wrapper\_file*])

**Auto Glue keywords:** [, /AUTO\_GLUE] [, CC=*string*]  
[, COMPILE\_DIRECTORY=*string*] [,

EXTRA\_CFLAGS=*string*] [, EXTRA\_LFLAGS=*string*]  
[, /IGNORE\_EXISTING\_GLUE] [, LD=*string*]  
[, /NOCLEANUP] [, /SHOW\_ALL\_OUTPUT]  
[, /VERBOSE]

**CALL\_FUNCTION** - Calls an IDL function.

*Result* = CALL\_FUNCTION(*Name* [, *P<sub>1</sub>*, ..., *P<sub>n</sub>*])

**CALL\_METHOD** - Calls an IDL object method.

CALL\_METHOD, *Name*, *ObjRef*, [, *P<sub>1</sub>*, ..., *P<sub>n</sub>*] or  
*Result* = CALL\_METHOD(*Name*, *ObjRef*, [, *P<sub>1</sub>*, ..., *P<sub>n</sub>*])

**CALL\_PROCEDURE** - Calls an IDL procedure.

CALL\_PROCEDURE, *Name* [, *P<sub>1</sub>*, ..., *P<sub>n</sub>*]

**CANNY** - Implements the Canny edge-detection algorithm.

*Result* = CANNY(*Image* [, HIGH=*value*] [, LOW=*value*]  
[, SIGMA=*value*])

**CASE** - Selects one statement for execution, depending on the value of an expression.

CASE *expression* OF  
*expression*: *statement*  
...  
*expression*: *statement*  
[ ELSE: *statement* ]  
ENDCASE

**CATCH** - Declares and clears exception handlers.

CATCH, [*Variable*] [, /CANCEL]

**CD** - Sets and/or changes the current working directory.

CD [, *Directory*] [, CURRENT=*variable*]

**CDF\_\* Routines** - See "CDF Routines" on page 132 the quick-ref.pdf located on your IDL CD-ROM.

**CEIL** - Returns the closest integer greater than or equal to *X*.

*Result* = CEIL(*X* [, /L<sub>64</sub>] [, *Thread pool keywords*])

**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<sub>1</sub>* [, ..., *D<sub>g</sub>*] [, *Thread pool keywords*])

**CIR\_3PNT** - Returns radius and center of circle, given 3 points.

CIR\_3PNT, *X*, *Y*, *R*, *X<sub>0</sub>*, *Y<sub>0</sub>*

**CLOSE** - Closes the specified files.

CLOSE[, *Unit<sub>1</sub>*, ..., *Unit<sub>n</sub>*] [, /ALL]  
[, EXIT\_STATUS=*variable*] [, /FILE] [, /FORCE]

**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] )
```

**CLUSTER\_TREE** - Computes the hierarchical clustering for a set of  $m$  items in an  $n$ -dimensional space.

```
Result = CLUSTER_TREE( Pairedistance, Linkdistance
  [, LINKAGE = value] )
```

or for LINKAGE = 3 (centroid):

```
Result = CLUSTER_TREE( Pairedistance, Linkdistance,
  LINKAGE = 3, DATA = array[, MEASURE=value]
  [, POWER_MEASURE=value] )
```

**CMYK\_CONVERT** - Converts color triples to and from RGB and CMYK.

```
CMYK_CONVERT, C, M, Y, K, R, G, B, [, /TO_CMYK]
```

**COLOR\_CONVERT** - Converts colors to and from the RGB (Red Green Blue) and a number of other color spaces.

```
COLOR_CONVERT, I0, I1, I2, O0, O1, O2 [, /HLS_RGB |
  , /HSV_RGB | , /YUV_RGB | , /YIQ_RGB |
  , /YPBPR_RGB | , /YCBCR_RGB | , /RGB_HLS |
  , /RGB_HSV | , /RGB_YUV | , /RGB_YIQ |
  , /RGB_YPBR | /RGB_YCBCR]
  [, INTERLEAVE=value]
```

or

```
COLOR_CONVERT, I0, O0 [, /HLS_RGB | , /HSV_RGB |
  , /YUV_RGB | , /YIQ_RGB | , /YPBPR_RGB |
  , /YCBCR_RGB | , /RGB_HLS | , /RGB_HSV |
  , /RGB_YUV | , /RGB_YIQ | , /RGB_YPBR |
  , /RGB_YCBCR] [, INTERLEAVE=value]
```

**COLOR\_EXCHANGE** - Replaces image pixels of a given color with pixels of a new color.

```
Result = COLOR_EXCHANGE( Image, Color,
  ReplaceColor [, THRESHOLD=vector] )
```

**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]
```

**COLOR\_RANGE\_MAP** - Maps all the pixels of an image to another set of pixels, using source and target ranges to control the mapping.

```
Result = COLOR_RANGE_MAP( Image, FromColor1,
  FromColor2, ToColor1, ToColor2)
```

**COLORIZE\_SAMPLE** - Colorizes a greyscale image by matching luminance levels with an RGB sample table.

```
Result = COLORIZE_SAMPLE( Image, Sample)
```

**COLORMAP\_APPLICABLE** - Determines whether the current visual class supports the use of a colormap.

```
Result = COLORMAP_APPLICABLE( redrawRequired )
```

**COLORMAP\_GRADIENT** - Maps an image into a specified luminance-based gradient.

```
Result = COLORMAP_GRADIENT( Image[, Gradient] )
```

**COLORMAP\_ROTATION** - Maps pixels within a given hue range to another hue range, using the HSV hue component.

```
Result = COLORMAP_ROTATION( Image, SrcAngleStart,
  SrcAngleStop, DstAngleStart, DstAngleStop
  [, /SOURCE_CW] [, /DEST_CW] )
```

**COMFIT** - Fits paired data using one of six common filtering functions.

```
Result = COMFIT( X, Y, A {, /DOUBLE |
  , /EXPONENTIAL | , /GEOMETRIC | , /GOMPRTZ |
  , /HYPERBOLIC | , /ITMAX | , /LOGISTIC |
  , /LOGSQUARE | , /TOL=value} [, CHISQ=variable]
  [, ITER=variable] [, SIGMA=variable]
  [, STATUS=variable] [, WEIGHTS=vector]
  [, YERROR=variable] [, YFIT=variable] )
```

**COMMAND\_LINE\_ARGS** - returns string values supplied when the user starts IDL with the -arg or -args command line options.

```
Result = COMMAND_LINE_ARGS( [COUNT=variable] )
```

**COMMON** - Creates a common block.

```
COMMON Block_Name, Variable1, ..., Variablen
```

**COMPILE\_OPT** - Gives IDL compiler information that changes the default rules for compiling functions or procedures.

```
COMPILE_OPT opt1 [, opt2, ..., optn]
```

**Note:**  $opt_n$  can be IDL2, DEFINT32, HIDDEN, LOGICAL\_PREDICATE, OBSOLETE, STRICTARR, or STRICTARRSUBS

**COMPLEX** - Converts argument to complex type.

```
Result = COMPLEX( Real [, Imaginary] [, /DOUBLE]
  [, Thread pool keywords] )
```

or

```
Result = COMPLEX( Expression, Offset, D1 [, ..., Dg]
  [, /DOUBLE] [, Thread pool keywords] )
```

**COMPLEXARR** - Creates a complex, single-precision, floating-point vector or array.

```
Result = COMPLEXARR( D1 [, ..., Dg] [, /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] [, LNORM={0 | 1 | 2}])
```

**CONGRID** - Resamples an image to any dimensions.

*Result* = CONGRID(*Array*, *X*, *Y*, *Z* [, /CENTER]  
[, CUBIC=*value*{-1 to 0}] [, /INTERP]  
[, /MINUS\_ONE])

**CONJ** - Returns the complex conjugate of *X*.

*Result* = CONJ(*X* [, *Thread pool keywords*])

**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*]

**CONTINUE** - Immediately starts the next iteration of the enclosing FOR, WHILE, or REPEAT loop.

CONTINUE

**CONTOUR** - Draws a contour plot.

CONTOUR, *Z* [, *X*, *Y*] [, C\_CHARSIZE=*value*]  
[, C\_CHARTHICK=*integer*] [, C\_COLORS=*vector*]  
[, C\_LABELS=*vector*{each element 0 or 1}]  
[, C\_LINETYPE=*vector*] [, /FILL | , /CELL\_FILL] |  
[, C\_ANNOTATION=*vector\_of\_strings*]  
[, C\_ORIENTATION=*degrees*] [, C\_SPACING=*value*]]  
[, C\_THICK=*vector*] [, /CLOSED] [, /DOWNHILL]  
[, /FOLLOW] [, /IRREGULAR] [, /ISOTROPIC]  
[, 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*]  
[, /PATH\_DOUBLE] [, /XLOG] [, /YLOG]  
[, ZAXIS={0 | 1 | 2 | 3 | 4}] [, /ZLOG]

**Graphics Keywords:** Accepts all PLOT graphics keywords except for: LINETYPE, PSYM, SYMSIZE.

**CONVERT\_COORD** - Transforms coordinates to and from the coordinate systems supported by IDL.

*Result* = CONVERT\_COORD(*X* [, *Y* [, *Z*]] [, /DATA | ,  
/DEVICE | , /NORMAL] [, /DOUBLE] [, /T3D]  
[, /TO\_DATA | , /TO\_DEVICE | , /TO\_NORMAL] )

**CONVOL** - Convolves two vectors or arrays.

*Result* = CONVOL(*Array*, *Kernel* [, *Scale\_Factor*]  
[, BIAS=*value*] [, /CENTER] [, /EDGE\_WRAP]  
[, /EDGE\_TRUNCATE] [, /EDGE\_ZERO]  
[, INVALID=*value*] [, MISSING=*value*] [, /NAN]  
[, /NORMALIZE] [, *Thread pool keywords*])

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

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

**COPY\_LUN** - Copies data between two open files.

COPY\_LUN, *FromUnit*, *ToUnit* [, *Num*] [, /EOF]  
[, /LINES] [, TRANSFER\_COUNT=*va;ie*]

**CORRELATE** - Computes the linear Pearson correlation.

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

**COS** - Returns the cosine of *X*.

*Result* = COS(*X* [, *Thread pool keywords*])

**COSH** - Returns the hyperbolic cosine of *X*.

*Result* = COSH(*X* [, *Thread pool keywords*])

**CPU** - Changes the values stored in the read-only !CPU system variable.

CPU [, /RESET] [, RESTORE=*structure*]  
[, TPOOL\_MAX\_ELTS = *NumMaxElts*]  
[, TPOOL\_MIN\_ELTS = *NumMinElts*]  
[, TPOOL\_NTHREADS = *NumThreads*]  
[, /VECTOR\_ENABLE]

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

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

**CREATE\_CURSOR** - Returns a 16x16 image suitable for a cursor from the input string array.

*Result* = CREATE\_CURSOR(*StringArray*  
[, HOTSPOT=*variable*] [, MASK=*variable*])

**CREATE\_STRUCT** - Creates and concatenates structures.

*Result* = CREATE\_STRUCT( [*Tag*<sub>1</sub>, *Value*<sub>1</sub>, ..., *Tag*<sub>*n*</sub>,  
*Value*<sub>*n*</sub>] [, NAME=*string*] )  
or  
*Result* = CREATE\_STRUCT( [*Tag*<sub>1</sub>, ..., *Tag*<sub>*n*</sub>, *Value*<sub>1</sub>, ...,  
*Value*<sub>*n*</sub>] [, NAME=*string*] )

**CREATE\_VIEW** - Sets up 3D transformations.

CREATE\_VIEW [, AX=*value*] [, AY=*value*] [, AZ=*value*]  
[, PERSP=*value*] [, /RADIAN] [, 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 |  
/NOWAIT | , /UP | , /WAIT]] [, /DATA | , /DEVICE | ,  
/NORMAL]

**CURVEFIT** - Fits multivariate data with a user-supplied function.

```
Result = CURVEFIT( X, Y, Weights, A [, Sigma]
  [, CHISQ=variable] [, /DOUBLE] [, FITA=vector]
  [, FUNCTION_NAME=string] [, ITER=variable]
  [, ITMAX=value] [, /NODERIVATIVE]
  [, STATUS={0 | 1 | 2}] [, TOL=value]
  [, YERROR=variable] )
```

**CV\_COORD** - Converts 2D and 3D coordinates between coordinate systems.

```
Result = CV_COORD( [, /DEGREES] [, /DOUBLE]
  [, 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
  [, MPEG_BITRATE=value] [, MPEG_FORMAT=value]
  [, MPEG_QUALITY=value] [, /NO_KILL]
  [, OPEN_FUNC=string] [, PIXMAPS=vector]
  [, TAB_MODE=value] [, /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] [, TAB_MODE=value] [, /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_FUNC=string] [{, /EXCLUSIVE | ,
  /NONEXCLUSIVE} | [, SPACE=pixels]
```

```
  [, XPAD=pixels] [, YPAD=pixels]] [, FONT=font]
  [, FRAME=width] [, IDS=variable]
  [, LABEL_LEFT=string | , LABEL_TOP=string]
  [, /MAP] [, /NO_RELEASE] [, /RETURN_ID | ,
  /RETURN_INDEX | , /RETURN_NAME]
  [, ROW=value] [, /SCROLL] [, SET_VALUE=value]
  [, SPACE=value] [, TAB_MODE=value]
  [, 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 to select color index.

```
Result = CW_CLR_INDEX( Parent
  [, COLOR_VALUES=vector | [, NCOLORS=value]
  [, START_COLOR=value]
  [, EVENT_FUNC='function_name'] [, /FRAME]
```

**CW\_CLR\_INDEX** - continued

```
  [, LABEL=string] [, NCOLORS=value]
  [, START_COLOR=value] [, TAB_MODE=value]
  [, UNAME=string] [, UVALUE=value]
  [, VALUE=value] [, XSIZE=pixels] [, YSIZE=pixels] )
```

**CW\_COLORSEL** - Creates compound widget that displays all colors in current colormap.

```
Result = CW_COLORSEL( Parent [, /FRAME]
  [, TAB_MODE=value] [, 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]
  [, TAB_MODE=value] [, ZOOM=vector] )
```

**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] [, TAB_MODE=value] [, /TEXT_FRAME]
  [, 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 | , /SAVE]
  [, PATH=string] [, TAB_MODE=value]
  [, UNAME=string] [, UVALUE=value]
  [, /WARN_EXIST] )
```

**CW\_FORM** - Creates compound widget for creating forms.

```
Result = CW_FORM( [Parent,] Desc [, /COLUMN]
[, IDS=variable] [, TAB_MODE=value] [, 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 [, /DOUBLE] [, /DRAG]
[, /EDIT] [, FORMAT=string] [, /FRAME]
[, MAXIMUM=value] [, MINIMUM=value]
[, SCROLL=units] [, /SUPPRESS_VALUE]
[, TAB_MODE=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]
[, TAB_MODE=value] [, /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] [, TAB_MODE=value]
[, 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]] [, TAB_MODE=value]
[, 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 [, /COLUMN]
[, /CONTEXT_MENU] [, DELIMITER=string]
[, FONT=value] [, /MBAR [, /HELP]] [, IDS=variable]
[, /RETURN_ID | , /RETURN_INDEX | ,
/RETURN_NAME | , /RETURN_FULL_NAME]
[, TAB_MODE=value] [, 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 | , GRAPHICS_LEVEL={1 | 2}]
[, /DRAG] [, /FRAME] [, LENGTH=value] [, /RGB]
[, TAB_MODE=value] [, UNAME=string]
[, UVALUE=value] [, VALUE=[r, g, b]]
[, /VERTICAL])
```

**CW\_TMPL** - Template for compound widgets that use XMANAGER.

```
Result = CW_TMPL( Parent [, TAB_MODE=value]
[, 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] [, TAB_MODE=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( D1 [, ..., D8] [, /NOZERO])
```

**DCINDGEN** - Returns a double-precision, complex array with each element set to its subscript.

*Result* = DCINDGEN( *D<sub>1</sub>* [, ..., *D<sub>8</sub>*] [, *Thread pool keywords*])

**DCOMPLEX** - Converts argument to double-precision complex type.

*Result* = DCOMPLEX( *Real* [, *Imaginary*] [, *Thread pool keywords*])

or

*Result* = DCOMPLEX( *Expression*, *Offset*, *D<sub>1</sub>* [, ..., *D<sub>8</sub>*] [, *Thread pool keywords*])

**DCOMPLEXARR** - Creates a complex, double-precision vector or array.

*Result* = DCOMPLEXARR( *D<sub>1</sub>* [, ..., *D<sub>8</sub>*] [, /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]

**DEFINE\_MSGBLK** - Defines and loads a new message block into the current IDL session.

DEFINE\_MSGBLK, *BlockName*, *ErrorNames*, *ErrorFormats* [, /IGNORE\_DUPLICATE] [, PREFIX = *PrefixStr*]

**DEFINE\_MSGBLK\_FROM\_FILE** - Reads the definition of a message block from a file, and loads it into the current IDL session.

DEFINE\_MSGBLK\_FROM\_FILE, *Filename* [, BLOCK = *BlockName*] [, /IGNORE\_DUPLICATE] [, PREFIX = *PrefixStr*] [, /VERBOSE]

**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*]

**DELVAR** - Deletes variables from the main IDL program level.

DELVAR, *V<sub>1</sub>*, ..., *V<sub>n</sub>*

**DENDRO\_PLOT** - Draws a two-dimensional dendrite plot on the current direct graphics device if given a hierarchical tree cluster, as created by CLUSTER\_TREE.

DENDRO\_PLOT, *Clusters*, *Linkdistance* [, LABEL\_CHARSIZE=*value*] [, LABEL\_CHARTHICK=*value*]

[, LABEL\_COLOR=*value*] [, LABEL\_NAMES=*vector*] [, LABEL\_ORIENTATION=*value*] [, LINECOLOR=*value*] [, LINSTYLE=*value*] [, ORIENTATION={1|2|3|4}] [, /OVERPLOT]

**DENDROGRAM** - Constructs a dendrogram and returns a set of vertices and connectivity that can be used to visualize the dendrite plot if given a hierarchical tree cluster, as created by CLUSTER\_TREE.

DENDROGRAM, *Clusters*, *Linkdistance*, *Outverts*, *Outconn* [, LEAFNODES=*variable*]

**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<sub>x</sub>*] *Sig<sub>y</sub>* )

**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{WIN, X}] [, /CLOSE{Z}] [, /CLOSE\_DOCUMENT{PRINTER}] [, /CLOSE\_FILE{CGM, HP, METAFILE, PCL, PS, REGIS, TEK}] [, /CMYK{PS}] [, /COLOR{PCL, PS}] [, COLORS=*value*{CGM, TEK}] [, COPY=*Xsource*, *Ysource*, *cols*, *rows*, *Xdest*, *Ydest*] [, Window\_index]{WIN, X}] [, /CURSOR\_CROSSHAIR{WIN, X}] [, /CURSOR\_IMAGE=*value*{ 16-element short int vector}{WIN, X}] [, /CURSOR\_MASK=*value*{WIN, X}] [, /CURSOR\_ORIGINAL{WIN, X}] [, /CURSOR\_STANDARD=*value*{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*]{WIN, X}] [, /DECOMPOSED{WIN, X}] [, /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, METAFILE, PCL, PS, REGIS, TEK}] [, /FLOYD{PCL, X}] [, FONT\_INDEX=*integer*{PS}]

```
[, FONT_SIZE=points{PS}]
[, GET_CURRENT_FONT=variable{METAFILE,
PRINTER, WIN, X}]
[, GET_DECOMPOSED=variable{WIN, X}]
[, GET_FONTNAMES=variable{METAFILE,
PRINTER, WIN, X}]
[, GET_FONTNUM=variable{METAFILE, PRINTER,
WIN, X}]
[, GET_GRAPHICS_FUNCTION=variable{WIN, X, Z}]
[, GET_PAGE_SIZE=variable{PRINTER}]
[, GET_PIXEL_DEPTH=variable]
[, GET_SCREEN_SIZE=variable{WIN, X}]
[, GET_VISUAL_DEPTH=variable{WIN, X}]
[, GET_VISUAL_NAME=variable{WIN, X}]
```

**DEVICE** - *continued*

```
[, GET_WINDOW_POSITION=variable{WIN, X}]
[, GET_WRITE_MASK=variable{X, Z}]
[, GIN_CHARS=number_of_characters{TEK}]
[, GLYPH_CACHE=number_of_glyphs{METAFILE,
PRINTER, PS, WIN, X}] [, /INCHES{HP, METAFILE,
PCL, PRINTER, PS}] [, /INDEX_COLOR{METAFILE,
PRINTER}] [, /ITALIC{PS}] [, /LANDSCAPE | ,
/PORTRAIT{HP, PCL, PRINTER, PS}]
[, LANGUAGE_LEVEL={1 | 2}{PS}] [, /DEMI | ,
/LIGHT | , /MEDIUM | , /NARROW | , /OBLIQUE{PS}]
[, OPTIMIZE={0 | 1 | 2}{PCL}] [, /ORDERED{PCL,
X}] [, OUTPUT=scalar string{HP, PS}]
[, /PIXELS{PCL}] [, PLOT_TO=logical unit
num{REGIS, TEK}] [, /PLOTTER_ON_OFF{HP}]
[, /POLYFILL{HP}] [, PRE_DEPTH=value{PS}]
[, PRE_XSIZE=width{PS}] [, PRE_YSIZE=height{PS}]
[, /PREVIEW{PS}] [, PRINT_FILE=filename{WIN}]
[, /PSEUDO_COLOR{X}]
[, RESET_STRING=string{TEK}]
[, RESOLUTION=value{PCL}] [, RETAIN={0 | 1 |
2}{WIN, X}] [, SCALE_FACTOR=value{PRINTER,
PS}] [, SET_CHARACTER_SIZE=[font size, line
spacing]{CGM, HP, METAFILE, 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{METAFILE,
PRINTER, PS, WIN, Z}]
[, SET_GRAPHICS_FUNCTION=code{0 to 15}{WIN,
X, Z}] [, SET_PIXEL_DEPTH={8 | 24}]
[, SET_RESOLUTION=[width, height]{Z}]
[, SET_STRING=string{TEK}]
[, SET_TRANSLATION=variable{X}] [, SET_WRITE_
MASK=value{0 to 2n-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{PCL, X}]
[, TRANSLATION=variable{WIN, X}]
```

```
[, TRUE_COLOR=value{bits per pixel}{METAFILE,
PRINTER, X}] [, /TT_FONT{METAFILE, PRINTER,
WIN, X, Z}] [, /TTY{REGIS, TEK}] [, /VT240 | ,
/VT241 | , /VT340 | , /VT341 {REGIS}]
[, WINDOW_STATE=variable{WIN, X}]
[, XOFFSET=value{HP, PCL, PRINTER, PS}]
[, XON_XOFF={0 | 1 (default)}{HP}]
[, XSIZE=width{HP, PCL, METAFILE, PRINTER, PS}]
[, YOFFSET=value{HP, PCL, PRINTER, PS}]
[, YSIZE=height{HP, PCL, METAFILE, 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]
```

**DIAG\_MATRIX** - Constructs a diagonal matrix from an input vector, or if given a matrix, extracts a diagonal vector.

*Result* = DIAG\_MATRIX(A [, Diag] )

**DIALOG\_MESSAGE** - Creates modal message dialog.

```
Result = DIALOG_MESSAGE( Message_Text
[, /CANCEL] [, /CENTER] [, /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(
[, DEFAULT_EXTENSION=string] [, /DIRECTORY]
[, DIALOG_PARENT=widget_id]
[, DISPLAY_NAME=string] [, FILE=string]
[, FILTER=string/string array] [, /FIX_FILTER]
[, GET_PATH=variable] [, GROUP=widget_id]
[, /MULTIPLE_FILES] [, /MUST_EXIST]
[, /OVERWRITE_PROMPT] [, PATH=string] [, /READ |
, /WRITE] [, RESOURCE_NAME=string]
[, 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\_TYPE=*string*] [, /FIX\_FILTER]  
 [, GET\_PATH=*variable*] [, 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*] [, FILE=*string*]  
 [, /FIX\_TYPE] [, /NOWRITE] [, OPTIONS=*variable*]  
 [, PATH=*string*] [, TITLE=*string*] [, TYPE=*variable*]  
 [, /WARN\_EXIST])

**DICOMEX\_GETCONFIGFILEPATH** - See [Appendix B](#), “IDL DICOM Quick Reference” in the *Medical Imaging in IDL* manual.

**DICOMEX\_GETSTORSCPDIR** - See [Appendix B](#), “IDL DICOM Quick Reference” in the *Medical Imaging in IDL* manual.

**DICOMEX\_NET** - See [Appendix B](#), “IDL DICOM Quick Reference” in the *Medical Imaging in IDL* manual.

**DIGITAL\_FILTER** - Calculates coefficients of a non-recursive, digital filter.

*Result* = DIGITAL\_FILTER( *Flow*, *Fhigh*, *A*, *Nterms*  
 [, /DOUBLE])

**DILATE** - Implements morphologic dilation operator on binary and grayscale images.

*Result* = DILATE( *Image*, *Structure* [, *X<sub>0</sub>* [, *Y<sub>0</sub>* [, *Z<sub>0</sub>*]]  
 [, /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<sub>1</sub>* [, ..., *D<sub>8</sub>*] [, *Thread pool keywords*])

**DISSOLVE** - Provides a digital “dissolve” effect for images.

DISSOLVE, *Image* [, DELAY=*seconds*] [, /ORDER]  
 [, SIZ=*pixels*] [, X0=*pixels*, Y0=*pixels*]

**DIST** - Creates array with each element proportional to its frequency.

*Result* = DIST(*N* [, *M*])

**DISTANCE\_MEASURE** - Computes the pairwise distance between a set of items or observations.

*Result* = DISTANCE\_MEASURE( *Array* [, /DOUBLE]  
 [, /MATRIX] [, MEASURE=*value*]  
 [, POWER\_MEASURE=*value*])

**DLM\_LOAD** - Explicitly causes a DLM to be loaded.

DLM\_LOAD, *DLMNameStr<sub>1</sub>*  
 [, *DLMNameStr<sub>2</sub>*, ..., *DLMNameStr<sub>n</sub>*]

**DOC\_LIBRARY** - Extracts documentation headers from IDL programs.

DOC\_LIBRARY [, *Name*] [, /PRINT]

**UNIX keywords:** [, DIRECTORY=*string*] [, /MULTI]

**DOUBLE** - Converts argument to double-precision type.

*Result* = DOUBLE(*Expression* [, *Offset* [, *D<sub>1</sub>* [, ..., *D<sub>8</sub>*]]  
 [, *Thread pool keywords*])

**DRAW\_ROI** - Draws region or group of regions to current Direct Graphics device.

DRAW\_ROI, *oROI* [, /LINE\_FILL] [, SPACING=*value*]

**Graphics Keywords:** [, CLIP={*X<sub>0</sub>*, *Y<sub>0</sub>*, *X<sub>1</sub>*, *Y<sub>1</sub>*}]

[, COLOR=*value*] [, /DATA] [, /DEVICE] [, /NORMAL]

[, LINESSTYLE={0 | 1 | 2 | 3 | 4 | 5}] [, /NOCLIP]

[, ORIENTATION=*ccw\_degrees\_from\_horiz*]

[, PSYM=*integer*(0 to 10)] [, SYMSIZE=*value*] [, /T3D]

[, THICK=*value*]

## E

**EDGE\_DOG** - Applies the Difference of Gaussians filter to a 2D image array.

*Result* = EDGE\_DOG(*Array* [, RADIUS1=*value*]  
 [, RADIUS2=*value*] [, THRESHOLD=*value*]  
 [, ZERO\_CROSSINGS=*value*])

**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])

**EMBOSS** - Applies a convolution operator to a 2D image array to generate an array.

*Result* = EMBOSS (*Array* [, /ADD\_BACK]  
 [, AZIMUTH=*value*] [, BIAS=*value*] [, /CENTER]  
 [, /EDGE\_TRUNCATE] [, /EDGE\_WRAP]  
 [, /EDGE\_ZERO] [, INVALID=*value*]  
 [, MISSING=*value*] [, /NAN] [, /NORMALIZE])

**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 134 the quick-ref.pdf located on your IDL CD-ROM.



**.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*]

**ERF** - Returns the value of an error function.

Result = ERF( $Z$  [, *Thread pool keywords*]

**ERFC** - Returns the value of a complementary error function.

Result = ERFC( $Z$  [, *Thread pool keywords*]

**ERFCX** - Returns the value of a scaled complementary error function.

Result = ERFCX( $Z$  [, *Thread pool keywords*]

**ERRPLOT** - Plots error bars over a previously drawn plot.

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

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

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

**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* [, *Thread pool keywords*]

**EXPAND** - Shrinks/expands image using bilinear interpolation.

EXPAND,  $A$ ,  $N_x$ ,  $N_y$ , *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* [, /ALL\_DIRS]  
[, /ARRAY] [, COUNT=*variable*] [, /DLM] [, /HELP]

**EXPINT** - Returns the value of the exponential integral.

Result = EXPINT(  $N$ ,  $X$  [, /DOUBLE] [, EPS=*value*]  
[, ITER=*variable*] [, ITMAX=*value*] [, *Thread pool keywords*]

**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]*]  
[, /CUBIC] [, OUT\_VAL=*value*] [, /RADIANS]  
[, /SAMPLE] [, VERTICES=*variable*]

or

Result = EXTRACT\_SLICE( *Vol*,  $Xsize$ ,  $Ysize$ ,  $Xcenter$ ,  
 $Ycenter$ ,  $Zcenter$ , *PlaneNormal*,  $Xvec$

[, ANISOTROPY=*[xspacing, yspacing, zspacing]*]  
[, /CUBIC] [, 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] [, /UL64]

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

Result = FFT( *Array* [, *Direction*] [, DIMENSION=*vector*]  
[, /DOUBLE] [, /INVERSE] [, /OVERWRITE]  
[, *Thread pool keywords*]

**FILE\_BASENAME** - Returns the *basename* of a *file path*.

Result = FILE\_BASENAME(*Path* [, *RemoveSuffix*]  
[, /FOLD\_CASE])

**FILE\_CHMOD** - Changes the current access permissions (or modes) associated with a file or directory.

FILE\_CHMOD, *File* [, *Mode*]  
[, /A\_EXECUTE] [, /A\_READ] [, /A\_WRITE]  
[, /G\_EXECUTE] [, /G\_READ] [, /G\_WRITE]  
[, /O\_EXECUTE] [, /O\_READ] [, /O\_WRITE]  
[, /NOEXPAND\_PATH]  
[, /U\_EXECUTE] [, /U\_READ] [, /U\_WRITE]  
**UNIX-Only Keywords:** [, /SETGID] [, /SETUID]  
[, /STICKY\_BIT]

**FILE\_COPY** - Copies files or directories to a new location.

FILE\_COPY, *SourcePath*, *DestPath* [, /ALLOW\_SAME]  
[, /NOEXPAND\_PATH] [, /OVERWRITE]  
[, /RECURSIVE] [, /REQUIRE\_DIRECTORY]  
[, /VERBOSE]

**Unix-Only Keywords:** [, /COPY\_NAMED\_PIPE]  
[, /COPY\_SYMLINK] [, /FORCE]

**FILE\_DELETE** - Deletes a file or empty directory, if the process has the necessary permissions to remove the file as defined by the current operating system.

FILE\_DELETE, *File1* [, ... *FileN*]  
[, /ALLOW\_NONEXISTENT] [, /NOEXPAND\_PATH]  
[, /QUIET] [, /RECURSIVE] [, /VERBOSE]

**FILE\_DIRNAME** - Returns the *dirname* of a *file path*.

Result = FILE\_DIRNAME(*Path*  
[, /MARK\_DIRECTORY])

**FILE\_EXPAND\_PATH** - Expands a given file or partial directory name to its fully qualified name regardless of the current working directory.

Return = FILE\_EXPAND\_PATH( *Path*)

**FILE\_INFO** - Returns status information about a file.

*Result* = FILE\_INFO(*Path* [, /NOEXPAND\_PATH] )

**FILE\_LINES** - Returns the number of lines of text in a file.

FILE\_LINES(*Path* [, /COMPRESS]  
[, /NOEXPAND\_PATH] )

**FILE\_LINK** - Creates UNIX file links.

FILE\_LINK, *SourcePath*, *DestPath* [, /ALLOW\_SAME]  
[, /HARDLINK] [, /NOEXPAND\_PATH] [, /VERBOSE]

**FILE\_MKDIR** - Creates a new directory, or directories, with default access permissions for the current process.

FILE\_MKDIR, *File1* [, ... *FileN*] [, /NOEXPAND\_PATH]

**FILE\_MOVE** - Renames files and directories.

FILE\_MOVE, *SourcePath*, *DestPath* [, /ALLOW\_SAME]  
[, /NOEXPAND\_PATH] [, /OVERWRITE]  
[, /REQUIRE\_DIRECTORY] [, /VERBOSE]

**FILE\_POLL\_INPUT** - Blocks processing until it detects that a read operation on a specified file will succeed.

*Result* = FILE\_POLL\_INPUT(*Units* [, COUNT=*variable*]  
[, TIMEOUT=*value*] )

**FILE\_READLINK** - Returns the path pointed to by a UNIX symbolic link.

FILE\_READLINK(*Path* [, /ALLOW\_NONEXISTENT]  
[, /ALLOW\_NONSYMLINK] [, /NOEXPAND\_PATH] )

**FILE\_SAME** - Determines whether two different file names refer to the same underlying file.

*Result* = FILE\_SAME(*Path1*, *Path2*  
[, /NOEXPAND\_PATH] )

**FILE\_SEARCH** - Returns a string array containing the names of all files matching the input path specification.

*Result* = FILE\_SEARCH(*Path\_Specification*)

or

*Result* = FILE\_SEARCH(*Dir\_Specification*,  
*Recur\_Pattern*)

**Keywords:** [, COUNT=*variable* ]  
[, /EXPAND\_ENVIRONMENT ] [, /EXPAND\_TILDE ]  
[, /FOLD\_CASE ] [, /FULLY\_QUALIFY\_PATH ]  
[, /ISSUE\_ACCESS\_ERROR ]  
[, /MARK\_DIRECTORY ]  
[, /MATCH\_ALL\_INITIAL\_DOT | /MATCH\_INITIAL\_DOT ]  
[, /NOSORT ] [, /QUOTE ]  
[, /TEST\_DIRECTORY ] [, /TEST\_EXECUTABLE ]  
[, /TEST\_READ ] [, /TEST\_REGULAR ]  
[, /TEST\_WRITE ] [, /TEST\_ZERO\_LENGTH ]  
[, /WINDOWS\_SHORT\_NAMES ]

**UNIX-Only Keywords:** [, /TEST\_BLOCK\_SPECIAL ]  
[, /TEST\_CHARACTER\_SPECIAL ]  
[, /TEST\_DANGLING\_SYMLINK ] [, /TEST\_GROUP ]  
[, /TEST\_NAMED\_PIPE ] [, /TEST\_SETGID ]  
[, /TEST\_SETUID ] [, /TEST\_SOCKET ]  
[, /TEST\_STICKY\_BIT ] [, /TEST\_SYMLINK ]  
[, /TEST\_USER ]

**FILE\_TEST** - Checks files for existence and other file attributes without first having to open the file.

*Result* = FILE\_TEST(*File* [, /DIRECTORY ] ,  
/EXECUTABLE | /READ | /REGULAR | /WRITE |  
/ZERO\_LENGTH] [, GET\_MODE=*variable*]  
[, /NOEXPAND\_PATH] )

**UNIX-Only Keywords:** [, /BLOCK\_SPECIAL |  
/CHARACTER\_SPECIAL | /DANGLING\_SYMLINK  
| /GROUP | /NAMED\_PIPE | /SETGID | /SETUID |  
/SOCKET | /STICKY\_BIT | /SYMLINK | /USER]

**FILE\_WHICH** - Separates a specified file path into its component directories, and searches each directory in turn for a specific file.

*Result* = FILE\_WHICH (*Path*, ] *File*  
[, /INCLUDE\_CURRENT\_DIR] )

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

*Result* = FILEPATH(*Filename* [, ROOT\_DIR=*string*]  
[, SUBDIRECTORY=*string/string\_array*]  
[, /TERMINAL] [, /TMP] )

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

*Result* = FINDGEN(*D<sub>1</sub>* [, ..., *D<sub>8</sub>*] [, *Thread pool*  
*keywords*] )

**FINITE** - Returns True if its argument is finite.

*Result* = FINITE (*X* [, /INFINITY] [, /NAN]  
[, SIGN=*value*] [, *Thread pool keywords*] )

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

*Result* = FIX(*Expression* [, *Offset* [, *D<sub>1</sub>* [, ..., *D<sub>8</sub>*]]]  
[, /PRINT] [, TYPE=*type code*{0 to 15}] [, *Thread pool*  
*keywords*] )

**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* [, *D<sub>1</sub>* [, ..., *D<sub>8</sub>*]]]  
[, *Thread pool keywords*] )

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

*Result* = FLOOR(*X* [, /L64 ] [, *Thread pool keywords*] )

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

FLOW3, *V<sub>x</sub>*, *V<sub>y</sub>*, *V<sub>z</sub>* [, 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<sub>1</sub>* [, ..., *D<sub>8</sub>*] [, /NOZERO] )

**FLUSH** - Flushes file unit buffers.

FLUSH, *Unit<sub>1</sub>*, ..., *Unit<sub>n</sub>*

**FOR** - Executes 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*<sub>1</sub>, *Name*<sub>2</sub>, ..., *Name*<sub>*n*</sub>

**FREE\_LUN** - Frees previously-reserved file units.

FREE\_LUN [, *Unit*<sub>1</sub>, ..., *Unit*<sub>*n*</sub>]  
 [, EXIT\_STATUS=*variable*] [, /FORCE ]

**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*<sub>1</sub>, ..., *parameter*<sub>*n*</sub>

**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 *Z*.

*Result* = GAMMA(*Z* [, *Thread pool keywords*] )

**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*] [, CHISQ=*variable*]  
 [, ESTIMATES=*array*] [, MEASURE\_ERRORS=*vector*]  
 [, NTERMS=*integer*{3 to 6}] [, SIGMA=*variable*]  
 [, YERROR=*variable*] )

**GAUSSINT** - Returns integral of Gaussian probability function.

*Result* = GAUSSINT(*X* [, *Thread pool keywords*] )

**GET\_DRIVE\_LIST** - Returns string array of the names of valid drives/volumes for the file system.

*Result* = GET\_DRIVE\_LIST( [, COUNT=*variable*] )

**Windows keywords:** [, /CDROM] [, /FIXED]  
 [, /REMOTE] [, /REMOVABLE]

**GET\_KBRD** - Gets one input character.

*Result* = GET\_KBRD([*Wait*]  
 [, /ESCAPE | /KEY\_NAME])

**GET\_LOGIN\_INFO** - Returns the name of the computer running the IDL process, as well as the user's login name.

*Result* = GET\_LOGIN\_INFO()

**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*]

**GETENV** - Returns the value of an environment variable.

*Result* = GETENV(*Name* [, /ENVIRONMENT] )

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

GOTO, *label*

**GRID\_INPUT** - Preprocesses and sorts two-dimensional scattered data points, and removes duplicate values.

GRID\_INPUT, *X*, *Y*, *F*, *X1*, *Y1*, *F1*  
 [, DUPLICATES=*string*] [, EPSILON=*value*]  
 [, EXCLUDE=*vector*]

or

GRID\_INPUT, *Lon*, *Lat*, *F*, *Xyz*, *F1*, /SPHERE  
 [, /DEGREES] [, DUPLICATES=*string*]  
 [, EPSILON=*value*] [, EXCLUDE=*vector*]

or

GRID\_INPUT, *R*, *Theta*, *F*, *X1*, *Y1*, *F1*, /POLAR  
 [, /DEGREES] [, DUPLICATES=*string*]  
 [, EPSILON=*value*] [, EXCLUDE=*vector*]

**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]*] )

**GRIDDATA** - Interpolates scattered data values and locations sampled on a plane or a sphere to a regular grid.

*Result* = GRIDDATA( *X*, *F* )  
 or  
*Result* = GRIDDATA( *X*, *Y*, *F* )  
 or  
*Result* = GRIDDATA( *X*, *Y*, *Z*, *F*, /SPHERE )  
 or  
*Result* = GRIDDATA( *Lon*, *Lat*, *F*, /SPHERE )

**GS\_ITER** - Solves linear system using Gauss-Seidel iteration.

*Result* = GS\_ITER( *A*, *B* [, /CHECK] [, /DOUBLE]  
 [, 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<sub>1</sub>* [, *N<sub>2</sub>*] [, ALPHA=*value*{0.5 to 1.0}] [, /DOUBLE] )

**HDF\_\* Routines** - See "HDF Routines" on page 138the quick-ref.pdf located on your IDL CD-ROM.

**HDF5\_\* Routines** - See "HDF5 Routines" on page 144the quick-ref.pdf located on your IDL CD-ROM for the new H5\_\* routines.

**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 meta-data into an output structure.

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

**HEAP\_FREE** - Recursively frees all heap variables referenced by its input argument.

HEAP\_FREE, *Var* [, /OBJ] [, /PTR] [, /VERBOSE]

**HEAP\_GC** - Performs garbage collection on heap variables.

HEAP\_GC [, /OBJ] [, /PTR] [, /VERBOSE]

**HEAP\_NOSAVE** - Used to clear the *save* attribute of pointer or object heap variables.

HEAP\_NOSAVE, *HeapVar*

**HEAP\_SAVE** - Used to query whether a pointer or object heap variable is savable. It can also be used to change the heap variable save attribute.

*Result* = HEAP\_SAVE(*HeapVar* [, *Set*])

**HELP** - Provides information about the current IDL session.

HELP, *Expression<sub>1</sub>*, ..., *Expression<sub>n</sub>* [, /BREAKPOINTS]  
 [, /BRIEF] [, /DEVICE] [, /DLM] [, /FILES] [, /FULL]  
 [, /FUNCTIONS] [, /HEAP\_VARIABLES] [, /KEYS]  
 [, /LAST\_MESSAGE] [, LEVEL=*value*] [, /MEMORY]  
 [, /MESSAGES] [, NAMES=*string\_of\_variable\_names*]  
 [, /OBJECTS] [, OUTPUT=*variable*] [, /PATH\_CACHE]  
 [, /PREFERENCES] [, /PROCEDURES]  
 [, /RECALL\_COMMANDS] [, /ROUTINES]  
 [, /SHARED\_MEMORY] [, /SOURCE\_FILES]

**HELP** - *continued*

[, /STRUCTURES] [, /SYSTEM\_VARIABLES]  
 [, /TRACEBACK]

**HILBERT** - Constructs a Hilbert transform.

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

**HIST\_2D** - Returns histogram of two variables.

*Result* = HIST\_2D( *V<sub>1</sub>*, *V<sub>2</sub>* [, BIN1=*width*] [, BIN2=*height*]  
 [, MAX1=*value*] [, MAX2=*value*] [, MIN1=*value*]  
 [, MIN2=*value*] )

**HIST\_EQUAL** - Histogram-equalizes an image.

*Result* = HIST\_EQUAL( *A* [, BINSIZE=*value*]  
 [, FCN=*vector*] [, /HISTOGRAM\_ONLY]  
 [, MAXV=*value*] [, MINV=*value*] [, OMAX=*variable*]  
 [, OMIN=*variable*] [, PERCENT=*value*] [, TOP=*value*] )

**HISTOGRAM** - Computes the density function of an array.

*Result* = HISTOGRAM( *Array* [, BINSIZE=*value*]  
 [, INPUT=*variable*] [, LOCATIONS=*variable*]  
 [, MAX=*value*] [, MIN=*value*] [, /NAN]  
 [, NBINS=*value*] [, OMAX=*variable*]  
 [, OMIN=*variable*]  
 [, /L64 | REVERSE\_INDICES=*variable*] )

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

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

**HOUGH** - Returns the Hough transform of a two-dimensional image.

**Hough Transform:** *Result* = HOUGH( *Array*  
 [, /DOUBLE] [, DRHO=*scalar*] [, DX=*scalar*]  
 [, DY=*scalar*] [, /GRAY] [, NRHO=*scalar*]  
 [, NTHETA=*scalar*] [, RHO=*variable*] [, RMIN=*scalar*]  
 [, THETA=*variable*] [, XMIN=*scalar*] [, YMIN=*scalar*] )  
**Hough Backprojection:** *Result* = HOUGH( *Array*,  
 /BACKPROJECT, RHO=*variable*, THETA=*variable*

[, /DOUBLE] [, DX=*scalar*] [, DY=*scalar*]  
 [, NX=*scalar*] [, NY=*scalar*] [, XMIN=*scalar*]  
 [, YMIN=*scalar*]

**HQR** - Returns all eigenvalues of an upper Hessenberg array.  
*Result* = HQR( *A* [, /COLUMN] [, /DOUBLE] )

**HSV** - Creates color table based on Hue/Saturation Value color system.  
 HSV, *Vlo*, *Vhi*, *Satlo*, *Sathi*, *Hue*, *Loops* [, *Colr*]

## I

**IBETA** - Computes the incomplete beta function.  
*Result* = IBETA( *A*, *B*, *Z*, [, /DOUBLE] [, EPS=*value*]  
 [, ITER=*variable*] [, ITMAX=*value*] )

**ICONTOUR** - Creates an iTool and associated user interface (UI) configured to display and manipulate contour data.

ICONTOUR[, *Z* [, *X*, *Y*]]

### iTool Common Keywords:

[, ANISTROPIC\_SCALE\_2D=*value*]  
 [, ANISTROPIC\_SCALE\_3D=*value*]  
 [, BACKGROUND\_COLOR=*value*]  
 [, DIMENSIONS=[*x*, *y*]]  
 [, /DISABLE\_SPLASH\_SCREEN] [, /FIT\_TO\_VIEW]  
 [, IDENTIFIER=*variable*] [, LOCATION=[*x*, *y*]]  
 [, MACRO\_NAMES=*string* or *string* array]  
 [, NAME=*string*] [, /NO\_SAVEPROMPT]  
 [, OVERPLOT=*iToolID*] [, RENDERER={0 | 1}]  
 [, SCALE\_ISOTROPIC=*value*]  
 [, STYLE\_NAME=*string*] [, TITLE=*string*]  
 [, VIEW\_GRID=[*columns*, *rows*]] [, /VIEW\_NEXT]  
 [, VIEW\_NUMBER=*integer*] [, VIEW\_TITLE=*string*]  
 [, VIEW\_ZOOM=*value*]

**iTool Contour Keywords:** [, RGB\_INDICES=*vector* of indices] [, RGB\_TABLE=*byte* array of 256 by 3 or 3 by 256 elements] [, /{X | Y | Z}LOG] [, ZVALUE=*value*]

**Contour Object Keywords:** [, AM\_PM=*vector* of two strings] [, ANISOTROPY=[*x*, *y*, *z*]] [, C\_COLOR=*color* array] [, C\_FILL\_PATTERN=*array* of IDLgrPattern objects] [, C\_LABEL\_INTERVAL=*vector*]  
 [, C\_LABEL\_NOGAPS=*vector*]  
 [, C\_LABEL\_OBJECTS=*array* of object references]  
 [, C\_LABEL\_SHOW=*vector* of integers]  
 [, C\_LINestyle=*array* of linestyles]  
 [, C\_THICK=*float* array{each element 1.0 to 10.0}]  
 [, C\_USE\_LABEL\_COLOR=*vector* of values]  
 [, C\_USE\_LABEL\_ORIENTATION=*vector* of values]  
 [, C\_VALUE=*scalar* or *vector*] [, CLIP\_PLANES=*array*]  
 [, COLOR=*RGB* vector] [, DAYS\_OF\_WEEK=*vector* of seven strings] [, DEPTH\_OFFSET=*value*]  
 [, /DOWNHILL] [, /FILL] [, GRID\_UNITS=*value*]  
 [, /HIDE] [, INSERT\_COLORBAR=*value*]  
 [, INSERT\_LEGEND=*value*] [, LABEL\_FONT=*objref*]  
 [, LABEL\_FORMAT=*string*]

[, LABEL\_FRMTDATA=*value*]  
 [, LABEL\_UNITS=*string*] [, MAX\_VALUE=*value*]  
 [, MIN\_VALUE=*value*] [, MONTHS=*vector* of 12 values]  
 [, N\_LEVELS=*value*] [, /PLANAR]  
 [, SHADE\_RANGE=[*min*, *max*] ] [, SHADING={0 | 1}]  
 [, TICKINTERVAL=*value*] [, TICKLEN=*value*]  
 [, TRANSPARENCY=*value*]  
 [, USE\_TEXT\_ALIGNMENTS=*value*]

**Axis Object Keywords:** [, {X | Y | Z}GRIDSTYLE={0 | 1 | 2 | 3 | 4 | 5 | 6}]  
 [, {X | Y | Z}MAJOR=*integer*]  
 [, {X | Y | Z}MINOR=*integer*]

### ICONTOUR Axis Object Keywords - continued

[, {X | Y | Z}RANGE=[*min*, *max*]] [, {X | Y | Z}SUBTICKLEN=*ratio*]  
 [, {X | Y | Z}TEXT\_COLOR=*RGB* vector]  
 [, {X | Y | Z}TICKFONT\_INDEX={0 | 1 | 2 | 3 | 4}]  
 [, {X | Y | Z}TICKFONT\_SIZE=*integer*] [, {X | Y | Z}TICKFONT\_STYLE={0 | 1 | 2 | 3}] [, {X | Y | Z}TICKFORMAT=*string* or *string* array] [, {X | Y | Z}TICKINTERVAL=*value*] [, {X | Y | Z}TICKLAYOUT={0 | 1 | 2}] [, {X | Y | Z}TICKLEN=*value*] [, {X | Y | Z}TICKNAME=*string* array] [, {X | Y | Z}TICKUNITS=*string*] [, {X | Y | Z}TICKVALUES=*vector*] [, {X | Y | Z}TITLE=*string*]

**IDENTITY** - Returns an identity array.

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

**IDL Container Object** - See “IDL Container” on page 78.

**IDL\_VALIDNAME** - Determines whether a string may be used as a valid IDL variable name or structure tag name.

*Result* = IDL\_VALIDNAME(*String* [, /CONVERT\_ALL] [, CONVERT\_SPACES])

**IDLanROI Object** - See “IDLanROI” on page 79.

**IDLanROIGroup Object** - See “IDLanROIGroup” on page 80.

**IDLEXBR\_ASSISTANT** - Launches the Export Bridge Assistant.

IDLEXBR\_ASSISTANT [, COLUMNHEIGHT=*value*]  
 [, COLUMNWIDTH=*value*] [, /COM]  
 [, OBJECT\_FILE=*filename*]  
 [, PROJECT\_FILE=*pathname*]  
 [, /DRAWABLE\_EVENTHANDLERS]

**IDLffDICOM Object** - See “IDLffDICOM” on page 80.

**IDLffDicomEx Object** - See Appendix B, “IDL DICOM Quick Reference” in the *Medical Imaging in IDL* manual.

**IDLffDicomExCfg Object** - See Appendix B, “IDL DICOM Quick Reference” in the *Medical Imaging in IDL* manual.

**IDLffDicomExQuery Object** - See Appendix B, “IDL DICOM Quick Reference” in the *Medical Imaging in IDL* manual.

**IDLffDicomExStorScu Object** - See Appendix B, “IDL DICOM Quick Reference” in the *Medical Imaging in IDL* manual.

**IDLffDXF Object** - See “IDLffDXF” on page 81.

**IDLffJPEG2000 Object** - See “IDLffJPEG2000” on page 81.

**IDLffLangCat Object** - See “IDLffLangCat” on page 82.

**IDLffMJPEG2000 Object** - See “IDLffMJPEG2000” on page 82.

**IDLffMrSID Object** - See “IDLffMrSID” on page 83.

**IDLffShape Object** - See “IDLffShape” on page 83.

**IDLffXMLDOM\*** - See “IDLffXMLDOM Classes” on page 84.

**IDLffXMLSAX Object** - See “IDLffXMLSAX” on page 89.

**IDLgr\* Objects** - IDLgr\* objects and their methods are described starting with “IDLgrAxis” on page 90.

**IDLit\* Objects** - IDLit\* objects and their methods are described starting with “IDLitCommand” on page 107.

**IDLITSYS\_CREATETOOL** - Creates an instance of the specified tool registered within the iTools system.

```
Result = IDLITSYS_CREATETOOL(StrTool
[, ANISTROPIC_SCALE_2D=value]
[, ANISTROPIC_SCALE_3D=value]
[, DIMENSIONS=[width,height]]
[, /DISABLE_SPLASH_SCREEN]
[, IDENTIFIER=variable] [, INITIAL_DATA=data]
[, LOCATION=[x, y]]
[, MACRO_NAMES=string or string array]
[, /NO_SAVEPROMPT] [OVERPLOT=iToolID]
[, RENDERER={0 | 1}] [, SCALE_ISOTROPIC=value]
[, STYLE_NAME=string]
[, USER_INTERFACE=string] [, VIEW_GRID=vector]
[, /VIEW_NEXT] [, VIEW_NUMBER=number]
[, VISUALIZATION_TYPE=vistype])
```

**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, Z [, /DOUBLE] [, EPS=value]
[, ITER=variable] [, ITMAX=value]
[, METHOD=variable] )
```

**IMAGE** - Creates an iTool and associated user interface (UI) configured to display and manipulate image data.

```
IIMAGE[, Image[, X, Y]]
```

**iTool Common Keywords:**

```
[, ANISTROPIC_SCALE_2D=value]
[, ANISTROPIC_SCALE_3D=value]
[, BACKGROUND_COLOR=value]
[, DIMENSIONS=[x, y]]
[, /DISABLE_SPLASH_SCREEN] [./FIT_TO_VIEW]
[, IDENTIFIER=variable] [, LOCATION=[x, y]]
[, MACRO_NAMES=string or string array]
[, NAME=string] [./NO_SAVEPROMPT]
[, OVERPLOT=iToolID] [, RENDERER={0 | 1}]
```

```
[, SCALE_ISOTROPIC=value]
[, STYLE_NAME=string] [, TITLE=string]
[, VIEW_GRID=[columns, rows]] [, /VIEW_NEXT]
[, VIEW_NUMBER=integer] [, VIEW_TITLE=string]
[, VIEW_ZOOM=value]
```

**Image Keywords:** [ ALPHA\_CHANNEL=2-D array]

```
[, BLUE_CHANNEL=2-D array]
[, GREEN_CHANNEL=2-D array]
[, IMAGE_DIMENSIONS=[width, height]]
[, IMAGE_LOCATION=[x, y]]
[, INSERT_COLORBAR=value] [, RED_CHANNEL=2-
D array] [, RGB_TABLE=byte array of 256 by 3 or 3 by
256 elements] [, TRANSPARENCY=value]
[, ZVALUE=value]
```

**Image Object Keywords:**

```
[, BLEND_FUNCTION=vector]
[, CHANNEL=hexadecimal bitmask]
[, CLIP_PLANES=array] [, /HIDE] [, /INTERPOLATE]
[, /ORDER] [, SUB_RECT=[x, y, xdim, ydim]]
```

**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 [, Thread
pool keywords] )
```

**IMAP** - Creates an iTool and associated user interface (UI) configured to display and manipulate map data.

```
IMAP[, MAP_PROJECTION=string]
```

or

```
IMAP[, Image[, X, Y]] [, GEOTIFF=variable]
[, GRID_UNITS=value] [, MAP_PROJECTION=string]
```

or

```
IMAP[, Z[, X, Y]] [, /CONTOUR] [, GRID_UNITS=value]
[, MAP_PROJECTION=string]
```

**iTool Common Keywords:**

```
[, ANISTROPIC_SCALE_2D=value]
[, ANISTROPIC_SCALE_3D=value]
[, BACKGROUND_COLOR=value]
[, DIMENSIONS=[x, y]]
[, /DISABLE_SPLASH_SCREEN] [./FIT_TO_VIEW]
[, IDENTIFIER=variable] [, LOCATION=[x, y]]
[, MACRO_NAMES=string or string array]
```

```
[, NAME=string] [, /NO_SAVEPROMPT]
[, OVERPLOT=iToolID] [, RENDERER={0 | 1}]
[, SCALE_ISOTROPIC=value]
[, STYLE_NAME=string] [, TITLE=string]
[, VIEW_GRID={columns, rows}] [, /VIEW_NEXT]
[, VIEW_NUMBER=integer] [, VIEW_TITLE=string]
[, VIEW_ZOOM=value]
```

**Image Keywords:** This procedure accepts all IIMAGE keywords.

**Contour Keywords:** If the CONTOUR keyword is set, this procedure accepts all ICONTOUR keywords.

**Map Projection Keywords:**

```
[, CENTER_LATITUDE=value]
[, CENTER_LONGITUDE=value] [, DATUM=string]
[, FALSE_EASTING=value]
[, FALSE_NORTHING=value] [, HEIGHT=value]
[, HOM_AZIM_LONGITUDE=value]
[, HOM_AZIM_ANGLE=value]
[, HOM_LATITUDE1=value]
[, HOM_LATITUDE2=value]
[, HOM_LONGITUDE1=value]
[, HOM_LONGITUDE2=value] [, IS_ZONES=value]
[, IS_JUSTIFY=value]
[, LIMIT={latmin, lonmin, latmax, lonmax}]
```

**IMAP Map Projection Keywords - continued**

```
[, MERCATOR_SCALE=value] [, OEA_ANGLE=value]
[, OEA_SHAPEM=value] [, OEA_SHAPEN=value]
[, SEMIMAJOR_AXIS=value]
[, SEMIMINOR_AXIS=value]
[, SOM_INCLINATION=value]
[, SOM_LONGITUDE=value] [, SOM_PERIOD=value]
[, SOM_RATIO=value] [, SOM_FLAG=value]
[, SOM_LANDSAT_NUMBER=value]
[, SOM_LANDSAT_PATH=value]
[, SPHERE_RADIUS=value]
[, STANDARD_PARALLEL=value]
[, STANDARD_PAR1=value]
[, STANDARD_PAR2=value]
[, TRUE_SCALE_LATITUDE=value] [, ZONE=value]
```

**Axis Keywords:**

```
[, [XY]GRIDSTYLE={0 | 1 | 2 | 3 | 4 | 5 | 6}]
[, [XY]MAJOR=integer] [, [XY]MINOR=integer]
[, [XY]RANGE={min, max}]
[, [XY]SUBTICKLEN=ratio]
[, [XY]TEXT_COLOR=RGB vector]
[, [XY]TICKFONT_INDEX={0 | 1 | 2 | 3 | 4}]
[, [XY]TICKFONT_SIZE=integer]
[, [XY]TICKFONT_STYLE={0 | 1 | 2 | 3}]
[, [XY]TICKFORMAT=string or string array]
[, [XY]TICKINTERVAL=value]
[, [XY]TICKLAYOUT={0 | 1 | 2}]
[, [XY]TICKLEN=value]
```

```
[, [XY]TICKNAME=string array]
[, [XY]TICKUNITS=string]
[, [XY]TICKVALUES=vector] [, [XY]TITLE=string]
```

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

```
Result = INDGEN(D1 [, ..., D8] [, /BYTE | /COMPLEX |
/DCOMPLEX | /DOUBLE | /FLOAT | /L64 | /LONG
| /STRING | /UINT | /UL64 | /ULONG]
[, TYPE=value] [, Thread pool keywords])
```

**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 [, /DOUBLE]
[, /SORT])
```

**INTARR** - Creates an integer vector or array.

```
Result = INTARR(D1 [, ..., D8] [, /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]
[, Thread pool keywords])
```

**INTERVAL\_VOLUME** - Generates a tetrahedral mesh from volumetric data.

```
INTERVAL_VOLUME, Data, Value0, Value1, Outverts,
Outconn [, AUXDATA_IN=array,
AUXDATA_OUT=variable]
[, GEOM_XYZ=array, TETRAHEDRA=array]
[, PROGRESS_CALLBACK=string]
[, PROGRESS_METHOD=string]
[, PROGRESS_OBJECT=objref]
[, PROGRESS_PERCENT=percent{0 to 100}]
[, PROGRESS_USERDATA=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] [, /SUPPRESS_ERROR])
```

**IPLOT** - Creates an iTool and associated user interface (UI) configured to display and manipulate plot data.

IPLOT, [X,] Y

or

IPLOT, X, Y, Z

ot

IPLOT[, R], Theta, /POLAR

**iTool Common Keywords:**

[, ANISTROPIC\_SCALE\_2D=*value*]  
 [, ANISTROPIC\_SCALE\_3D=*value*]  
 [, BACKGROUND\_COLOR=*value*]  
 [, DIMENSIONS=[*x, y*]]  
 [, /DISABLE\_SPLASH\_SCREEN] [, /FIT\_TO\_VIEW]  
 [, IDENTIFIER=*variable*] [, LOCATION=[*x, y*]]  
 [, MACRO\_NAMES=*string* or *string array*]  
 [, NAME=*string*] [, /NO\_SAVEPROMPT]  
 [, OVERPLOT=*iToolID*] [, RENDERER={0 | 1}]  
 [, SCALE\_ISOTROPIC=*value*]  
 [, STYLE\_NAME=*string*] [, TITLE=*string*]  
 [, VIEW\_GRID=[*columns, rows*]] [, /VIEW\_NEXT]  
 [, VIEW\_NUMBER=*integer*] [, VIEW\_TITLE=*string*]  
 [, VIEW\_ZOOM=*value*]

**Plot Keywords:** [, ERRORBAR\_COLOR=*RGB vector*]

[, ERROR\_CAPSIZE=*points*{0.0 to 1.0}]  
 [, /FILL\_BACKGROUND] [, FILL\_COLOR=*RGB vector*]  
 [, FILL\_LEVEL=*value*]  
 [, INSERT\_COLORBAR=*value*] [, RGB\_TABLE=*byte array* of 256 by 3 or 3 by 256 elements]  
 [, FILL\_TRANSPARENCY=*value*] [, /SCATTER]  
 [, SYM\_COLOR=*RGB color*]  
 [, SYM\_INCREMENT=*integer*]  
 [, SYM\_INDEX=*integer*]  
 [, SYM\_OBJECT=*object reference*]  
 [, SYM\_SIZE=*point*{0.0 to 1.0}]  
 [, SYM\_THICK=*points*{1.0 to 10.0}]

**IPLOT iTool Plot Keywords - continued**

[, TRANSPARENCY=*value*]  
 [, /USE\_DEFAULT\_COLOR] [, /XY\_SHADOW]  
 [, /{X | Y | Z}\_ERRORBARS] [, /{X | Y | Z}\_LOG]  
 [, {X | Y | Z}\_ERROR=*vector* or *array*]  
 [, /XZ\_SHADOW] [, /YZ\_SHADOW]

**Plot Object Keywords:** [, CLIP\_PLANES=*array*]

[, COLOR = *RGB vector*] [, /HIDE] [, /HISTOGRAM]  
 [, INSERT\_LEGEND=*value*] [, LINSTYLE=*integer*]  
 [, MAX\_VALUE=*value*] [, MIN\_VALUE=*value*]  
 [, NSUM=*value*] [, /POLAR] [, THICK=*points*{1.0 to 10.0}] [, VERT\_COLORS=*vector*]

**Axis Object Keywords:** [, {X | Y | Z}\_GRIDSTYLE={0 | 1

| 2 | 3 | 4 | 5 | 6}] [, {X | Y | Z}\_MAJOR=*integer*]  
 [, {X | Y | Z}\_MINOR=*integer*]  
 [, {X | Y | Z}\_RANGE=[*min, max*]]  
 [, {X | Y | Z}\_SUBTICKLEN=*ratio*]  
 [, {X | Y | Z}\_TEXT\_COLOR=*RGB vector*]  
 [, {X | Y | Z}\_TICKFONT\_INDEX= {0 | 1 | 2 | 3 | 4}]

[, {X | Y | Z}\_TICKFONT\_SIZE=*integer*] [, {X | Y | Z}\_TICKFONT\_STYLE={0 | 1 | 2 | 3}] [, {X | Y | Z}\_TICKFORMAT=*string* or *string array*] [, {X | Y | Z}\_TICKINTERVAL=*value*]  
 [, {X | Y | Z}\_TICKLAYOUT= {0 | 1 | 2}]  
 [, {X | Y | Z}\_TICKLEN=*value*] [, {X | Y | Z}\_TICKNAME=*string array*] [, {X | Y | Z}\_TICKUNITS=*string*] [, {X | Y | Z}\_TICKVALUES=*vector*] [, {X | Y | Z}\_TITLE=*string*]

**IR\_FILTER** - Filters data with an infinite impulse response (IIR) or finite impulse response (FIR) filter.

*Result* = IR\_FILTER( *C* [, *D*], *Data* [, /DOUBLE] )

**ISHFT** - Performs integer bit shift.

*Result* = ISHFT(*P*<sub>1</sub>, *P*<sub>2</sub> [, *Thread pool keywords*])

**ISOCONTOUR** - Interprets the contouring algorithm found in the IDLgrContour object.

ISOCONTOUR, *Values*, *Outverts*, *Outconn*

[, AUXDATA\_IN=*array*, AUXDATA\_OUT=*variable*]  
 [, C\_LABEL\_INTERVAL=*vector* of *values*]  
 [, C\_LABEL\_SHOW=*vector* of *integers*]  
 [, C\_VALUE=*scalar* or *vector*] [, /DOUBLE] [, /FILL]  
 [, GEOMX=*vector*] [, GEOMY=*vector*]  
 [, GEOMZ=*vector*] [, LEVEL\_VALUES=*variable*]  
 [, N\_LEVELS=*levels*]  
 [, OUT\_LABEL\_OFFSETS=*variable*]  
 [, OUT\_LABEL\_POLYLINES=*variable*]  
 [, OUT\_LABEL\_STRINGS=*variable*]  
 [, OUTCONN\_INDICES=*variable*]  
 [, POLYGONS=*array* of *polygon descriptions*]

**ISOSURFACE** - Returns topologically consistent triangles by using oriented tetrahedral decomposition.

ISOSURFACE, *Data*, *Value*, *Outverts*, *Outconn*

[, AUXDATA\_IN=*array*, AUXDATA\_OUT=*variable*]

**ISOSURFACE - continued**

[, GEOM\_XYZ=*array*, TETRAHEDRA=*array*]  
 [, PROGRESS\_CALLBACK=*string*]  
 [, PROGRESS\_METHOD=*string*]  
 [, PROGRESS\_OBJECT=*objref*]  
 [, PROGRESS\_PERCENT=*percent*{0 to 100}]  
 [, PROGRESS\_USERDATA=*value*]

**ISURFACE** - Creates an iTool and associated user interface (UI) configured to display and manipulate surface data.

ISURFACE[, Z[, X, Y]]

**iTool Common Keywords:**

[, ANISTROPIC\_SCALE\_2D=*value*]  
 [, ANISTROPIC\_SCALE\_3D=*value*]  
 [, BACKGROUND\_COLOR=*value*]  
 [, DIMENSIONS=[*x, y*]]  
 [, /DISABLE\_SPLASH\_SCREEN] [, /FIT\_TO\_VIEW]  
 [, IDENTIFIER=*variable*] [, LOCATION=[*x, y*]]  
 [, MACRO\_NAMES=*string* or *string array*]



[, NAME=*string*] [, /NO\_SAVEPROMPT]  
 [, OVERPLOT=*iToolID*] [, SCALE\_ISOTROPIC=*value*]  
 [, STYLE\_NAME=*string*] [, TITLE=*string*]  
 [, VIEW\_GRID=[*columns, rows*] [, /VIEW\_NEXT]  
 [, VIEW\_NUMBER=*integer*] [, VIEW\_TITLE=*string*]  
 [, VIEW\_ZOOM=*value*]

**Surface Keywords:**

[, RGB\_TABLE=*array of 256 by 3 or 3 by 256 elements*]  
 [, TEXTURE\_ALPHA=*2-D array*]  
 [, TEXTURE\_BLUE=*2-D array*]  
 [, TEXTURE\_GREEN=*2-D array*]  
 [, TEXTURE\_IMAGE=*array*]  
 [, TEXTURE\_RED=*2-D array*] [, /{X | Y | Z}LOG]

**Surface Object Keywords:** [, BOTTOM=*index or RGB*

*vector*] [, CLIP\_PLANES=*array*] [, COLOR=*RGB vector*]  
 [, DEPTH\_OFFSET=*value*] [, /DOUBLE]  
 [, /EXTENDED\_LEGO] [, /HIDDEN\_LINES] [, /HIDE]  
 [, INSERT\_COLORBAR=*value*]  
 [, INSERT\_LEGEND=*value*] [, LINSTYLE=*value*]  
 [, MAX\_VALUE=*value*] [, MIN\_VALUE=*value*]  
 [, SHADING={0 | 1}] [, /SHOW\_SKIRT] [, SKIRT=*Z value*]  
 [, STYLE={0 | 1 | 2 | 3 | 4 | 5 | 6}]  
 [, TEXTURE\_HIGHRES={0 | 1 | 2}]  
 [, /TEXTURE\_INTERP] [, THICK=*points*{1.0 to 10.0}]  
 [, TRANSPARENCY=*value*] [, /USE\_TRIANGLES]  
 [, VERT\_COLORS=*vector or 2-D array*]  
 [, ZERO\_OPACITY\_SKIP={0 | 1}]

**Axis Object Keywords:** [, {X | Y | Z}GRIDSTYLE={0 | 1

| 2 | 3 | 4 | 5 | 6}] [, {X | Y | Z}MAJOR=*integer*]  
 [, {X | Y | Z}MINOR=*integer*]  
 [, {X | Y | Z}RANGE=[*min, max*]]  
 [, {X | Y | Z} SUBTICKLEN=*ratio*]  
 [, {X | Y | Z}TEXT\_COLOR=*RGB vector*]  
 [, {X | Y | Z}TICKFONT\_INDEX={0 | 1 | 2 | 3 | 4}]  
 [, {X | Y | Z}TICKFONT\_SIZE=*integer*] [, {X | Y | Z}  
 TICKFONT\_STYLE={0 | 1 | 2 | 3}] [, {X | Y | Z}  
 TICKFORMAT=*string or string array*] [, {X | Y | Z}  
 TICKINTERVAL=*value*] [, {X | Y | Z}TICKLAYOUT=  
 {0 | 1 | 2}] [, {X | Y | Z}TICKLEN=*value*] [, {X | Y | Z}  
 TICKNAME=*string array*] [, {X | Y | Z}TICKUNITS=  
*string*] [, {X | Y | Z}TICKVALUES=*vector*] [, {X | Y | Z}  
 TITLE=*string*]

**ITCURRENT** - Set the current tool in the iTools system.ITCURRENT, *iToolID***ITDELETE** - Deletes a tool in the iTools system.ITDELETE[, *iToolID*]**ITGETCURRENT** - Gets the identifier of the current iTool.*Result* = ITGETCURRENT([, TOOL=*variable*])**ITREGISTER** - Registers tool object classes with the iTools system.ITREGISTER, *Name, ItemName* [, /ANNOTATION]  
 [, /DEFAULT] [, /FILE\_READER] [, /FILE\_WRITER][, TYPES=*string*] [, /UI\_PANEL] [, /UI\_SERVICE]  
 [, /USER\_INTERFACE] [, /VISUALIZATION]**ITRESET** - Resets the iTools session.

ITRESET[, /NO\_PROMPT]

**ITRESOLVE** - Resolves all IDL code within the iTools directory, as well as all other IDL code required for the iTools framework.ITRESOLVE[, PATH=*string*]**IVECTOR** - creates an iTool and associated user interface (UI) configured to display and manipulate vector data.

IVECTOR [, U, V [, X, Y]]

**iTool Common Keywords:**

[, ANISOTROPIC\_SCALE\_2D=*value*]  
 [, ANISOTROPIC\_SCALE\_3D=*value*]  
 [, BACKGROUND\_COLOR=*value*]  
 [, DIMENSIONS=[*x, y*]]  
 [, /DISABLE\_SPLASH\_SCREEN] [, /FIT\_TO\_VIEW]  
 [, IDENTIFIER=*variable*] [, LOCATION=[*x, y*]]  
 [, MACRO\_NAMES=*string or string array*]  
 [, NAME=*string*] [, /NO\_SAVEPROMPT]  
 [, OVERPLOT=*iToolID*] [, RENDERER={0 | 1}]  
 [, SCALE\_ISOTROPIC=*value*]  
 [, STYLE\_NAME=*string*] [, TITLE=*string*]  
 [, VIEW\_GRID=[*columns, rows*] [, /VIEW\_NEXT]  
 [, VIEW\_NUMBER=*integer*] [, VIEW\_TITLE=*string*]  
 [, VIEW\_ZOOM=*value*]

**Vector Keywords:**

[, ARROW\_STYLE={0 | 1}]  
 [, ARROW\_THICK=*float* {1.0 to 10.0}]  
 [, AUTO\_COLOR={0 | 1 | 2}]  
 [, /AUTO\_SUBSAMPLE] [, COLOR=*RGB vector*]  
 [, DATA\_LOCATION={0 | 1 | 2}]  
 [, DIRECTION\_CONVENTION={0 | 1 | 2}]  
 [, GRID\_UNITS={0 | 1 | 2}] [, HEAD\_ANGLE=*float*]  
 [, HEAD\_INDENT=*float*] [, /HEAD\_PROPORTIONAL]  
 [, HEAD\_SIZE=*float*] [, /HIDE]  
 [, INSERT\_COLORBAR=*value*]  
 [, LENGTH\_SCALE=*float*] [, MAX\_VALUE=*float*]  
 [, MIN\_VALUE=*float*] [, RGB\_TABLE=*array*]  
 [, /STREAMLINES]  
 [, STREAMLINE\_NSTEPS=*integer*]  
 [, STREAMLINE\_STEPSIZE=*float*]  
 [, SUBSAMPLE\_METHOD={0 | 1}]  
 [, SYM\_COLOR=*value*] [, SYM\_INDEX=*value*]  
 [, SYM\_SIZE=*value*] [, THICK=*float* {1.0 to 10.0}]  
 [, TRANSPARENCY=*value*]  
 [, USE\_DEFAULT\_COLOR=*value*]  
 [, VECTOR\_COLORS=*vector or array*]  
 [, VECTOR\_STYLE={0 | 1}]  
 [, X\_STREAMPARTICLES=*integer*]  
 [, X\_SUBSAMPLE=*integer*] [, /{X | Y | Z}LOG]  
 [, Y\_STREAMPARTICLES=*integer*]  
 [, Y\_SUBSAMPLE=*integer*] [, ZVALUE=*float*]

**Axis Object Keywords:**

[, {XYZ}GRIDSTYLE={0 | 1 | 2 | 3 | 4 | 5 | 6}]  
 [, {XYZ}MAJOR=*integer*] [, {XYZ}MINOR=*integer*]  
 [, {XYZ}RANGE=[*min*, *max*]]  
 [, {XYZ}SUBTICKLEN=*ratio*]  
 [, {XYZ}TEXT\_COLOR=*RGB vector*]  
 [, {XYZ}TICKFONT\_INDEX={0 | 1 | 2 | 3 | 4}]  
 [, {XYZ}TICKFONT\_SIZE=*integer*]  
 [, {XYZ}TICKFONT\_STYLE={0 | 1 | 2 | 3}]  
 [, {XYZ}TICKFORMAT=*string* or *string array*]  
 [, {XYZ}TICKINTERVAL=*value*]  
 [, {XYZ}TICKLAYOUT={0 | 1 | 2}]  
 [, {XYZ}TICKLEN=*value*] [, {XYZ}TICKNAME=*string array*]  
 [, {XYZ}TICKUNITS=*string*]  
 [, {XYZ}TICKVALUES=*vector*] [, {XYZ}TITLE=*string*]

**IVOLUME** - Creates an iTool and associated user interface (UI) configured to display and manipulate volume data.

IVOLUME[, *Vol*<sub>0</sub>[, *Vol*<sub>1</sub>][, *Vol*<sub>2</sub>, *Vol*<sub>3</sub>]]

**iTool Common Keywords:**

[, ANISOTROPIC\_SCALE\_2D=*value*]  
 [, ANISOTROPIC\_SCALE\_3D=*value*]  
 [, BACKGROUND\_COLOR=*value*]  
 [, DIMENSIONS=[*x*, *y*]]  
 [, /DISABLE\_SPLASH\_SCREEN] [, /FIT\_TO\_VIEW]  
 [, IDENTIFIER=*variable*] [, LOCATION=[*x*, *y*]]  
 [, MACRO\_NAMES=*string* or *string array*]  
 [, NAME=*string*] [, /NO\_SAVEPROMPT]  
 [, OVERPLOT=*iToolID*] [, RENDERER={0 | 1}]  
 [, SCALE\_ISOTROPIC=*value*]  
 [, STYLE\_NAME=*string*] [, TITLE=*string*]  
 [, VIEW\_GRID=[*columns*, *rows*]] [, /VIEW\_NEXT]  
 [, VIEW\_NUMBER=*integer*] [, VIEW\_TITLE=*string*]  
 [, VIEW\_ZOOM=*value*]

**Volume Keywords:** [, /AUTO\_RENDER]

[, RENDER\_EXTENTS={0 | 1 | 2}]  
 [, RENDER\_QUALITY={1 | 2}]  
 [, SUBVOLUME=[*xmin*, *ymin*, *zmin*, *xmax*, *ymax*, *zmax*]]  
 [, VOLUME\_DIMENSIONS=[*width*, *height*, *depth*]]  
 [, VOLUME\_LOCATION=[*x*, *y*, *z*]]

**Volume Object Keywords:** [, AMBIENT=*RGB vector*]

[, BOUNDS=[*xmin*, *ymin*, *zmin*, *xmax*, *ymax*, *zmax*]]  
 [, CLIP\_PLANES=*array*]  
 [, COMPOSITE\_FUNCTION={0 | 1 | 2 | 3}]  
 [, DEPTH\_CUE=[*zbright*, *zdim*]] [, /HIDE]  
 [, HINTS={0 | 1 | 2 | 3}] [, INSERT\_COLORBAR=*value*]  
 [, /INTERPOLATE] [, /LIGHTING\_MODEL]  
 [, OPACITY\_TABLE0=*byte array of 256 elements*]  
 [, OPACITY\_TABLE1=*byte array of 256 elements*]  
 [, RENDER\_STEP=[*x*, *y*, *z*]]  
 [, RGB\_TABLE0=*byte array of 256 by 3 or 3 by 256 elements*]  
 [, RGB\_TABLE1=*byte array of 256 by 3 or 3 by 256 elements*]  
 [, /TWO\_SIDED] [, /ZBUFFER]  
 [, ZERO\_OPACITY\_SKIP={0 | 1}]

**IVOLUME - continued**

**Axis Object Keywords:** [, {X | Y | Z}GRIDSTYLE={0 | 1 | 2 | 3 | 4 | 5 | 6}] [, {X | Y | Z}MAJOR=*integer*]  
 [, {X | Y | Z}MINOR=*integer*]  
 [, {X | Y | Z}RANGE=[*min*, *max*]]  
 [, {X | Y | Z}SUBTICKLEN=*ratio*]  
 [, {X | Y | Z}TEXT\_COLOR=*RGB vector*]  
 [, {X | Y | Z}TICKFONT\_INDEX={0 | 1 | 2 | 3 | 4}]  
 [, {X | Y | Z}TICKFONT\_SIZE=*integer*] [, {X | Y | Z}TICKFONT\_STYLE={0 | 1 | 2 | 3}] [, {X | Y | Z}TICKFORMAT=*string* or *string array*] [, {X | Y | Z}TICKINTERVAL=*value*] [, {X | Y | Z}TICKLAYOUT={0 | 1 | 2}] [, {X | Y | Z}TICKLEN=*value*] [, {X | Y | Z}TICKNAME=*string array*] [, {X | Y | Z}TICKUNITS=*string*] [, {X | Y | Z}TICKVALUES=*vector*] [, {X | Y | Z}TITLE=*string*]

**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*] [, /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*<sub>1</sub> [, ..., *D*<sub>8</sub>] [, *Thread pool keywords*])

**LABEL\_DATE** - Labels axes with dates. Use with [XYZ]TICKFORMAT keyword.

*Result* = LABEL\_DATE  
 ( [DATE\_FORMAT=string/string array]  
 [, AM\_PM=2-element vector of strings]  
 [, DAYS\_OF\_WEEK=7-element vector of strings]  
 [, MONTHS=12-element vector of strings]  
 [, OFFSET=value] [, /ROUND\_UP] )  
 and then,  
 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 paired data using least absolute deviation method.

*Result* = LADFIT( *X*, *Y* [, ABSDEV=variable]  
 [, /DOUBLE] )

**LAGUERRE** - Returns value of the associated Laguerre polynomial.

*Result* = LAGUERRE( *X*, *N* [, *K*]  
 [, COEFFICIENTS=variable] [, /DOUBLE] )

**LAPLACIAN** - Applies a Laplacian operator to a 2D image array.

*Result* = LAPLACIAN(*Array* [, /ADD\_BACK]  
 [, BIAS=value] [, /CENTER] [, /EDGE\_WRAP]  
 [, /EDGE\_TRUNCATE] [, /EDGE\_ZERO]  
 [, INVALID=value] [, KERNEL\_SIZE={3 | 5}]  
 [, MISSING=value] [, /NAN] [, /NORMALIZE] )

**LA\_CHOLDC** - Computes the Cholesky factorization of an  $n$ -by- $n$  symmetric (or Hermitian) positive-definite array.

LA\_CHOLDC, *Array* [, /DOUBLE] [, STATUS=variable]  
 [, /UPPER]

**LA\_CHOLMPROVE** - Uses Cholesky factorization to improve the solution to a system of linear equations.

*Result* = LA\_CHOLMPROVE( *Array*, *Achol*, *B*, *X*  
 [, BACKWARD\_ERROR=variable] [, /DOUBLE]  
 [, FORWARD\_ERROR=variable] [, /UPPER] )

**LA\_CHOLSOL** - Used in conjunction with the LA\_CHOLDC procedure to solve a set of linear equations.

*Result* = LA\_CHOLSOL( *A*, *B* [, /DOUBLE] [, /UPPER] )

**LA\_DETERM** - Uses LU decomposition to compute the determinant of a square array.

*Result* = LA\_DETERM( *A* [, /CHECK] [, /DOUBLE]  
 [, ZERO=value] )

**LA\_EIGENPROBLEM** - Uses the QR algorithm to compute all eigenvalues and eigenvectors of an array.

*Result* = LA\_EIGENPROBLEM( *A* [, *B*]  
 [, ALPHA=variable] [, BALANCE=value]  
 [, BETA=variable] [, /DOUBLE]  
 [, EIGENVECTORS=variable]  
 [, LEFT\_EIGENVECTORS=variable]  
 [, NORM\_BALANCE=variable]  
 [, PERMUTE\_RESULT=variable]  
 [, SCALE\_RESULT=variable]  
 [, RCOND\_VALUE=variable]  
 [, RCOND\_VECTOR=variable] [, STATUS=variable] )

**LA\_EIGENQL** - Computes eigenvalues and eigenvectors of an array.

*Result* = LA\_EIGENQL( *A* [, *B*] [, /DOUBLE]  
 [, EIGENVECTORS=variable] [, FAILED=variable]  
 [, GENERALIZED=value] [, METHOD=value]  
 [, RANGE=vector] [, SEARCH\_RANGE=vector]  
 [, STATUS=variable] [, TOLERANCE=value] )

**LA\_EIGENVEC** - Uses the QR algorithm to compute all or some of the eigenvectors of an array.

*Result* = LA\_EIGENVEC( *T*, *QZ* [, BALANCE=value]  
 [, /DOUBLE] [, EIGENINDEX=variable]  
 [, LEFT\_EIGENVECTORS=variable]  
 [, PERMUTE\_RESULT={ilo, ihi}]  
 [, SCALE\_RESULT=vector]  
 [, RCOND\_VALUE=variable]  
 [, RCOND\_VECTOR=variable] [, SELECT=vector] )

**LA\_ELMHES** - Reduces a real nonsymmetric or complex non-Hermitian array to upper Hessenberg form  $H$ .

*Result* = LA\_ELMHES( *Array* [, *Q*] [, BALANCE=value]  
 [, /DOUBLE] [, NORM\_BALANCE=variable]  
 [, PERMUTE\_RESULT=variable]  
 [, SCALE\_RESULT=variable] )

**LA\_GM\_LINEAR\_MODEL** - Used to solve a general Gauss-Markov linear model problem.

*Result* = LA\_GM\_LINEAR\_MODEL( *A*, *B*, *D*, *Y*  
 [, /DOUBLE] )

**LA\_HQR** - Uses the multishift QR algorithm to compute all eigenvalues of an  $n$ -by- $n$  upper Hessenberg array.

*Result* = LA\_HQR(*H* [, *Q*] [, /DOUBLE]  
 [, PERMUTE\_RESULT={ilo, ihi}]  
 [, STATUS=variable] )

**LA\_INVERT** - Uses LU decomposition to compute the inverse of a square array.

*Result* = LA\_INVERT( *A* [, /DOUBLE]  
 [, STATUS=variable] )

**LA\_LEAST\_SQUARE\_EQUALITY** - Used to solve the linear least-squares problem.

*Result* = LA\_LEAST\_SQUARE\_EQUALITY( *A*, *B*, *C*, *D*  
 [, /DOUBLE] [, RESIDUAL=variable] )

**LA\_LEAST\_SQUARES** - Used to solve the linear least-squares problem.

*Result* = LA\_LEAST\_SQUARES( *A*, *B* [, /DOUBLE] [, METHOD=*value*] [, RANK=*variable*] [, RCONDITION=*value*] [, RESIDUAL=*variable*] [, STATUS=*variable*] )

**LA\_LINEAR\_EQUATION** - Uses LU decomposition to solve a system of linear equations.

*Result* = LA\_LINEAR\_EQUATION( *Array*, *B* [, BACKWARD\_ERROR=*variable*] [, /DOUBLE] [, FORWARD\_ERROR=*variable*] [, STATUS=*variable*] )

**LA\_LUDC** - Computes the LU decomposition of an *n*-column by *m*-row array.

LA\_LUDC, *Array*, *Index* [, /DOUBLE] [, STATUS=*variable*] )

**LA\_LUMPROVE** - Uses LU decomposition to improve the solution to a system of linear equations.

*Result* = LA\_LUMPROVE( *Array*, *Aludc*, *Index*, *B*, *X* [, BACKWARD\_ERROR=*variable*] [, /DOUBLE] [, FORWARD\_ERROR=*variable*] )

**LA\_LUSOL** - Used in conjunction with the LA\_LUDC procedure to solve a set of *n* linear equations in *n* unknowns,  $AX = B$ .

*Result* = LA\_LUSOL( *A*, *Index*, *B* [, /DOUBLE] )

**LA\_SVD** - procedure computes the singular value decomposition of an *n*-columns by *m*-row array.

LA\_SVD, *Array*, *W*, *U*, *V* [, /DOUBLE] [, /DIVIDE\_CONQUER] [, STATUS=*variable*] )

**LA\_TRIDC** - Computes the LU decomposition of a tridiagonal array as array.

LA\_TRIDC, *AL*, *A*, *AU*, *U2*, *Index* [, /DOUBLE] [, STATUS=*variable*] )

**LA\_TRIMPROVE** - Improves the solution to a system of linear equations with a tridiagonal array.

*Result* = LA\_TRIMPROVE( *AL*, *A*, *AU*, *DAL*, *DA*, *DAU*, *DU2*, *Index*, *B*, *X* [, BACKWARD\_ERROR=*variable*] [, /DOUBLE] [, FORWARD\_ERROR=*variable*] )

**LA\_TRIQL** - Uses the QL and QR variants of the implicitly-shifted QR algorithm to compute the eigenvalues and eigenvectors of a symmetric tridiagonal array.

LA\_TRIQL, *D*, *E* [, *A*] [, /DOUBLE] [, STATUS=*variable*] )

**LA\_TRIRED** - Reduces a real symmetric or complex Hermitian array to real tridiagonal form *T*.

LA\_TRIRED, *Array*, *D*, *E* [, /DOUBLE] [, /UPPER] )

**LA\_TRISOL** - Used in conjunction with the LA\_TRIDC procedure to solve a set of linear equations.

*Result* = LA\_TRISOL( *AL*, *A*, *AU*, *U2*, *Index*, *B* [, /DOUBLE] )

**LEEFILT** - Performs the Lee filter algorithm on an image array.

*Result* = LEEFILT( *A* [, *N*, *Sig*] [, /DOUBLE] [, /EXACT] )

**LEGENDRE** - Returns value of the associated Legendre polynomial.

*Result* = LEGENDRE( *X*, *L* [, *M*] [, /DOUBLE] )

**LINBCG** - Solves a set of sparse linear equations using the iterative biconjugate gradient method.

*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*<sub>1</sub> [, ..., *D*<sub>8</sub>] [, *Thread pool keywords*] )

**LINFIT** - Fits by minimizing the Chi-square error statistic.

*Result* = LINFIT( *X*, *Y* [, CHISQ=*variable*] [, COVAR=*variable*] [, /DOUBLE] [, MEASURE\_ERRORS=*vector*] [, PROB=*variable*] [, SIGMA=*variable*] [, YFIT=*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*] )

**LL\_ARC\_DISTANCE** - Returns the longitude and latitude of a point 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*] [, MEASURE\_ERRORS=*vector*] [, SIGMA=*variable*] [, TOL=*value*] )

**LMGR** - Determines the type of license used by the current IDL session.

*Result* = LMGR( [, /CLIENTSERVER | /DEMO | /EMBEDDED | /RUNTIME | /STUDENT | /TRIAL | /VM] [, EXPIRE\_DATE=*variable*] [, /FORCE\_DEMO] [, INSTALL\_NUM=*variable*] [, LMHOSTID=*variable*] [, SITE\_NOTICE=*variable*] )

**LNGAMMA** - Returns logarithm of the gamma function of *Z*.

*Result* = LNGAMMA(*Z* [, *Thread pool keywords*] )

**LNP\_TEST** - Computes the Lomb Normalized Periodogram.

*Result* = LNP\_TEST( *X*, *Y* [, /DOUBLE] [, 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*] [, RGB\_TABLE=*value*] [, /SILENT] )

**LOCALE\_GET** - Returns the current locale of the operating platform.

*Result* = LOCALE\_GET()

**LOGICAL\_AND** - Performs a logical AND operation on its arguments.

*Result* = LOGICAL\_AND(*Arg1*, *Arg2*)

**LOGICAL\_OR** - Performs a logical OR operation on its arguments.

*Result* = LOGICAL\_OR(*Arg1*, *Arg2*)

**LOGICAL\_TRUE** - Determines whether its arguments are non-zero (or non-NULL).

*Result* = LOGICAL\_TRUE(*Arg*)

**LON64ARR** - Returns a 64-bit integer vector or array.

*Result* = LON64ARR(*D<sub>1</sub>* [, ..., *D<sub>8</sub>*] [, /NOZERO])

**LONARR** - Returns a longword integer vector or array.

*Result* = LONARR(*D<sub>1</sub>* [, ..., *D<sub>8</sub>*] [, /NOZERO])

**LONG** - Converts argument to longword integer type.

*Result* = LONG(*Expression* [, *Offset* [, *D<sub>1</sub>* [, ..., *D<sub>8</sub>*]]] [, *Thread pool keywords*])

**LONG64** - Converts argument to 64-bit integer type.

*Result* = LONG64(*Expression* [, *Offset* [, *D<sub>1</sub>* [, ..., *D<sub>8</sub>*]]] [, *Thread pool keywords*])

**LSODE** - Advances a solution to a system of ordinary differential equations one time-step H.

*Result* = LSODE(*Y*, *X*, *H*, *Derivs* [, *Status*] [, *ATOL*=*value*] [, /QUIET] [, *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* [, /DOUBLE])

**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<sub>1</sub>* [, ..., *D<sub>8</sub>*]] [, /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*] [, *Thread pool keywords*])

**MAKE\_DLL** - Builds a shareable library suitable for use with IDL's dynamic linking.

MAKE\_DLL, *InputFiles* [, *OutputFile*], *ExportedRoutineNames* [, CC=*string*] [, COMPILE\_DIRECTORY=*path*] [, DLL\_PATH=*variable*] [, EXPORTED\_DATA=*string*] [, EXTRA\_CFLAGS=*string*] [, EXTRA\_LFLAGS=*string*] [, INPUT\_DIRECTORY=*path*] [, LD=*string*] [, /NOCLEANUP] [, OUTPUT\_DIRECTORY=*path*] [, /REUSE\_EXISTING] [, /SHOW\_ALL\_OUTPUT] [, /VERBOSE]

**MAKE\_RT** - Creates a stand-alone IDL runtime distribution for one or more platforms.

MAKE\_RT, *AppName*, *Outdir*, [, IDLDIR=*path*] [, LOGFILE=*path*] [, MANIFEST=*path*] [, SAVEFILE=*path*] [, /VM] [, /DATAMINER] [, /DICOMEX] [, /HIRES\_MAPS] [, /IDL\_ASSISTANT] [, /IDL\_HELP] [, /XPRINTER] [, /LIN32] [, /LIN64] [, /MACOS] [, /SUN32] [, /SUN64] [, /WIN32] [, /WIN64]

**MAP\_2POINTS** - Returns distance, azimuth, and path relating to the great circle or rhumb line connecting two points on a sphere.

*Result* = MAP\_2POINTS(*lon0*, *lat0*, *lon1*, *lat1* [, DPATH=*value* |, /METERS |, /MILES |, NPATH=*integer*{2 or greater} |, /PARAMETERS |, RADIUS=*value*] [, /RADIANS] [, /RHUMB])

**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*] [, /CONTINENTS] [, /COUNTRIES] [, /FILL\_CONTINENTS={1 | 2}] [, ORIENTATION=*value*] [, /HIRES] [, LIMIT=*vector*] [, MAP\_STRUCTURE=*structure*] [, MLINESTYLE={0 | 1 | 2 | 3 | 4 | 5}] [, MLINEHICK=*value*] [, /RIVERS] [, SPACING=*centimeters*] [, /USA]

**Graphics Keywords:** [, /T3D] [, ZVALUE=*value*{0 to 1}]

**MAP\_GRID** - Draws parallels and meridians over a map projection.

```
MAP_GRID [, /BOX_AXES | [, CLIP_TEXT=0]
[, LATALIGN=value{0.0 to 1.0}]
[, LONALIGN=value{0.0 to 1.0}]
[, LATLAB=longitude] [, LONLAB=latitude]
[, ORIENTATION=clockwise_degrees_from_horiz]
[, CHARSIZE=value] [, COLOR=index]
[, /FILL_HORIZON] [, GLINESTYLE={0 | 1 | 2 | 3 | 4 |
5}] [, GLINETHICK=value] [, /HORIZON]
[, INCREMENT=value]
[, LABEL=n{label_every_nth_gridline}]
[, LATDEL=degrees] [, LATNAMES=array,
LATS=vector] [, LONDEL=degrees]
[, LONNAMES=array, LONS=vector]
[, MAP_STRUCTURE=structure] [, /NO_GRID]
```

**Graphics Keywords:** [, /T3D] [, ZVALUE=*value*{0 to 1}]

**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]
[, MAP_STRUCTURE=structure] [, MASK=variable]
[, 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\_FORWARD** - Transforms map coordinates from longitude/latitude to Cartesian (X, Y) coordinates.

```
Result = MAP_PROJ_FORWARD(Longitude [, Latitude]
[, CONNECTIVITY=vector] [, /FILL]
[, MAP_STRUCTURE=value] [, POLYGONS=variable]
[, POLYLINES=variable] [, /RADIANS]
[, Thread pool keywords])
```

**MAP\_PROJ\_IMAGE** - Warps an image from geographic coordinates to a specified map projection.

```
Result = MAP_PROJ_IMAGE(Image [, Range]
[, /BILINEAR] [, DIMENSIONS=vector]
[, IMAGE_STRUCTURE=structure]
[, MAP_STRUCTURE=structure] [, MASK=variable]
[, MAX_VALUE=value] [, MIN_VALUE=value]
[, MISSING=value] [, UVRANGE=variable]
[, XINDEX=variable] [, YINDEX=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\_PROJ\_INIT** - Initializes a mapping projection.

```
Result = MAP_PROJ_INIT(Projection [, DATUM=value]
[, /GCTP] [, LIMIT=vector] [, /RADIANS]
[, /RELAXED] )
```

**Keywords—Projection Parameters:**

```
[, CENTER_AZIMUTH=value]
[, CENTER_LATITUDE=value]
[, CENTER_LONGITUDE=value]
[, FALSE_EASTING=value]
[, FALSE_NORTHING=value] [, HEIGHT=value]
[, HOM_AZIM_LONGITUDE=value]
[, HOM_AZIM_ANGLE=value]
[, HOM_LATITUDE1=value]
[, HOM_LATITUDE2=value]
[, HOM_LONGITUDE1=value]
[, HOM_LONGITUDE2=value] [, IS_ZONES=value]
[, IS_JUSTIFY=value] [, MERCATOR_SCALE=value]
[, OEA_ANGLE=value] [, OEA_SHAPEM=value]
[, OEA_SHAPEN=value] [, ROTATION=value]
[, SEMIMAJOR_AXIS=value]
[, SEMIMINOR_AXIS=value]
[, SOM_INCLINATION=value]
[, SOM_LONGITUDE=value] [, SOM_PERIOD=value]
[, SOM_RATIO=value] [, SOM_FLAG=value]
[, SOM_LANDSAT_NUMBER=value]
[, SOM_LANDSAT_PATH=value]
[, SPHERE_RADIUS=value]
[, STANDARD_PARALLEL=value]
[, STANDARD_PAR1=value]
[, STANDARD_PAR2=value] [, SAT_TILT=value]
[, TRUE_SCALE_LATITUDE=value] [, ZONE=value]
```

**MAP\_PROJ\_INVERSE** - Transforms map coordinates from Cartesian (X, Y) coordinates to longitude/latitude.

```
Result = MAP_PROJ_INVERSE (X [, Y]
[, MAP_STRUCTURE=value] [, /RADIANS]
[, Thread pool keywords])
```

**MAP\_SET** - Establishes map projection type and limits.

```
MAP_SET [, P0lat, P0lon, Ror]
```

**Keywords—Projection Types:** [ [, /AITOFF] [, /ALBERS  
| [, /AZIMUTHAL] [, /CONIC] [, /CYLINDRICAL] [,  
/GNOMIC] [, /GOODESHOMOLOSINE] [, /HAMMER]  
| [, /LAMBERT] [, /MERCATOR] [,  
/MILLER\_CYLINDRICAL] [, /MOLLWEIDE] [,  
/ORTHOGRAPHIC] [, /ROBINSON] [, /SATELLITE] [,  
/SINUSOIDAL] [, /STEREOGRAPHIC] [,  
/TRANSVERSE\_MERCATOR] | NAME=*string*] ]

**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}]

**MAP\_SET Keywords - Map Characteristics - continued**  
 [, LATALIGN=*value*{0.0 to 1.0}] [, LATDEL=*degrees*]  
 [, LATLAB=*longitude*] [, LONALIGN=*value*{0.0 to  
 1.0}] [, LONDEL=*degrees*] [, LONLAB=*latitude*]  
 [, MLINESTYLE={0 | 1 | 2 | 3 | 4 | 5}]  
 [, MLINETHICK=*value*] [, /NOBORDER]  
 [, /NOERASE] [, REVERSE={0 | 1 | 2 | 3}]  
 [, 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<sub>0</sub>, Y<sub>0</sub>, X<sub>1</sub>, Y<sub>1</sub>]*]  
 [, /T3D] [, ZVALUE=*value*{0 to 1}]

**MATRIX\_MULTIPLY** - Calculates the IDL matrix-multiply operator (#) of two (possibly transposed) arrays.  
*Result* = MATRIX\_MULTIPLY( A, B [, /ATRANSPOSE]  
 [, /BTRANSPOSE] [, *Thread pool keywords*] )

**MATRIX\_POWER** - Computes the product of a matrix with itself.  
*Result* = MATRIX\_POWER(Array, N [, /DOUBLE]  
 [, STATUS=*value*])

**MAX** - Returns the value of the largest element of Array.  
*Result* = MAX( Array [, *Max\_Subscript*] [, /ABSOLUTE]  
 [, DIMENSION=*value*] [, MIN=*variable*] [, /NAN]  
 [, SUBSCRIPT\_MIN=*variable*] [, *Thread pool  
 keywords*] )

**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] [, /DOUBLE]  
 [, DIMENSION=*value*] [, /EVEN] )

**MEMORY** - Returns a vector containing information on the amount of dynamic memory currently in use by the IDL session.  
*Result* = MEMORY( [, /CURRENT] [, /HIGHWATER |  
 [, /NUM\_ALLOC] [, /NUM\_FREE |  
 [, /STRUCTURE] [, /L64] )

**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** - Reduces the density of geometry while preserving as much of the original data as possible.  
*Result* = MESH\_DECIMATE (Verts, Conn, Connout  
 [, VERTICES=*variable*]  
 [, PERCENT\_VERTICES=*percent* |  
 [, PERCENT\_POLYGONS=*percent*]  
 [, PROGRESS\_CALLBACK=*string*]  
 [, PROGRESS\_METHOD=*string*]  
 [, PROGRESS\_OBJECT=*objref*]  
 [, PROGRESS\_PERCENT=*percent*{0 to 100}]  
 [, PROGRESS\_USERDATA=*value*])

**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] [, /CLOSED] [, /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] | [Arg1, ... ArgN] |
  [, /REISSUE_LAST]] [, BLOCK=BlockName]
  [, /CONTINUE] [, /INFORMATIONAL] [, /OERROR]
  [, LEVEL=CallLevel] [, NAME=ErrorName]
  [, /NONAME] [, /NOPREFIX] [, /NOPRINT] [, /RESET]
```

**MIN** - Returns the value of the smallest element of an array.

```
Result = MIN( Array [, Min_Subscript] [, /ABSOLUTE]
  [, DIMENSION=value] [, MAX=variable] [, /NAN]
  [, SUBSCRIPT_MAX] [, Thread pool keywords])
```

**MIN\_CURVE\_SURF** - Interpolates over either a plane or a sphere with a minimum curvature surface or a thin-plate-spline surface.

```
Result = MIN_CURVE_SURF(Z [, X, Y] [, /DOUBLE]
  [, /TPS] [, /REGULAR] [, /SPHERE] [, /CONST])
  [, 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 grayscale 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[, BITRATE=value]
  [, FILENAME=string] [, IFRAME_GAP=integer value]
  [, MOTION_VEC_LENGTH={1 | 2 | 3}]
  [, QUALITY=value{0 to 100}])
```

**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]
```

**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 148th the [quickref.pdf](#) located on your IDL CD-ROM.

**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] )
```

**NOISE\_HURL** - Introduces noise into an image by changing randomly selected pixels to random colors.

```
Result = NOISE_HURL (Image [, Randomization]
  [, ITERATIONS=value] [, REPLACE_MAX=value]
  [, REPLACE_MIN=value] [, SEED=value])
```



**NOISE\_PICK** - Introduces noise into an image by picking randomly selected pixels to be replaced by a neighboring pixel from a random direction.

*Result* = NOISE\_PICK (*Image*  
[, *Randomization*][, ITERATIONS=*value*]  
[, SEED=*value*])

**NOISE\_SCATTER** - Introduces noise into an image by apply normally distributed noise to the entire image.

*Result* = NOISE\_SCATTER(*Image*  
[, /CORRELATED\_NOISE] [, LEVELS=*value*]  
[, NOISE\_FACTOR=*value*] [, SEED=*value*])

**NOISE\_SLUR** - Introduces noise into an image by selecting random pixels to be replaced by a neighboring pixel from a random location in the row above.

*Result* = NOISE\_SLUR(*Image* [, *Randomization*]  
[, ITERATIONS=*value*] [, SEED=*value*])

**NORM** - Computes Euclidean norm of vector or Infinity norm of array.

*Result* = NORM( *A* [, /DOUBLE]  
[, LNORM={0 | 1 | 2 | *n*})

## 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*<sub>1</sub>, ..., *Arg*<sub>*n*</sub>]

**OBJ\_HASMETHOD** - Determines whether an object class implements a specified method or methods.

*Result* = OBJ\_HASMETHOD(*ObjRef*, *Method*)

**OBJ\_ISA** - Determines inheritance relationship of an object.

*Result* = OBJ\_ISA(*ObjectInstance*, *ClassName*)

**OBJ\_NEW** - Creates an object reference.

*Result* = OBJ\_NEW(*ObjectClassName* [, *Arg*<sub>1</sub>, ..., *Arg*<sub>*n*</sub>])

**OBJ\_VALID** - Verifies validity of object references.

*Result* = OBJ\_VALID( [*Arg*] [, /CAST]  
[, COUNT=*variable*] )

**OBJARR** - Creates an array of object references.

*Result* = OBJARR(*D*<sub>1</sub> [, ..., *D*<sub>*g*</sub>] [, /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 online help viewer from programs.

ONLINE\_HELP [, *Value*] [, BOOK='filename']  
[, /FULL\_PATH] [, /QUIT]

**Windows-Only Keywords:** [, /CONTEXT]

**OPEN** - Opens files for reading, updating, or writing.

OPENR, *Unit*, *File*

OPENW, *Unit*, *File*

OPENU, *Unit*, *File*

**Keywords (all platforms):** [, /APPEND | , /COMPRESS]

[, BUFSIZE={0 | 1 | *value*>512}] [, /DELETE]

[, ERROR=*variable*] [, /F77\_UNFORMATTED]

[, /GET\_LUN] [, /MORE] [, /NOEXPAND\_PATH]

[, /STDIO] [, /SWAP\_ENDIAN]

[, SWAP\_IF\_BIG\_ENDIAN]

[, /SWAP\_IF\_LITTLE\_ENDIAN] [, /VAX\_FLOAT]

[, WIDTH=*value*] [, /XDR]

**UNIX-Only Keywords:** [, /RAWIO]

**O PLOT** - Plots vector data over a previously-drawn plot.

O PLOT, [*X*,] *Y* [, MAX\_VALUE=*value*]

[, MIN\_VALUE=*value*] [, NSUM=*value*] [, /POLAR]

[, THICK=*value*]

**Graphics Keywords:** [, CLIP={*X*<sub>0</sub>, *Y*<sub>0</sub>, *X*<sub>1</sub>, *Y*<sub>1</sub>}]

[, COLOR=*value*] [, LINSTYLE={0 | 1 | 2 | 3 | 4 | 5}]

[, /NOCLIP] [, PSYM=*integer*{0 to 10}]

[, SYMSIZE=*value*] [, /T3D] [, ZVALUE=*value*{0 to 1}]

**O PLOTERR** - Draws error bars over a previously drawn plot.

O PLOTERR, [*X*,] *Y*, *Err* [, *Psym*]

## P

**P\_CORRELATE** - Computes partial correlation coefficient.

*Result* = P\_CORRELATE(*X*, *Y*, *C* [, /DOUBLE])

**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]

**PATH\_CACHE** - Used to control IDL's path cache mechanism.

PATH\_CACHE [, /CLEAR] [, /ENABLE] [, /REBUILD]

**PATH\_SEP** - Returns the proper file path segment separator character for the current operating system.

*Result* = PATH\_SEP( [ /PARENT\_DIRECTORY ]  
[, /SEARCH\_PATH ] )

**PCOMP** - Computes principal components/derived variables.

*Result* = PCOMP( *A* [, COEFFICIENTS=*variable*]

[, /COVARIANCE] [, /DOUBLE]

[, EIGENVALUES=*variable*] [, N VARIABLES=*value*]

[, /STANDARDIZE] [, VARIANCES=*variable*] )

**PLOT** - Plots vector arguments as X versus Y graphs.

PLOT, [*X*,] *Y* [, /ISOTROPIC] [, 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<sub>0</sub>, Y<sub>0</sub>, X<sub>1</sub>, Y<sub>1</sub>*] [, COLOR=*value*] [, /DATA |  
 /DEVICE | /NORMAL] [, FONT=*integer*]  
 [, LINSTYLE={0 | 1 | 2 | 3 | 4 | 5}] [, /NOCLIP]  
 [, /NODATA] [, /NOERASE] [, POSITION=*X<sub>0</sub>, Y<sub>0</sub>, X<sub>1</sub>,  
 Y<sub>1</sub>*] [, 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}TICK\_GET=*variable*]  
 [, {X | Y | Z}TICKFORMAT=*string*]  
 [, {X | Y | Z}TICKINTERVAL=*value*]  
 [, {X | Y | Z}TICKLAYOUT=*scalar*]  
 [, {X | Y | Z}TICKLEN=*value*]  
 [, {X | Y | Z}TICKNAME=*string\_array*]  
 [, {X | Y | Z}TICKS=*integer*]  
 [, {X | Y | Z}TICKUNITS=*string*]  
 [, {X | Y | Z}TICKV=*array*] [, {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 [, AX=*degrees*] [, AZ=*degrees*]  
 [, 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}] [, 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<sub>0</sub>, Y<sub>0</sub>, X<sub>1</sub>, Y<sub>1</sub>*]  
 [, COLOR=*value*] [, /DATA | /DEVICE | /NORMAL]  
 [, LINSTYLE={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 [, P1] [, /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 coord-  
 inates.  
 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<sub>0</sub>, y<sub>0</sub>, x<sub>1</sub>, y<sub>1</sub>]*] [, /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<sub>x</sub>, Dim<sub>y</sub>]]  
 [, CUBIC={-1 to 0}] [, MISSING=*value*] [, Thread pool  
 keywords] )

**POLY\_AREA** - Returns the area of a polygon given the coordinates of  
 its vertices.  
 Result = POLY\_AREA( X, Y [, /DOUBLE] [, /SIGNED] )

**POLY\_FIT** - Performs a least-square polynomial fit.  
 Result = POLY\_FIT( X, Y, Degree [, CHISQ=*variable*]  
 [, COVAR=*variable*] [, /DOUBLE]  
 [, MEASURE\_ERRORS=*vector*] [, SIGMA=*variable*]  
 [, STATUS=*variable*] [, YBAND=*variable*]  
 [, YERROR=*variable*] [, YFIT=*variable*] )

**POLYFILL** - Fills the interior of a polygon.  
 POLYFILL, X [, Y [, Z]] [, IMAGE\_COORD=*array*]  
 [, /IMAGE\_INTERP] [, /LINE\_FILL]  
 [, PATTERN=*array*] [, SPACING=*centimeters*]  
 [, TRANSPARENT=*value*]

**Graphics Keywords:** [, CLIP=*X<sub>0</sub>, Y<sub>0</sub>, X<sub>1</sub>, Y<sub>1</sub>*]  
 [, COLOR=*value*] [, /DATA | /DEVICE | /NORMAL]  
 [, LINSTYLE={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<sub>x</sub>, S<sub>y</sub> [, Run\_Length] )

- 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* [ , /DOUBLE ]  
 [ , STATUS=*variable* ]
- POPD** - Removes the top directory on the working directory stack maintained by PUSH/POPD.  
 POPD
- POWELL** - Minimizes a function using the Powell method.  
 POWELL, *P*, *Xi*, *Ftol*, *Fmin*, *Func* [ , /DOUBLE ]  
 [ , ITER=*variable* ] [ , ITMAX=*value* ]
- PREF\_COMMIT** - Commits IDL preferences already in the pending state.  
 PREF\_COMMIT [ , *PreferenceName* ] [ , /RESET ]  
 [ , /RESIGNAL ]
- PREF\_GET** - Returns information about IDL preferences.  
*Result* = PREF\_GET([*PrefName*] [ , /NAMES\_ALL |  
 /NUM\_PENDING | /STRUCTURE ])
- PREF\_MIGRATE** - Imports IDL user preferences from other versions of IDL for use by the currently running version.  
 PREF\_MIGRATE [ , /MACRO ] [ , /PREFERENCE ]  
 [ , /STARTUP ]
- PREF\_SET** - Sets new values for IDL preferences.  
 PREF\_SET, *PrefName<sub>1</sub>*, *Value<sub>1</sub>* [ , ... *PrefName<sub>n</sub>*, *Value<sub>n</sub>* ]  
 [ , /COMMIT ]  
 or  
 PREF\_SET, *PrefName<sub>1</sub>* [ , ... *PrefName<sub>n</sub>* ] /DEFAULT  
 [ , /COMMIT ]  
 or  
 PREF\_SET, FILENAME=*file* [ , /COMMIT ]
- PREWITT** - Returns an approximation to the Prewitt edge enhancement operator for images.  
*Result* = PREWITT(*Image*)
- PRIMES** - Computes the first *K* prime numbers.  
*Result* = PRIMES(*K*)
- PRINT/PRINTF** - Writes formatted output to screen or file.  
 PRINT [ , *Expr<sub>1</sub>*, ..., *Expr<sub>n</sub>* ]  
 PRINTF [ , *Unit*, *Expr<sub>1</sub>*, ..., *Expr<sub>n</sub>* ]  
**Keywords:** [ , AM\_PM=*[string, string]* ]  
 [ , DAYS\_OF\_WEEK=*string\_array*{7 names} ]  
 [ , FORMAT=*value* ]  
 [ , MONTHS=*string\_array*{12 names} ]  
 [ , /STDIO\_NON\_FINITE ]
- PRINTD** - Prints contents of the directory stack maintained by PUSH/POPD.  
 PRINTD
- PRO** - Defines a procedure.  
 PRO *Procedure\_Name*, *argument<sub>1</sub>*, ..., *argument<sub>n</sub>*  
 ...  
 END
- PRODUCT** - Returns the product of elements within an array.  
*Result* = PRODUCT(*Array* [ , *Dimension* ]  
 [ , /CUMULATIVE ] [ , /INTEGER ] [ , /NAN ]  
 [ , /PRESERVE\_TYPE ] )
- 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* [ , /AVG\_INTENSITY ] [ , /CUBIC ]  
 [ , DEPTH\_Q=*value* ] [ , OPAQUE=3*D\_array* ]  
 [ , TRANS=*array* ] [ , XSIZE=*longword integer* ]  
 [ , YSIZE=*longword integer* ] [ , /Z\_BUFFER ] )
- 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<sub>1</sub>*, ..., ..., *P<sub>n</sub>*
- 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<sub>1</sub>* [ , ...*D<sub>8</sub>* ] [ , /ALLOCATE\_HEAP | ,  
 /NOZERO ] )

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

## Q

---

**QGRID3** - Interpolates the dependent variable values to points in a regularly sampled volume.

*Result* = QGRID3( *XYZ*, *F*, *Tetrahedra* )

or

*Result* = QGRID3( *X*, *Y*, *Z*, *F*, *Tetrahedra* )

**Keywords:** [ , DELTA=*array* ] [ , DIMENSION=*array* ]  
 [ , MISSING=*value* ] [ , START=*array* ]

**QHULL** - Constructs convex hulls, Delaunay triangulations, and Voronoi diagrams.

QHULL, *V*, *Tr* or QHULL, *V0*, *V1*, [ , *V2* ... [ , *V6* ] ] , *Tr*  
 [ , BOUNDS=*variable* ] [ , CONNECTIVITY=*variable* ]  
 [ , /DELAUNAY ] [ , SPHERE=*variable* ]  
 [ , VDIAGRAM=*array* ] [ , VNORMALS=*array* ]  
 [ , VVERTICES=*array* ]

**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\_ASCII** - Tests a file for compatibility with READ\_ASCII.

*Result* = QUERY\_ASCII ( *Filename* [ , *Info* ] )

**QUERY\_BMP** - Obtains information about a BMP image file.

*Result* = QUERY\_BMP ( *Filename* [ , *Info* ] )

**QUERY\_DICOM** - Obtains information about a DICOM archive file.

*Result* = QUERY\_DICOM( *Filename*[ , *Info* ]  
 [ , IMAGE\_INDEX=*index* ] [ , /DICOMEX ] )

**QUERY\_GIF** - Obtains information about a GIF image file.

*Result* = QUERY\_GIF ( *Filename* [ , *Info* ] )

**QUERY\_IMAGE** - Determines if a file 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\_JPEG2000** - Obtains information about a JPEG2000 image file.

*Result* = QUERY\_JPEG2000( *Filename* [ , *Info* ] )

**QUERY\_MRSID** - Obtains information about a MrSID image file.

*Result* = QUERY\_MRSID( *Filename* [ , *Info* ]  
 [ , LEVEL=*lv* ] )

**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* ]  
 [ , GEOTIFF=*variable* ] [ , IMAGE\_INDEX=*index* ] )

**QUERY\_WAV** - Obtains information about a WAV sound 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* ] )

**RADON** - Returns the Radon transform of a two-dimensional image.

**Radon Transform:** *Result* = RADON( *Array*  
 [ , /DOUBLE ] [ , DRHO=*scalar* ] [ , DX=*scalar* ]  
 [ , DY=*scalar* ] [ , /GRAY ] [ , /LINEAR ] [ , NRHO=*scalar* ]  
 [ , NTHETA=*scalar* ] [ , RHO=*variable* ] [ , RMIN=*scalar* ]  
 [ , THETA=*variable* ] [ , XMIN=*scalar* ] [ , YMIN=*scalar* ] )  
**Radon Backprojection:** *Result* = RADON( *Array*,  
 /BACKPROJECT, RHO=*variable*, THETA=*variable*  
 [ , /DOUBLE ] [ , DX=*scalar* ] [ , DY=*scalar* ] [ , /LINEAR ]  
 [ , NX=*scalar* ] [ , NY=*scalar* ] [ , XMIN=*scalar* ]  
 [ , YMIN=*scalar* ] )

**RANDOMN** - Returns normally-distributed pseudo-random numbers.

*Result* = RANDOMN( *Seed* [ , *D1* [ , ..., *D8* ] ]  
 [ [ , BINOMIAL=*trials, probability*] ] [ , /DOUBLE ]  
 [ , GAMMA=*integer* >0 ] ] [ , /NORMAL ]  
 [ , POISSON=*value* ] [ , /UNIFORM ] [ [ , /LONG ] ] )

**RANDOMU** - Returns uniformly-distributed pseudo-random numbers.

*Result* = RANDOMU( *Seed* [ , *D1* [ , ..., *D8* ] ]  
 [ [ , BINOMIAL=*trials, probability*] ] [ , /DOUBLE ]

[, GAMMA=*integer*{>0}] [, /NORMAL]  
[, POISSON=*value*] [, /UNIFORM] [, /LONG] ] )

**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*<sub>1</sub>, ..., *Var*<sub>*n*</sub>

READF, [*Prompt*,] *Unit*, *Var*<sub>1</sub>, ..., *Var*<sub>*n*</sub>

**Keywords:** [, AM\_PM=*string*, *string*]

[, DAYS\_OF\_WEEK=*string\_array*{7 names}]

[, FORMAT=*value*] [, MONTHS=*string\_array*{12

names}] [, PROMPT=*string*]

**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*] [, /DICOMEX] )

**READ\_GIF** - Reads a GIF image.

READ\_GIF, *Filename*, *Image* [, *R*, *G*, *B*] [,

BACKGROUND\_COLOR=*variable*] [, /CLOSE]

[, DELAY\_TIME=*variable*]

[, DISPOSAL\_METHOD=*variable*] [, /MULTIPLE]

[, REPEAT\_COUNT=*variable*]

[, TRANSPARENT=*value*] [, USER\_INPUT=*variable*]

**READ\_IMAGE** - Reads the image contents of a file and returns the image in an IDL variable.

*Result* = READ\_IMAGE (*Filename* [, *Red*, *Green*, *Blue*]

[, IMAGE\_INDEX=*index*])

**READ\_INTERFILE** - Reads Interfile (v3.3) file.

READ\_INTERFILE, *File*, *Data*

**READ\_JPEG** - Reads JPEG file.

READ\_JPEG [, *Filename* | , UNIT=*lum*] , *Image*

[, *Colortable*] [, BUFFER=*variable*]

[, COLORS=*value*{8 to 256}] [, DITHER={0 | 1 | 2}]

[, /GRAYSCALE] [, /ORDER] [, TRUE={1 | 2 | 3}]

[, /TWO\_PASS\_QUANTIZE]

**READ\_JPEG2000** - Reads a JPEG2000 file.

*Result* = READ\_JPEG2000(*Filename* [, *Red*, *Green*, *Blue*]

[, DISCARD\_LEVELS=*value*] [,

MAX\_LAYERS=*value*] [, /ORDER]

[, REGION=[*StartX*, *StartY*, *Width*, *Height*] ] )

**READ\_MRSID** - Reads MrSID file.

*Result* = READ\_MRSID ( *Filename* [, LEVEL=*lv*]

[, SUB\_RECT=*rect*] )

**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*] [, /ORDER]

[, /VERBOSE] [, TRANSPARENT=*variable*] )

or

READ\_PNG, *Filename*, *Image* [, *R*, *G*, *B*] [, /ORDER]

[, /VERBOSE] [, TRANSPARENT=*variable*]

**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*]

[, CHANNELS=*scalar* or *vector*]

[, DOT\_RANGE=*variable*] [, GEOTIFF=*variable*]

[, IMAGE\_INDEX=*value*] [, ICC\_PROFILE=*variable*]

[, INTERLEAVE={0 | 1 | 2}]

[, ORIENTATION=*variable*] [, PHOTOSHOP=*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<sub>1</sub>*, ..., *Var<sub>n</sub>* [, 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<sub>1</sub>*, ..., *Var<sub>n</sub>* [, TRANSFER\_COUNT=*variable*]

**REAL\_PART** - Returns the real part of a complex-valued argument.

*Result* = REAL\_PART(*Z*)

**REBIN** - Resizes a vector or array by integer multiples.

*Result* = REBIN( *Array*, *D<sub>1</sub>* [, ..., *D<sub>g</sub>*] [, /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* [, /CUBIC] [, MISSING=*value*] [, MODE=*value*] [, /QUIET] )

**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<sub>1</sub>* [, ..., *D<sub>g</sub>*] [, /OVERWRITE] )

**REGION\_GROW** - Perform region growing.

*Result* = REGION\_GROW( *Array*, *ROI* *Pixels* [, /ALL\_NEIGHBORS] [, /NAN] [, STDDEV\_MULTIPLIER=*value*] THRESHOLD=[*min*, *max*] )

**REGISTER\_CURSOR** - Associates the given name with the given cursor information.

REGISTER\_CURSOR, *Name*, *Image* [, MASK=*value*] [, HOTSPOT=*value*] [, /OVERWRITE]

**REGRESS** - Computes fit using multiple linear regression.

*Result* = REGRESS( *X*, *Y* [, CHISQ=*variable*] [, CONST=*variable*] [, CORRELATION=*variable*] [, /DOUBLE] [, FTEST=*variable*] [, MCORRELATION=*variable*] [, MEASURE\_ERRORS=*vector*] [, SIGMA=*variable*] [, STATUS=*variable*] [, YFIT=*variable*] )

**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<sub>1</sub>* [, ..., *D<sub>g</sub>*] [, *Thread pool keywords*] )

**REPLICATE\_INPLACE** - Updates an array by replacing all or selected parts of it with a specified value.

REPLICATE\_INPLACE, *X*, *Value* [, *D<sub>1</sub>*, *Loc1*] [, *D<sub>2</sub>*, *Range*] [, *Thread pool keywords*]

**RESOLVE\_ALL** - Compiles any uncompiled routines.

RESOLVE\_ALL [, CLASS=*string*] [, /CONTINUE\_ON\_ERROR] [, /QUIET]

**RESOLVE\_ROUTINE** - Compiles a routine.

RESOLVE\_ROUTINE, *Name* [, /COMPILE\_FULL\_FILE] [, /EITHER | /IS\_FUNCTION] [, /NO\_RECOMPLIE]

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

RESTORE [, *Filename*] [, DESCRIPTION=*variable*] [, FILENAME=*name*] [, /RELAXED\_STRUCTURE\_ASSIGNMENT] [, RESTORED\_OBJECTS=*variable*] [, /VERBOSE]

**RETAIL** - Returns control to the main program level.

RETAIL

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

RETURN [, *Return\_value*]

**REVERSE** - Reverses the order of one dimension of an array.

*Result* = REVERSE( *Array* [, *Subscript\_Index*] [, /OVERWRITE] )

**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<sub>0</sub>*, *Y<sub>0</sub>*] [, /INTERP] [, CUBIC=*value*{-1 to 0}] [, MISSING=*value*] [, /PIVOT])

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

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

**ROUND** - Rounds the argument to its closest integer.

*Result* = ROUND( *X* [, /L64] [, *Thread pool keywords*] )

**ROUTINE\_FILEPATH** - Returns the full path to a currently-compiled procedure or function (or a list of procedures and functions).

*Result* = ROUTINE\_FILEPATH( *Routine* [, /EITHER] [, /IS\_FUNCTION] )

**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* ] )

**SAVGOL** - Returns coefficients of Savitzky-Golay smoothing filter.

*Result* = SAVGOL( *Nleft*, *Nright*, *Order*, *Degree*  
[ , /DOUBLE ] )

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

SAVE [ , *Var<sub>1</sub>*, ..., *Var<sub>n</sub>* ] [ , /ALL ] [ , /COMM,  
/VARIABLES ] [ , /COMPRESS ]  
[ , DESCRIPTION=*string* ] [ , /EMBEDDED ]  
[ , FILENAME=*string* ] [ , /ROUTINES ]  
[ , /SYSTEM\_VARIABLES ] [ , /VERBOSE ]

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

SCALE3 [ , X RANGE=*vector* ] [ , Y RANGE=*vector* ]  
[ , Z RANGE=*vector* ] [ , AX=*degrees* ] [ , AZ=*degrees* ]

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

SCALE3D

**SCOPE\_LEVEL** - Returns the scope level of the currently running procedure or function.

*Result* = SCOPE\_LEVEL()

**SCOPE\_TRACEBACK** - Returns the current interpreter call stack (the sequence of routine calls to the present point).

*Result* = SCOPE\_TRACEBACK( [ /STRUCTURE ]  
[ , /SYSTEM ] )

**SCOPE\_VARFETCH** - Returns variables outside the local scope of the currently running procedure or function.

*Variable* = SCOPE\_VARFETCH(*VarName*  
[ , COMMON=*string* ] [ , /ENTER ] [ , LEVEL=*value* ]  
[ , /REF\_EXTRA ] )

**SCOPE\_VARNAME** - Returns the names of variables outside the local scope of the currently running procedure or function.

*Result* = SCOPE\_VARNAME([*Var<sub>1</sub>*, ..., *Var<sub>n</sub>*] [ , COM-  
MON=*string* ] [ , COUNT=*variable* ] [ , LEVEL=*value* ] )

**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 ] )

**SEM\_CREATE** - Creates a reference to a semaphore in the current IDL process. Creates the semaphore itself if necessary.

*Result* = SEM\_CREATE(*strName*  
[ , /DESTROY\_SEMAPHORE ] )

**SEM\_DELETE** - Destroys a reference to the specified semaphore in the current IDL process. Optionally destroys the semaphore itself.

SEM\_DELETE, *strName*

**SEM\_LOCK** - Attempts to gain the lock on an existing semaphore (created by a call to the SEM\_CREATE function) for the current IDL process.

*Result* = SEM\_LOCK(*strName*)

**SEM\_RELEASE** - Releases the lock on the specified semaphore held by the current IDL process.

SEM\_RELEASE, *strName*

**SET\_PLOT** - Sets the output device used by the IDL direct 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*} ]

**SETENV** - Adds or changes an environment variable.

SETENV, *Environment\_Expression*

**SFIT** - Performs polynomial fit to a surface.

*Result* = SFIT( *Data*, *Degree*  
[ , /IRREGULAR, KX=*variable*, /MAX\_DEGREE ] )

**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* ] [ , COLOR=*value* ] [ , /DATA | ,  
/DEVICE | , /NORMAL ] [ , FONT=*integer* ] [ , /NODATA ]  
[ , POSITION={*X<sub>0</sub>*, *Y<sub>0</sub>*, *X<sub>1</sub>*, *Y<sub>1</sub>*} ] [ , 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*}TICKINTERVAL=*value* ]

[ , {*X* | *Y* | *Z*}TICKLAYOUT=*scalar* ]

[ , {*X* | *Y* | *Z*}TICKLEN=*value* ]

- [, {X | Y | Z}TICKNAME=*string\_array*]  
 [, {X | Y | Z}TICKS=*integer*]  
 [, {X | Y | Z}TICKUNITS=*string*]  
 [, {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<sub>1</sub> [, ..., S<sub>n</sub>])
- SHIFT\_DIFF** - The SHIFT\_DIFF function applies a shift-difference filter to an image array.  
 Result = SHIFT\_DIFF (Array [, BIAS=*value*]  
 [, /CENTER] [, DIRECTION={0 | 1 | 2 | 3 | 4 | 5 | 6 | 7}]  
 [, /EDGE\_TRUNCATE] [, /EDGE\_WRAP]  
 [, /EDGE\_ZERO] [, INVALID=*value*]  
 [, MISSING=*value*] [, /NAN] [, /NORMALIZE] )
- SHMDEBUG** - Print debugging information when a variable loses reference to an underlying shared memory segment.  
 Result = SHMDEBUG(Enable)
- SHMMAP** - Maps anonymous shared memory, or local disk files, into the memory address space of the currently executing IDL process.  
 SHMMAP [, SegmentName] [, D<sub>1</sub>, ..., D<sub>8</sub>] [, /BYTE]  
 [, /COMPLEX] [, /DCOMPLEX]  
 [, /DESTROY\_SEGMENT] [, DIMENSION=*value*]  
 [, /DOUBLE] [, FILENAME=*value*] [, /FLOAT]  
 [, GET\_NAME=*value*] [, GET\_OS\_HANDLE=*value*]  
 [, /INTEGER] [, /L64] [, /LONG] [, OFFSET=*value*]  
 [, OS\_HANDLE=*value*] [, /PRIVATE] [, SIZE=*value*]  
 [, /SYSV] [, TEMPLATE=*value*] [, TYPE=*value*]  
 [, /UINT] [, /UL64] [, /ULONG]
- SHMUNMAP** - Removes a memory segment previously created by SHMMAP from the system.  
 SHMUNMAP, SegmentName
- SHMVAR** - Creates an IDL array variable that uses the memory from a current mapped memory segment created by the SHMMAP procedure.  
 Result = SHMVAR(SegmentName [, D<sub>1</sub>, ..., D<sub>8</sub>] [, /BYTE]  
 [, /COMPLEX] [, /DCOMPLEX] [, DIMENSION=*value*]  
 [, /DOUBLE] [, /FLOAT] [, /INTEGER] [, /L64
- [, /LONG] [, SIZE=*value*] [, TEMPLATE=*value*]  
 [, TYPE=*value*] [, /UINT] [, /UL64] [, /ULONG] )
- 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]
- SIMPLEX** - Use the simplex method to solve linear programming problems.  
 Result = SIMPLEX( Zequation, Constraints, M<sub>1</sub>, M<sub>2</sub>, M<sub>3</sub>  
 [, Tableau [, Izrow [, Iposv] ] ] [, /DOUBLE]  
 [, EPS = *value*] [, STATUS = *variable* ] )
- SIN** - Returns the trigonometric sine of X.  
 Result = SIN(X [, Thread pool keywords] )
- SINDGEN** - Returns a string array with each element set to its subscript.  
 Result = SINDGEN(D<sub>1</sub> [, ..., D<sub>8</sub>])
- SINH** - Returns the hyperbolic sine of X.  
 Result = SINH(X [, Thread pool keywords] )
- SIZE** - Returns array size and type information.  
 Result = SIZE( Expression [, /L64] [, /DIMENSIONS |  
 /FILE\_LUN | /FILE\_OFFSET] [, /N\_DIMENSIONS |  
 /N\_ELEMENTS | /SNAME | /STRUCTURE |  
 /TNAME | /TYPE] )
- SKEWNESS** - Computes statistical skewness of an n-element vector.  
 Result = SKEWNESS( X [, /DOUBLE] [, /NAN] )
- SKIP\_LUN** - Reads data in an open file and moves the file pointer.  
 SKIP\_LUN, FromUnit, [, Num] [, /EOF] [, /LINES]  
 [, /TRANSFER\_COUNT=*variable*]
- 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]  
 [, MISSING=*value*] [, /NAN] )



**SOBEL** - Returns an approximation of Sobel edge enhancement.

*Result* = SOBEL(*Image*)

**SOCKET** - Opens client-side TCP/IP Internet socket as IDL file unit.

SOCKET, *Unit, Host, Port*  
 [, CONNECT\_TIMEOUT=*value*] [, ERROR=*variable*]  
 [, /GET\_LUN] [, /RAWIO] [, READ\_TIMEOUT=*value*]  
 [, /SWAP\_ENDIAN] [, /SWAP\_IF\_BIG\_ENDIAN]  
 [, /SWAP\_IF\_LITTLE\_ENDIAN] [, WIDTH=*value*]  
 [, WRITE\_TIMEOUT=*value*]

**UNIX-Only Keywords:** [, /STDIO]

**SORT** - Returns indices of an array sorted in ascending order.

*Result* = SORT(*Array* [, /L64 ])

**SPAWN** - Spawns child process for access to operating system.

SPAWN [, *Command* [, *Result*] [, *ErrResult* ] ]  
**Keywords (all platforms):** [, COUNT=*variable*]  
 [, EXIT\_STATUS=*variable*] [, PID=*variable*]

**UNIX-Only Keywords:** [, /NOSHELL]

[, /NOTTYRESET] [, /NULL\_STDIN] [, /SH]  
 [, /STDERR] [, UNIT=*variable* {*Command* required,  
*Result* not allowed}]

**Windows-Only Keywords:** [, /HIDE] [, /LOG\_OUTPUT]

[, /NOSHELL] [, /NOWAIT] [, /NULL\_STDIN]  
 [, /STDERR]

**SPH\_4PNT** - Returns center and radius of a sphere given 4 points.

SPH\_4PNT, *X, Y, Z, Xc, Yc, Zc, R* [, /DOUBLE]

**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*] )

**SPHER\_HARM** - Returns value of the spherical harmonic function.

*Result*=SPHER\_HARM( *Theta, Phi, L, M* [, /DOUBLE] )

**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.

*Result* = SPL\_INTERP( *X, Y, Y2, X2* [, /DOUBLE] )

**SPLINE** - Performs cubic spline interpolation.

*Result* = SPLINE( *X, Y, T* [, *Sigma*] [, /DOUBLE] )

**SPLINE\_P** - Performs parametric cubic spline interpolation.

SPLINE\_P, *X, Y, Xr, Yr* [, /DOUBLE]  
 [, INTERVAL=*value*] [, TANO=*[X<sub>0</sub>, Y<sub>0</sub>]*] [, TAN1=*[X<sub>n-1</sub>,*  
*Y<sub>n-1</sub>]*]

**SPRSAB** - Performs matrix multiplication on sparse matrices.

*Result* = SPRSAB(*A, B* [, /DOUBLE]  
 [, THRESHOLD=*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]  
 [, THRESHOLD=*value*] ) or  
*Result* = SPRSIN( *Columns, Rows, Values, N*  
 [, /DOUBLE] [, THRESHOLD=*value*] )

**SPRSTP** - Constructs the transpose of a sparse matrix.

*Result* = SPRSTP(*A*)

**SQRT** - Returns the square root of *X*.

*Result* = SQRT(*X* [, *Thread pool keywords*] )

**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<sub>1</sub>, ..., Expr<sub>n</sub>*]

**STRARR** - Returns string array containing zero-length strings.

*Result* = STRARR(*D<sub>1</sub>* [, ..., *D<sub>g</sub>*])

**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<sub>1</sub>, ..., Expression<sub>n</sub>*  
 [, 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*]  
[, COUNT=*variable*] [, 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<sub>1</sub>* [, *Arg<sub>2</sub>*, ..., *Arg<sub>n</sub>*]

**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]  
[, ITMAX=*value*]

**SVDFIT** - Multivariate least squares fit using SVD method.

*Result* = SVDFIT( *X*, *Y* [, *M*] [, A=*vector*]  
[, CHISQ=*variable*] [, COVAR=*variable*] [, /DOUBLE]  
[, FUNCTION\_NAME=*string*] [, /LEGENDRE]

[, MEASURE\_ERRORS=*vector*] [, SIGMA=*variable*]  
[, SING\_VALUES=*variable*] [, SINGULAR=*variable*]  
[, STATUS=*variable*] [, TOL=*value*]  
[, VARIANCE=*variable*] [, 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*  
[, /SWAP\_IF\_BIG\_ENDIAN]  
[, /SWAP\_IF\_LITTLE\_ENDIAN] )

**SWAP\_ENDIAN\_INPLACE** - Reverses the byte ordering of scalars, arrays or structures.

SWAP\_ENDIAN\_INPLACE, *Variable*  
[, /SWAP\_IF\_BIG\_ENDIAN]  
[, /SWAP\_IF\_LITTLE\_ENDIAN]

**SWITCH** - Selects one statement for execution from multiple choices, depending upon the value of an expression.

SWITCH expression OF  
expression: statement  
expression: statement  
ELSE: statement  
ENDSWITCH

**SYSTIME** - Returns the current time as either a string, as the number of seconds elapsed since 1 January 1970, or as a Julian d value.

*String* = SYSTIME( [0 [, *ElapsedSeconds*]] [, /UTC] )  
or  
*Seconds* = SYSTIME( 1 | /SECONDS )  
or  
*Julian* = SYSTIME( /JULIAN [, /UTC] )

## 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 [, *Array*] [, /RESET] [, MATRIX=*variable*]  
[, OBLIQUE=*vector*] [, PERSPECTIVE=*p*{eye at  
(0,0,*p*)}] [, ROTATE=*[x, y, z]*] [, SCALE=*[x, y, z]*]  
[, TRANSLATE=*[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* [, *Thread pool keywords*] )

**TANH** - Returns the hyperbolic tangent of X.

*Result* = TANH(*X* [, *Thread pool keywords*] )

**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*]

**TIMEGEN** - Returns an array of double-precision floating-point values that represent times in Julian values.

Result = TIMEGEN( [*D1*, ..., *D8*] [, *FINAL=value*] [, *DAYS=vector*] [, *HOURS=vector*] [, *MINUTES=vector*] [, *MONTHS=vector*] [, *SECONDS=vector*] [, *START=value*] [, *STEP\_SIZE=value*] [, *UNITS=string*] [, *YEAR=value*] )

**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*] [, */INTEGER*] [, */NAN*] [, */PRESERVE\_TYPE*] [, *Thread pool keywords*] )

**TRACE** - Computes the trace of an array.

Result = TRACE(*A* [, */DOUBLE*] )

**TrackBall Object** - See “TrackBall” on page 122.

**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={ystart*, *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*] [, *SPHERE=variable*] [, */DEGREES*] [, *FVALUE=variable*] [, *REPEATS=variable*] [, *TOLERANCE=value*]

**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*] [, *XOUT=vector*, *YOUT=vector*]

**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*] )

**TRUNCATE\_LUN** - Truncates an open file at the location of the current file pointer.

TRUNCATE\_LUN, *Unit<sub>1</sub>*, ..., *Unit<sub>n</sub>*

**TS\_COEF** - Computes the coefficients for autoregressive time-series.

Result = TS\_COEF(*X*, *P* [, */DOUBLE*] [, *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*] [, */ORDER*] [, *TRUE={1 | 2 | 3}*] [, */WORDS*] [, *XSIZE=value*] [, *YSIZE=value*]

**Graphics Keywords:** [ , CHANNEL=*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]

**Graphics Keywords:** [ , /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] [ , /NAN] [ , /ORDER] [ , TOP=*value*] [ , TRUE={1 | 2 | 3}] [ , /WORDS] [ , XSIZE=*value*] [ , YSIZE=*value*]

**Graphics Keywords:** [ , CHANNEL=*value*] [ , /DATA | , /DEVICE | , /NORMAL] [ , /T3D | Z=*value*] [ , Thread pool keywords]

## U

**UINDGEN** - Returns unsigned integer array with each element set to its subscript.

Result = UINDGEN( $D_1$  [ , ...,  $D_8$ ] [ , Thread pool keywords] )

**UINT** - Converts argument to unsigned integer type.

Result = UINT( Expression [ , Offset [ ,  $D_1$  [ , ...,  $D_8$ ]]] [ , Thread pool keywords] )

**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( $D_1$  [ , ...,  $D_8$ ] [ , Thread pool keywords] )

**ULINDGEN** - Returns an unsigned longword array with each element set to its subscript.

Result = ULINDGEN( $D_1$  [ , ...,  $D_8$ ] [ , Thread pool keywords] )

**ULON64ARR** - Returns an unsigned 64-bit integer vector or array.

Result = ULON64ARR(  $D_1$  [ , ...,  $D_8$ ] [ , /NOZERO] )

**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 [ ,  $D_1$  [ , ...,  $D_8$ ]]] [ , Thread pool keywords] )

**ULONG64** - Converts argument to unsigned 64-bit integer type.

Result = ULONG64( Expression [ , Offset [ ,  $D_1$  [ , ...,  $D_8$ ]]] [ , Thread pool keywords] )

**UNIQ** - Returns subscripts of the unique elements in an array.

Result = UNIQ( Array [ , Index] )

**UNSHARP\_MASK** - Performs an unsharp-mask sharpening filter on a two-dimensional array or a TrueColor image.

Result = UNSHARP\_MASK(Image [ , AMOUNT=*value*] [ , RADIUS=*value*] [ , THRESHOLD=*value*] [ , TRUE={1 | 2 | 3}] )

**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 [ , /L64] )

**VARIANCE** - Computes the statistical variance of an  $n$ -element vector.

Result = VARIANCE( X [ , /DOUBLE] [ , /NAN] )

**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*] [ , /OVERPLOT] [Also accepts all PLOT keywords]

**VERT\_T3D** - Transforms a 3D array by a 4x4 transformation matrix.

Result = VERT\_T3D( Vertex\_List [ , DOUBLE=*value*] [ , MATRIX=*4x4\_array*] [ , /NO\_COPY] [ , /NO\_DIVIDE] [ , SAVE\_DIVIDE=*variable*] )

**VOIGT** - Calculates intensity of atomic absorption line (Voigt) profile.

Result = VOIGT(A, U [ , Thread pool keywords] )

**VORONOI** - Computes Voronoi polygon given Delaunay triangulation.

VORONOI, *X*, *Y*, *IO*, *C*, *Xp*, *Yp*, *Rect*

**VOXEL\_PROJ** - Creates 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*  
[, /EXTRAPOLATE] [, *OUTPUT\_SIZE=vector*]  
[, /QUINTIC] [, /TPS] )

**WATERSHED** - Applies the morphological watershed operator to a grayscale image.

Result = WATERSHED ( *Image*  
[, *CONNECTIVITY={4 | 8}*] [, /LONG]  
[, *NREGIONS=variable*] )

**WDELETE** - Deletes IDL graphics windows.

WDELETE [, *Window\_Index* [, ...]]

**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 |, /NORMAL] [, *INTERVAL=value*]  
[, *PSYM=value*] [, *SYM\_HT=value*]  
[, *SYM\_LEN=value*] [, *THICK=value*]

**WHERE** - Returns subscripts of nonzero array elements.

Result = WHERE( *Array\_Expression* [, *Count*]  
[, *COMPLEMENT=variable*] [, /L64]  
[, *NCOMPLEMENT=variable*] [, *Thread pool keywords*] )

**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\_ACTIVEX** - Create an ActiveX control and place it into an IDL widget hierarchy.

Result = WIDGET\_ACTIVEX( *Parent*, *COM\_ID*,  
[, /ALIGN\_BOTTOM |, /ALIGN\_CENTER |,

/ALIGN\_LEFT |, /ALIGN\_RIGHT |, /ALIGN\_TOP]  
[, *EVENT\_FUNC=string*] [, *EVENT\_PRO=string*]  
[, *FUNC\_GET\_VALUE=string*] [*ID\_TYPE=value*]  
[, *KILL\_NOTIFY=string*] [, /NO\_COPY]  
[, *NOTIFY\_REALIZE=string*]  
[, *PRO\_SET\_VALUE=string*] [, *SCR\_XSIZE=width*]  
[, *SCR\_YSIZE=height*] [, /SENSITIVE]  
[, *UNAME=string*] [, *UNITS={0 | 1 | 2}*]  
[, *UVALUE=value*] [, *XOFFSET=value*]  
[, *XSIZE=value*] [, *YOFFSET=value*] [, *YSIZE=value*] )

**WIDGET\_BASE** - Creates base widget (containers for other widgets).

Result = WIDGET\_BASE( *Parent* [, /ALIGN\_BOTTOM  
|, /ALIGN\_CENTER |, /ALIGN\_LEFT |,  
/ALIGN\_RIGHT |, /ALIGN\_TOP] [, *MBAR=variable* |  
/MODAL] [, /BASE\_ALIGN\_BOTTOM |,  
/BASE\_ALIGN\_CENTER |, /BASE\_ALIGN\_LEFT |,  
/BASE\_ALIGN\_RIGHT |, /BASE\_ALIGN\_TOP]  
[, *BITMAP=value*] [, *COLUMN=ncols*] [, *ROW=nrows*]  
[, /CONTEXT\_EVENTS] [, /CONTEXT\_MENU]  
[, *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}]  
[, /MASK] [, /NO\_COPY]  
[, *NOTIFY\_REALIZE=string*]

**WIDGET\_BASE** - continued

[, *PRO\_SET\_VALUE=string*] [, *SCR\_XSIZE=width*]  
[, *SCR\_YSIZE=height*] [, /SCROLL{not for modal  
bases}] [, /SENSITIVE] [, *SPACE=value*{ignored if  
exclusive or nonexclusive}] [, *TAB\_MODE=value*]  
[, *TITLE=string*] [, *TLB\_FRAME\_ATTR=value*{top-  
level bases only}] [, /TLB\_ICONIFY\_EVENTS{top-  
level bases only}]  
[, /TLB\_KILL\_REQUEST\_EVENTS{top-level bases  
only}] [, /TLB\_MOVE\_EVENTS{top-level bases only}]  
[, /TLB\_SIZE\_EVENTS{top-level bases only}]  
[, /TOOLBAR] [, /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*  
[, /ACCELERATOR=string] [, /ALIGN\_CENTER |,  
/ALIGN\_LEFT |, /ALIGN\_RIGHT] [, /BITMAP]

[, /CHECKED\_MENU] [, /DYNAMIC\_RESIZE]  
 [, EVENT\_FUNC=*string*] [, EVENT\_PRO=*string*]  
 [, FONT=*string*] [, 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*]  
 [, /PUSHBUTTON\_EVENTS] [, SCR\_XSIZE=*width*]  
 [, SCR\_YSIZE=*height*] [, /SENSITIVE]  
 [, /SEPARATOR] [, TAB\_MODE=*value*]  
 [, TOOLTIP=*string*] [, /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\_COMBOBOX** - Creates editable droplist widgets.

*Result* = WIDGET\_COMBOBOX(*Parent*  
 [, /DYNAMIC\_RESIZE] [, /EDITABLE]  
 [, EVENT\_FUNC=*string*] [, EVENT\_PRO=*string*]  
 [, FONT=*string*] [, FRAME=*value*]  
 [, FUNC\_GET\_VALUE=*string*]  
 [, GROUP\_LEADER=*widget\_id*]  
 [, IGNORE\_ACCELERATORS=*value*]  
 [, KILL\_NOTIFY=*string*] [, /NO\_COPY]  
 [, NOTIFY\_REALIZE=*string*]  
 [, PRO\_SET\_VALUE=*string*]

**WIDGET\_COMBOBOX** - *continued*

[, RESOURCE\_NAME=*string*] [, SCR\_XSIZE=*width*]  
 [, SCR\_YSIZE=*height*] [, /SENSITIVE]  
 [, TAB\_MODE=*value*] [, /TRACKING\_EVENTS]  
 [, UNAME=*string*] [, UNITS={0 | 1 | 2}]  
 [, UVALUE=*value*] [, VALUE=*value*]  
 [, XOFFSET=*value*] [, XSIZE=*value*]  
 [, YOFFSET=*value*] [, YSIZE=*value*]

**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*] [, /HOURLASS{do  
 not specify *Widget\_ID*}] [, KILL\_NOTIFY=*string*]  
 [, /MAP] [, /NO\_COPY] [, NOTIFY\_REALIZE=*string*]  
 [, PRO\_SET\_VALUE=*string*]  
 [, /PUSHBUTTON\_EVENTS] [, /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:** [, /CONTEXT\_EVENTS]  
 [, /ICONIFY] [, /KBRD\_FOCUS\_EVENTS]  
 [, TAB\_MODE=*value*] [, /TLB\_ICONIFY\_EVENTS]  
 [, /TLB\_KILL\_REQUEST\_EVENTS]  
 [, /TLB\_MOVE\_EVENTS] [, /TLB\_SIZE\_EVENTS]

**WIDGET\_BUTTON:** [, /BITMAP]

[, /DYNAMIC\_RESIZE] [, SET\_VALUE=*value*]  
 [, /INPUT\_FOCUS] [, /PUSHBUTTON\_EVENTS]  
 [, /SET\_BUTTON] [, SET\_VALUE=*value*]  
 [, TAB\_MODE=*value*] [, TOOLTIP=*string*]  
 [, X\_BITMAP\_EXTRA=*bits*]

**WIDGET\_COMBOBOX:**

[, COMBOBOX\_ADDITEM=*string*]  
 [, COMBOBOX\_DELETEITEM=*integer*]  
 [, COMBOBOX\_INDEX=*integer*]  
 [, /DYNAMIC\_RESIZE] [, GET\_VALUE=*value*]  
 [, IGNORE\_ACCELERATORS={*string\_array*{0 | 1}}]  
 [, SET\_COMBOBOX\_SELECT=*integer*]  
 [, SET\_VALUE=*value*] [, TAB\_MODE=*value*]

**WIDGET\_CONTROL** - *continued*

**WIDGET\_DRAW:** [, /DRAW\_BUTTON\_EVENTS]  
 [, /DRAW\_EXPOSE\_EVENTS]  
 [, DRAW\_KEYBOARD\_EVENTS={0 | 1 | 2}]  
 [, /DRAW\_MOTION\_EVENTS]  
 [, /DRAW\_VIEWPORT\_EVENTS]  
 [, /DRAW\_WHEEL\_EVENTS]  
 [, DRAW\_XSIZE=*integer*] [, DRAW\_YSIZE=*integer*]  
 [, GET\_DRAW\_VIEW=*variable*]  
 [, GET\_UVALUE=*variable*] [, GET\_VALUE=*variable*]  
 [, IGNORE\_ACCELERATORS={*string\_array*{0 | 1}}]  
 [, /INPUT\_FOCUS] [, SET\_DRAG\_NOTIFY=*string*]  
 [, SET\_DRAW\_VIEW=*[x, y]*] [, /SET\_DROP\_EVENTS]  
 [, TOOLTIP=*string*]

**WIDGET\_DROPLIST:** [, /DYNAMIC\_RESIZE]

[, GET\_VALUE=*value*]  
 [, SET\_DROPLIST\_SELECT=*integer*]  
 [, SET\_VALUE=*value*] [, TAB\_MODE=*value*]

**WIDGET\_LABEL:** [, /DYNAMIC\_RESIZE]

[, GET\_VALUE=*value*] [, SET\_VALUE=*value*]

**WIDGET\_LIST:** [, /CONTEXT\_EVENTS]

[, SET\_LIST\_SELECT=*value*]

[, SET\_LIST\_TOP=*integer*] [, SET\_VALUE=*value*]  
 [, TAB\_MODE=*value*]

**WIDGET\_PROPERTY SHEET:**

[, /CONTEXT\_EVENTS] [, /EDITABLE]  
 [, GET\_VALUE=*variable*]  
 [, IGNORE\_ACCELERATORS={*string\_array*{0 | 1}}]  
 [, /MULTIPLE\_PROPERTIES]  
 [, PROPERTY SHEET\_SELECTED=*empty string*,  
*string*, or *array of strings*]  
 [, REFRESH\_PROPERTY=*string*, *string array*, or  
*integer*] [, SET\_VALUE=*value*]

**WIDGET\_SLIDER:** [, GET\_VALUE=*value*]

[, SET\_SLIDER\_MAX=*value*]  
 [, SET\_SLIDER\_MIN=*value*] [, SET\_VALUE=*value*]  
 [, TAB\_MODE=*value*]

**WIDGET\_TAB:** [, SET\_TAB\_CURRENT=*index*]

[, SET\_TAB\_MULTILINE=*value*]  
 [, TAB\_MODE=*value*]

**WIDGET\_CONTROL - continued**

**WIDGET\_TABLE:** [, ALIGNMENT={0 | 1 | 2}]  
 [, /ALL\_TABLE\_EVENTS] [, AM\_PM={*string*, *string*}]  
 [, BACKGROUND\_COLOR=*array*]  
 [, COLUMN\_LABELS=*string\_array*]  
 [, COLUMN\_WIDTHS=*array*]  
 [, /CONTEXT\_EVENTS]  
 [, DAYS\_OF\_WEEK=*string\_array*{7 names}]  
 [, /DELETE\_COLUMNS{not for row\_major mode}]  
 [, /DELETE\_ROWS{not for column\_major mode}]  
 [, /DISJOINT\_SELECTION] [, /EDITABLE]  
 [, EDIT\_CELL={*integer*, *integer*}]  
 [, FONT=*string array*]  
 [, FOREGROUND\_COLOR=*array*] [, FORMAT=*value*]  
 [, GET\_VALUE=*variable*]  
 [, IGNORE\_ACCELERATORS={*string\_array*{0 | 1}}]  
 [, 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*] [, TAB\_MODE=*value*]  
 [, TABLE\_BLANK=*cells*]  
 [, /TABLE\_DISJOINT\_SELECTION]  
 [, 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] [, /CONTEXT\_EVENTS] [, /EDITABLE]  
 [, GET\_VALUE=*variable*]  
 [, IGNORE\_ACCELERATORS={*string\_array*{0 | 1}}]

[, /INPUT\_FOCUS] [, /KBRD\_FOCUS\_EVENTS]  
 [, /NO\_NEWLINE] [, SET\_TEXT\_SELECT={*integer*,  
*integer*}] [, SET\_TEXT\_TOP\_LINE=*line\_number*]  
 [, SET\_VALUE=*value*] [, TAB\_MODE=*value*]  
 [, /USE\_TEXT\_SELECT]

**WIDGET\_TREE:** [, /CONTEXT\_EVENTS]

[, GET\_VALUE=*variable*]  
 [, SET\_DRAG\_NOTIFY=*string*]  
 [, /SET\_DRAGGABLE] [, /SET\_DROP\_EVENTS]  
 [, /SET\_MASK] [, SET\_TREE\_BITMAP=*array*]  
 [, /SET\_TREE\_EXPANDED]  
 [, SET\_TREE\_INDEX=*value*] [, SET\_TREE\_SELECT=  
 {0 | 1 | *widget ID* | *array of widget IDs*}]  
 [, /SET\_TREE\_VISIBLE] [, SET\_VALUE=*value*]  
 [, TAB\_MODE=*value*]

**WIDGET\_DISPLAYCONTEXTMENU** - Displays a context-sensitive menu.

WIDGET\_DISPLAYCONTEXTMENU, *Parent*, *X*, *Y*,  
*ContextBase\_ID*

**WIDGET\_DRAW** - Creates drawable widgets.

*Result* = WIDGET\_DRAW(*Parent* [, /APP\_SCROLL]  
 [, /BUTTON\_EVENTS] [, CLASSNAME=*string*]  
 [, /COLOR\_MODEL] [, COLORS=*integer*]  
 [, DRAG\_NOTIFY=*string*] [, /DROP\_EVENTS]  
 [, EVENT\_FUNC=*string*] [, EVENT\_PRO=*string*]  
 [, /EXPOSE\_EVENTS] [, FRAME=*width*]  
 [, FUNC\_GET\_VALUE=*string*]  
 [, GRAPHICS\_LEVEL=2]  
 [, GROUP\_LEADER=*widget\_id*]  
 [, IGNORE\_ACCELERATORS=*value*]  
 [, KEYBOARD\_EVENTS={0 | 1 | 2}]  
 [, 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] [, TOOLTIP=*string*]  
 [, /TRACKING\_EVENTS] [, UNAME=*string*]  
 [, UNITS={0 | 1 | 2}] [, UVALUE=*value*]  
 [, /VIEWPORT\_EVENTS] [, /WHEEL\_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*] [, FONT=*string*]  
 [, FRAME=*value*] [, 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]  
 [, TAB\_MODE=*value*] [, 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\_HOURLASS]

**UNIX Keywords:** [, /YIELD\_TO\_TTY]

**WIDGET\_INFO** - Obtains information about widgets.

*Result* = WIDGET\_INFO([*Widget\_ID*])  
**All widgets:** [, /ACTIVE] [, /ALL\_CHILDREN]  
 [, /CHILD] [, /EVENT\_FUNC] [, /EVENT\_PRO]  
 [, FIND\_BY\_UNAME=*string*] [, /FONTNAME]  
 [, /GEOMETRY] [, /KBRD\_FOCUS\_EVENTS]  
 [, /MANAGED] [, /MAP] [, /N\_CHILDREN] [, /NAME]  
 [, /PARENT] [, /PUSHBUTTON\_EVENTS]  
 [, /REALIZED] [, /SENSITIVE] [, /SIBLING]  
 [, STRING\_SIZE={*string* | [*string*, *font*]}]  
 [, /SYSTEM\_COLORS] [, TAB\_MODE=*value*]  
 [, /TRACKING\_EVENTS] [, /TYPE] [, UNITS={0 | 1 |  
 2}] [, /UNAME] [, /UPDATE] [, /VALID\_ID]  
 [, /VERSION] [, /VISIBLE]

**WIDGET\_BASE:** [, /CONTEXT\_EVENTS]  
 [, /MODAL] [, /TLB\_ICONIFY\_EVENTS]  
 [, /TLB\_KILL\_REQUEST\_EVENTS]  
 [, /TLB\_MOVE\_EVENTS] [, /TLB\_SIZE\_EVENTS]

**WIDGET\_BUTTON:** [, /BUTTON\_SET]  
 [, /DYNAMIC\_RESIZE] [, /PUSHBUTTON\_EVENTS]  
 [, /TOOLTIP]

**WIDGET\_COMBOBOX:** [, /COMBOBOX\_GETTEXT]  
 [, /COMBOBOX\_NUMBER] [, /DYNAMIC\_RESIZE]

**WIDGET\_DRAW:** [, /DRAG\_NOTIFY]  
 [, /DRAW\_BUTTON\_EVENTS]  
 [, /DRAW\_EXPOSE\_EVENTS]  
 [, /DRAW\_KEYBOARD\_EVENTS]  
 [, /DRAW\_MOTION\_EVENTS]  
 [, /DRAW\_VIEWPORT\_EVENTS]  
 [, /DRAW\_WHEEL\_EVENTS] [, /DROP\_EVENTS]  
 [, /TOOLTIP]

**WIDGET\_DROPLIST:** [, /DROPLIST\_NUMBER]  
 [, /DROPLIST\_SELECT] [, /DYNAMIC\_RESIZE]

**WIDGET\_LABEL:** [, /DYNAMIC\_RESIZE]

**WIDGET\_LIST:** [, /CONTEXT\_EVENTS]  
 [, /LIST\_MULTIPLE] [, /LIST\_NUMBER]  
 [, /LIST\_NUM\_VISIBLE] [, /LIST\_SELECT]  
 [, /LIST\_TOP]

**WIDGET\_PROPERTY SHEET:**  
 [, /CONTEXT\_EVENTS] [, COMPONENT=*objref*]  
 [, /MULTIPLE\_PROPERTIES]  
 [, PROPERTY\_VALID=*string*]

[, PROPERTY\_VALUE=*string*]  
 [, /PROPERTY SHEET\_NSELECTED]  
 [, /PROPERTY SHEET\_SELECTED]  
**WIDGET\_SLIDER:** [, /SLIDER\_MIN\_MAX]  
**WIDGET\_TAB:** [, /TAB\_CURRENT]  
 [, /TAB\_MULTILINE] [, /TAB\_NUMBER]  
**WIDGET\_TABLE:** [, /COLUMN\_WIDTHS]  
 [, /CONTEXT\_EVENTS] [, /ROW\_HEIGHTS]  
 [, /STRING\_SIZE] [, /TABLE\_ALL\_EVENTS]  
 [, /TABLE\_BACKGROUND\_COLOR]  
 [, /TABLE\_DISJOINT\_SELECTION]  
 [, /TABLE\_EDITABLE] [, /TABLE\_EDIT\_CELL]  
 [, /TABLE\_FONT]  
 [, /TABLE\_FOREGROUND\_COLOR]  
 [, /TABLE\_SELECT] [, /TABLE\_VIEW]  
 [, /USE\_TABLE\_SELECT]

**WIDGET\_TEXT:** [, /CONTEXT\_EVENTS]  
 [, /TEXT\_ALL\_EVENTS] [, /TEXT\_EDITABLE]  
 [, /TEXT\_NUMBER]  
 [, TEXT\_OFFSET\_TO\_XY=*integer*]  
 [, /TEXT\_SELECT] [, /TEXT\_TOP\_LINE]  
 [, TEXT\_XY\_TO\_OFFSET=[*column*, *line*]]  
**WIDGET\_TREE:** [, /CONTEXT\_EVENTS]  
 [, /DRAGGABLE] [, /DRAG\_NOTIFY]  
 [, /DROP\_EVENTS] [, /MASK] [, /TREE\_BITMAP]  
 [, /TREE\_DRAG\_SELECT] [, /TREE\_EXPANDED]  
 [, /TREE\_FOLDER] [, /TREE\_INDEX]  
 [, /TREE\_ROOT] [, /TREE\_SELECT]

**WIDGET\_LABEL** - Creates label widgets.

*Result* = WIDGET\_LABEL(*Parent* [, /ALIGN\_CENTER  
 |, /ALIGN\_LEFT |, /ALIGN\_RIGHT]  
 [, /DYNAMIC\_RESIZE] [, FONT=*string*]  
 [, 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]  
 [, /SUNKEN\_FRAME] [, /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*  
 [, /CONTEXT\_EVENTS] [, EVENT\_FUNC=*string*]  
 [, EVENT\_PRO=*string*] [, FONT=*string*]  
 [, FRAME=*width*] [, FUNC\_GET\_VALUE=*string*]  
 [, GROUP\_LEADER=*widget\_id*]  
 [, KILL\_NOTIFY=*string*] [, /MULTIPLE]  
 [, /NO\_COPY] [, NOTIFY\_REALIZE=*string*]



**WIDGET\_LIST** - *continued*

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

**WIDGET\_PROPERTY SHEET** - Creates a property sheet widget, which exposes the properties of an IDL object.. This widget transparently handles property value changes

```
Result = WIDGET_PROPERTY SHEET(Parent
[, /ALIGN_BOTTOM |, /ALIGN_CENTER
|, /ALIGN_LEFT |, /ALIGN_RIGHT |, /ALIGN_TOP]
[, /CONTEXT_EVENTS] [, /EDITABLE]
[, EVENT_FUNC=string] [, EVENT_PRO=string]
[, FONT=STRING] [, FRAME=width]
[, FUNC_GET_VALUE=string]
[, IGNORE_ACCELERATORS=value]
[, KILL_NOTIFY=string]
[, /MULTIPLE_PROPERTIES] [, /NO_COPY]
[, NOTIFY_REALIZE=string]
[, PRO_SET_VALUE=string] [, SCR_XSIZE=width]
[, SCR_YSIZE=height] [, /SENSITIVE]
[, /SUNKEN_FRAME] [, /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=string] [, 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]
[, TAB_MODE=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\_TAB** - Creates tab widgets.

```
Result = WIDGET_TAB( Parent [, /ALIGN_BOTTOM |,
/ALIGN_CENTER |, /ALIGN_LEFT |,
/ALIGN_RIGHT |, /ALIGN_TOP]
[, EVENT_FUNC=string] [, EVENT_PRO=string]
[, FUNC_GET_VALUE=string]
[, GROUP_LEADER=widget_id]
[, KILL_NOTIFY=string] [, LOCATION={0 | 1 | 2 | 3}]
[, MULTILINE=0 | 1 (Windows) or num tabs per row
(Motif)] [, /NO_COPY] [, NOTIFY_REALIZE=string]
[, PRO_SET_VALUE=string] [, SCR_XSIZE=width]
[, SCR_YSIZE=height] [, /SENSITIVE]
[, TAB_MODE=value] [, /TRACKING_EVENTS]
[, UNAME=string] [, UNITS={0 | 1 | 2}]
[, UVALUE=value] [, 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]]
[, BACKGROUND_COLOR=array]
[, COLUMN_LABELS=string_array]
[, /COLUMN_MAJOR |, /ROW_MAJOR]
[, COLUMN_WIDTHS=array]
[, /CONTEXT_EVENTS]
[, DAYS_OF_WEEK=string_array{7 names}]
[, /DISJOINT_SELECTION] [, /EDITABLE]
[, EVENT_FUNC=string] [, EVENT_PRO=string]
[, FONT=string] [, FOREGROUND_COLOR=array]
[, FORMAT=value] [, FRAME=width]
[, FUNC_GET_VALUE=string]
[, GROUP_LEADER=widget_id]
[, IGNORE_ACCELERATORS=value]
[, /KBRD_FOCUS_EVENTS] [, KILL_NOTIFY=string]
[, MONTHS=string_array{12 names}]
[, /NO_COLUMN_HEADERS] [, /NO_COPY]
[, /NO_HEADERS] [, /NO_ROW_HEADERS]
[, NOTIFY_REALIZE=string]
[, PRO_SET_VALUE=string]
[, /RESIZEABLE_COLUMNS]
[, /RESIZEABLE_ROWS{not supported in Windows}]
[, RESOURCE_NAME=string]
[, ROW_HEIGHTS=array]
[, ROW_LABELS=string_array] [, SCR_XSIZE=width]
[, SCR_YSIZE=height] [, /SCROLL] [, /SENSITIVE]
[, TAB_MODE=value] [, /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]
[, /CONTEXT_EVENTS] [, /EDITABLE]
[, EVENT_FUNC=string] [, EVENT_PRO=string]
[, FONT=string] [, FRAME=width]
[, FUNC_GET_VALUE=string]
[, GROUP_LEADER=widget_id]
[, IGNORE_ACCELERATORS=value]
[, /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]
[, TAB_MODE=value] [, /TRACKING_EVENTS]
[, UNAME=string] [, UNITS={0 | 1 | 2}]
[, UVALUE=value] [, VALUE=value] [, /WRAP]
[, XOFFSET=value] [, XSIZE=value]
[, YOFFSET=value] [, YSIZE=value] )
```

**WIDGET\_TREE** - Creates tree widgets.

```
Result = WIDGET_TREE( Parent [, /ALIGN_BOTTOM |
./ALIGN_CENTER | ./ALIGN_LEFT |
./ALIGN_RIGHT | ./ALIGN_TOP] [, BITMAP=array]
[, /CONTEXT_EVENTS] [, DRAG_NOTIFY=string]
[, /DRAGGABLE] [, /DROP_EVENTS]
[, EVENT_FUNC=string] [, EVENT_PRO=string]
[, /EXPANDED] [, /FOLDER]
[, FUNC_GET_VALUE=string]
[, GROUP_LEADER=widget_id] [, INDEX=value]
[, KILL_NOTIFY=string] [, /MASK] [, /MULTIPLE]
[, /NO_COPY] [, NOTIFY_REALIZE=string]
[, PRO_SET_VALUE=string] [, SCR_XSIZE=width]
[, SCR_YSIZE=height] [, /SENSITIVE]
[, TAB_MODE=value] [, /TRACKING_EVENTS]
[, UNAME=string] [, UNITS={0 | 1 | 2}]
[, UVALUE=value] [, VALUE=string]
[, XOFFSET=value] [, XSIZE=value]
[, YOFFSET=value] [, YSIZE=value] )
```

**WIDGET\_TREE\_MOVE** - Copies and moves tree widget nodes from one tree to another or within the same tree.

```
WIDGET_TREE_MOVE, wMoveNodes, wDestFolder
[, CALLBACK_FUNC=name] [, /COPY]
[, INDEX=index] [, /SELECT] [, /UPDATE]
[, USERDATA=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}] [, /RGB]
```

**WRITE\_GIF** - Writes a Graphics Interchange Format (GIF) file.

```
WRITE_GIF, Filename, Image[, R, G, B]
[, BACKGROUND_COLOR=value] [, /CLOSE]
[, DELAY_TIME=integer]
[, DISPOSAL_METHOD=integer] [, /MULTIPLE]
[, REPEAT_COUNT=integer]
[, TRANSPARENT=value] [, USER_INPUT=value]
```

**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 a JPEG file.

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

**WRITE\_JPEG2000** - Writes a JPEG2000 file.

```
WRITE_JPEG2000, Filename, Image [, Red, Green, Blue]
[, N_LAYERS=value] [, N_LEVELS=value] [, /ORDER]
[, /REVERSIBLE] )
```

**WRITE\_NRFIF** - Writes NCAR Raster Interchange Format rasterfile.

```
WRITE_NRFIF, 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] [, /ORDER]
```

**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] [, /APPEND]
[, BITS_PER_SAMPLE={1 | 4 | 8}] [, RED=value]
[, GREEN=value] [, BLUE=value] [, /CMYK]
[, COMPRESSION={0 | 1 | 2 | 3}]
[, DESCRIPTION=string]
[, DOCUMENT_NAME=string]
[, DOT_RANGE=intarray] [, GEOTIFF=structure]
[, ICC_PROFILE=array] [, /LONG | , /SHORT |
./FLOAT] [, ORIENTATION=value]
[, PHOTOSHOP=bytearray] [, PLANARCONFIG={1 |
2}] [, /VERBOSE] [, XPOSITION=units]
[, XRESOL=pixels/inch] [, YPOSITION=units]
[, YRESOL=pixels/inch]
```

**Note:** LZW compression (COMPRESSION=1) is only available with the appropriate license feature.

**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, Expr1 ..., Exprn
[, TRANSFER_COUNT=variable]
```

**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]
[, DONE_BUTTON=string] [, /EDITABLE]
[, FONT=string] [, GROUP=widget_id]
[, /GROW_TO_SCREEN] [, HEIGHT=lines]
[, /MODAL] [, RETURN_ID=variable] [, TEXT=string
or string array] [, TITLE=string] [, WIDTH=characters]
[, WTEXT=variable]
```

**XDXF** - Utility for displaying and interactively manipulating DXF objects

```
XDXF [, Filename] [, /BLOCK] [, GROUP=widget_id]
[, SCALE=value] [, /TEST] [keywords to XOBJVIEW]
```

**XFONT** - Creates modal widget to select and view an X Windows font.

```
Result = XFONT( [, GROUP=widget_id]
[, /PRESERVE_FONT_INFO] )
```

**XINTERANIMATE** - Displays animated sequence of images.

```
XINTERANIMATE [, Rate]
```

**Keywords for initialization:** [, SET=*fsize*, *size*, *nframes*]] [, /BLOCK] [, /CYCLE]

```
[, GROUP=widget_id] [, /MODAL] [,
MPEG_BITRATE=value] [, MPEG_FORMAT=value]
```

```
[, MPEG_IFRAME_GAP=integer value]
[, MPEG_MOTION_VEC_LENGTH={1 | 2 | 3}]
```

```
[, /MPEG_OPEN, MPEG_FILENAME=string]
[, MPEG_QUALITY=value{0 to 100}] [, /SHOWLOAD]
[, /TRACK] [, TITLE=string]
```

**Keywords for loading images:** [, FRAME=*value*{0 to (*nframes*-1)}], IMAGE=*value*]] [, /ORDER]
[, WINDOW=*[window\_num* [, *x0*, *y0*, *sx*, *sy*]]]

**Keywords for running animations:** [, /CLOSE]
[, /KEEP\_PIXMAPS] [, /MPEG\_CLOSE]
[, XOFFSET=*pixels*] [, YOFFSET=*pixels*]

**XLOADCT** - Provides GUI to interactively select and load color tables.

```
XLOADCT [, /BLOCK] [, BOTTOM=value]
[, FILE=string] [, GROUP=widget_id] [, /MODAL]
[, NCOLORS=value] [, /SILENT]
[, UPDATECALLBACK='procedure_name']
[, UPDATECBDATA=value]] [, /USE_CURRENT]
```

**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 [, BACKGROUND=[r, g, b]]
[, /BLOCK] [, /DOUBLE_VIEW] [, GROUP=widget_id]
[, .JUST_REG] [, /MODAL] [, REFRESH=widget_id]
[, RENDERER={0|1}] [, SCALE=value]
[, STATIONARY=objref(s)] [, /TEST] [, TITLE=string]
[, TLB=variable] [, XOFFSET=value] [, XSIZE=pixels]
[, YOFFSET=value] [, YSIZE=pixels]
```

**XOBJVIEW\_ROTATE** - Programmatically rotate the object currently displayed in XOBJVIEW.

XOBJVIEW\_ROTATE, *Axis*, *Angle* [, /PREMULTIPLY]

**XOBJVIEW\_WRITE\_IMAGE** - Write the object currently displayed in XOBJVIEW to an image file.

XOBJVIEW\_WRITE\_IMAGE, *Filename*, *Format*  
[, DIMENSIONS=*[x, y]*]

**XPALETTE** - Displays widget used to create and modify color tables.

XPALETTE [, /BLOCK] [, GROUP=*widget\_id*]  
[, UPDATECALLBACK=*'procedure\_name'*]  
[, UPDATECBDATA=*value*]

**XPCOLOR** - Adjusts the value of the current foreground plotting color, !P.COLOR.

XPCOLOR [, GROUP=*widget\_id*]

**XPLOT3D** - Utility for creating and interactively manipulating 3D plots.

XPLOT3D, *X*, *Y*, *Z* [, /BLOCK] [, COLOR=*[r,g,b]*]  
[, /DOUBLE\_VIEW] [, GROUP=*widget\_id*]  
[, LINSTYLE={0 | 1 | 2 | 3 | 4 | 5 | 6}] [, /MODAL]  
[, NAME=*string*] [, /OVERPLOT]  
[, SYMBOL=*objref(s)*] [, /TEST]  
[, THICK=*points*{1.0 to 10.0}] [, TITLE=*string*]  
[, XRANGE=*[min, max]*] [, YRANGE=*[min, max]*]  
[, ZRANGE=*[min, max]*] [, XTITLE=*string*]  
[, YTITLE=*string*] [, ZTITLE=*string*]

**XREGISTERED** - Returns registration status of a given widget.

*Result* = XREGISTERED(*Name* [, /NOSHOW] )

**XROI** - Utility for interactively creating and obtaining information about ROIs.

XROI [, *ImageData*] [, *R*] [, *G*] [, *B*] [, /BLOCK]  
[, /FLOATING] [, GROUP=*widget\_ID*] [, /MODAL]  
[, REGIONS\_IN=*value*] [, REGIONS\_OUT=*value*]  
[, REJECTED=*variable*] [, RENDERER={0 | 1}]  
[, ROI\_COLOR=*[r, g, b]* or *variable*]  
[, ROI\_GEOMETRY=*variable*]  
[, ROI\_SELECT\_COLOR=*[r, g, b]* or *variable*]  
[, STATISTICS=*variable*] [, TITLE=*string*]  
[, TOOLS=*string* or *string array*{valid values are  
'Freehand Draw', 'Polygon Draw', and 'Selection'}]  
[, X\_SCROLL\_SIZE=*vlaue*]  
[, Y\_SCROLL\_SIZE=*value*]

**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*]

**XVAREEDIT** - Provides widget-based editor for IDL variables.

XVAREEDIT, *Var* [, NAME=*'variable\_name'*{ignored if  
variable is a structure}] [, GROUP=*widget\_id*]  
[, X\_SCROLL\_SIZE=*columns*]  
[, Y\_SCROLL\_SIZE=*rows*]

**XVOLUME** - Utility for viewing and interactively manipulating volumes and isosurfaces.

XVOLUME, *Vol*, [, /BLOCK] [, GROUP=*widget\_id*]  
[, /INTERPOLATE] [, /MODAL] [, RENDERER={0 |  
1}] [, /REPLACE] [, SCALE=*value*] [, /TEST]  
[, XSIZE=*pixels*] [, YSIZE=*pixels*]

**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*]

**Graphics Keywords:** [, CLIP=*[X<sub>0</sub>, Y<sub>0</sub>, X<sub>1</sub>, Y<sub>1</sub>]*]  
[, COLOR=*value*] [, /DATA | /DEVICE | /NORMAL]  
[, FONT=*integer*]  
[, ORIENTATION=*ccw\_degrees\_from\_horiz*]  
[, /NOCLIP] [, /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

---

See the online version of this document, [quickref.pdf](#) located on your IDL CD-ROM for scientific data format routine information.

# Objects

This section lists all IDL objects and their methods. In addition to the syntax conventions discussed in “IDL Syntax Conventions” on page 28, 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 IDLgrView-group.

**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\_IDLBridge::SetVar** - Copies the value of a variable from the main IDL process to the child process under the given name.

**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]

**IDL\_IDLBridge** - Creates and controls a synchronous or an asynchronous child IDL process that can operate sequentially or in parallel with the main IDL process.

**Properties:** [, CALLBACK{Get, Init, Set}=*string*] [, OUTPUT{Get, Init, Set}=*filename*] [, USERDATA{Get, Init, Set}=*userdef*]

**IDL\_IDLBridge::Abort** - Aborts the current asynchronous child IDL process.

*Obj->*[IDL\_IDLBridge::]Abort

**IDL\_IDLBridge::Cleanup** - Performs all cleanup on the object.

OBJ\_DESTROY, *Obj* or *Obj->*[IDL\_IDLBridge::]Cleanup

**IDL\_IDLBridge::Execute** - Causes the child IDL process to execute a given IDL statement in synchronous or asynchronous mode.

*Obj->*[IDL\_IDLBridge::]Execute, *IDLStmt*, [, /NOWAIT]

**IDL\_IDLBridge::GetProperty** - Retrieves the value of a property or group of properties for the bridge.

*Obj->*[IDL\_IDLBridge::]GetProperty [, *PROPERTY=value*]

**IDL\_IDLBridge::GetVar** - Copies the value of a variable from the child process to the main process.

*Result = Obj->*[IDL\_IDLBridge::]GetVar, *VarName*

**IDL\_IDLBridge::Init** - Initializes an IDL bridge object that lets you create and control a child IDL session.

*Obj=OBJ\_NEW('IDL\_IDLBridge' [, *PROPERTY=value*])*  
or  
*Result = Obj->*[IDL\_IDLBridge::]Init ([, *PROPERTY=value*])

**IDL\_IDLBridge::OnCallback** - When an asynchronous command ens, this property is automatically called. If the CALLBACK property is set, the callback procedure specified by the property is executed.

*Obj->*[IDL\_IDLBridge::]OnCallback, *Status*, *Error*

**IDL\_IDLBridge::SetProperty** - Sets the value of a property or group of properties for the bridge.

*Obj->*[IDL\_IDLBridge::]SetProperty [, *PROPERTY=value*]

*Obj->*[IDL\_IDLBridge::]SetVar, *Name*, *Value*

**IDL\_IDLBridge::Status** - Returns the state of an asynchronous child process (one executed with the NOWAIT keyword set).

*Result* =  
*Obj*->[IDL\_IDLBridge::]Status([ERROR=*ErrorString*])

**IDL\_Savefile** - Object that provides complete query and restore capabilities for IDL SAVE files.

**Properties:** None.

**IDL\_Savefile::Cleanup** - Performs all cleanup on the object.

OBJ\_DESTROY, *Obj* or *Obj*->[IDL\_Savefile::]Cleanup

**IDL\_Savefile::Contents** - returns a structure variable of type IDL\_SAVEFILE\_CONTENTS containing information about the associated SAVE file and its contents.

*Result* = *Obj*->[IDL\_Savefile::]Contents()

**IDL\_Savefile::Init** - Initializes the Safefile object.

*Obj* = OBJ\_NEW('IDL\_Savefile' [, *Filename* ]  
 [, FILENAME=*string*]  
 [, /RELAXED\_STRUCTURE\_ASSIGNMENT])  
*Result* = *Obj*->[IDL\_Savefile::]Init( [, *Filename* ]  
 [, FILENAME=*string*]  
 [, /RELAXED\_STRUCTURE\_ASSIGNMENT] )

**IDL\_Savefile::Names** - Returns names, or heap variable identifiers, of items contained within the SAVE file.

*Result* = *Obj*->[IDL\_SAVEFILE::]Names(  
 [, COUNT=*variable*] [, /COMMON\_BLOCK ]  
 , COMMON\_VARIABLE=*string* | , /FUNCTION |  
 , /OBJECT\_HEAPVAR | , /POINTER\_HEAPVAR |  
 , /PROCEDURE | , /STRUCTURE\_DEFINITION |  
 , /SYSTEM\_VARIABLE ] )

**IDL\_Savefile::Restore** - Selectively restores individual items from the associated SAVE file.

*Obj*->[IDL\_Savefile::]Restore, *SaveItem* [, /COMMON]  
 [, /FUNCTION] [, NEW\_HEAPVAR=*variable*]  
 [, /OBJECT\_HEAPVAR] [, /POINTER\_HEAPVAR]  
 [, /PROCEDURE] [, RESTORED\_OBJECTS=*variable*]  
 [, /STRUCTURE\_DEFINITION] [, /VERBOSE]

**IDL\_Savefile::Size** - Returns the size and type information for the specified variable, system variable, or heap variable in the SAVE file .

*Result* = *Obj*->[IDL\_Savefile::]Size(*SaveItem*)

**IDLanROI** - Represents a region of interest. Superclass of IDLgrROI.

**Properties:** [, ALL{Get}=*variable*] [, BLOCKSIZE{Get, Init, Set}=*vertices*] [, DATA{Get, Init, Set}=*array*]  
 [, DOUBLE{Get, Init, Set}=*value*] [, /INTERIOR{Get, Init, Set}] [, N\_VERTS{Get}=*variable*]  
 [, ROI\_XRANGE{Get}=*variable*]  
 [, ROI\_YRANGE{Get}=*variable*]  
 [, ROI\_ZRANGE{Get}=*variable*] [, TYPE{Get, Init}={ 0 | 1 | 2 }]

**IDLanROI::SetProperty** - Sets the value of a property or group of properties for the region.

**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 } ]  
 [, PIXEL\_CENTER={*x*, *y*} [, PLANE\_NORMAL={*x*, *y*,  
*z*} [, PLANE\_XAXIS={*x*, *z*} ]  
 [, RUN\_LENGTH=*value*] )

**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[, *PROPERTY*=*value*]

**IDLanROI::Init** - Initializes a region of interest object.

*Obj* = OBJ\_NEW( 'IDLanROI' [, X [, Y [, Z ]]]  
 [, *PROPERTY*=*value*] ) or  
*Result* = *Obj*->[IDLanROI::]Init( [X [, Y [, Z ]]]  
 [, *PROPERTY*=*value*] )

**IDLanROI::RemoveData** - Removes vertices from the region.

*Obj*->[IDLanROI::]RemoveData[, COUNT=*vertices*]  
 [, START=*index*] [, XRANGE=*variable*]  
 [, YRANGE=*variable*] [, ZRANGE=*variable*]

**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]]

*Obj*->[IDLanROI::]SetProperty[, *PROPERTY*=*value*]

**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.

**Properties:** [, ALL{Get}=variable]  
[, ROIGROUP\_XRANGE{Get}=variable]  
[, ROIGROUP\_YRANGE{Get}=variable]  
[, ROIGROUP\_ZRANGE{Get}=variable]

**IDLanROIGroup::Add** - Adds a region to the region group.

*Obj*->[IDLanROIGroup::]Add, ROI

**IDLanROIGroup::Cleanup** - Performs all cleanup for the object.

OBJ\_DESTROY, *Obj*  
or *Obj*->[IDLanROIGroup::]Cleanup

**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 }]  
[, RUN\_LENGTH=value] )

**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  
[, PROPERTY=variable]

**IDLanROIGroup::Init** - Initializes a region of interest group object.

*Obj* = OBJ\_NEW('IDLanROIGroup') or  
*Result* = *Obj*->[IDLanROIGroup::]Init( )

**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]]

**IDLcomActiveX** - Creates an IDL object that encapsulates an ActiveX control.

**IDLcomIDispatch** - Creates a COM object that implements an IDispatch interface. A dynamic sub-class of IDLcomIDispatch is created when the object is instantiated.

**IDLcomIDispatch::GetProperty** - Get properties for an IDispatch interface.

IDLcomIDispatch->GetProperty, PROPERTY=variable,  
[arg0, arg1, ...]

**IDLcomIDispatch::Init** - Initialize a COM object and establish a link between the resulting IDL object and the IDispatch interface

*Obj* = OBJ\_NEW('IDLcomIDispatch\$IDTYPE\$ID')

**IDLcomIDispatch::SetProperty** - Set properties for an IDispatch interface.

IDLcomIDispatch->SetProperty, PROPERTY=value

**IDLffDICOM** - Contains the data for one or more images embedded in a DICOM part 10 file. No superclasses. No subclasses.

**Property:** [, /VERBOSE{Init}]

**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]
[, PROPERTY=value] ) or
Result = Obj->[IDLffDICOM::]Init( [, Filename]
[, PROPERTY=value] )
```

**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
```

**IDLffDicomEx Object** - See [Appendix B, "IDL DICOM Quick Reference"](#) in the *Medical Imaging in IDL* manual.

**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] )
or
Result = Obj->[IDLffDXF::]Init( [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 )
```

**IDLffJPEG2000** - Object class used for reading and writing JPEG2000 files. No superclasses. No subclasses

**Properties:** [, BIT\_DEPTH {Get, Init, Set}=integer vector] [, BIT\_RATE {Init}=array] [, COLOR\_SPACE {Get}=variable] [, COMMENT {Get, Init, Set}=string vector] [, DIMENSIONS {Get, Init, Set}=vector] [, DISPLAY\_RESOLUTION {Get, Init, Set}=vector] [, FILENAME {Get}=string] [, JP2 {Get}=0 | 1] [, N\_COMPONENTS {Get, Init, Set}=integer] [, N\_LAYERS {Get, Init, Set}=integer] [, N\_LEVELS {Get, Init, Set}=integer] [, N\_TILES {Get}=variable] [, OFFSET {Get, Init, Set}=integer] [, PALETTE {Get, Init, Set}=array] [, PROGRESSION {Get, Init, Set}=string] [, QUIET {Init, Set}=0 | 1] [, READ {Get, Init}=0 | 1] [, REVERSIBLE {Get, Init, Set}=0 | 1] [, SIGNED {Get, Init, Set}=0 | 1] [, SUBSAMPLING {Get}=array] [, TILE\_DIMENSIONS {Get, Init, Set}=vector] [, TILE\_OFFSET {Get, Init, Set}=vector] [, TILE\_RANGE {Get, Init, Set}=vector] [, UUIDS {Get}=array] [, WRITE {Get, Init, Set}=0 | 1] [, XML {Get, Init, Set}=string vector] [, YCC {Get, Init, Set}=0 | 1]

**IDLffJPEG2000::GetData** - Returns image data from the IDLffJPEG2000 object.

```
Result = Obj->[IDLffJPEG2000::]GetData(
[COMPONENT=value] [, DISCARD_LEVELS=value]
[, MAX_LAYERS=value] [, N_COMPONENTS=value]
[, ORDER=value] [, REGION=value] [, /RGB]
[, TILE_INDEX=value])
```

**IDLffJPEG2000::GetProperty** - Retrieves the value of a property or group of properties for the IDLffJPEG2000 object.

```
Obj->[IDLffJPEG2000::]GetProperty
[, PROPERTY=variable]
```



**IDLffJPEG2000::GetTileProperty** - Retrieves the properties of a tile in an IDLffJPEG2000 object.

```
Obj->[IDLffJPEG2000:]GetTileProperty [, TileIndex
[, TileComponent]] [, N_LAYERS=variable]
[, N_LEVELS=variable] [, PROGRESSION=variable]
[, REVERSIBLE=variable]
[, TILE_DIMENSIONS=variable]
[, TILE_OFFSET=variable] [, YCC=variable]
```

**IDLffJPEG2000::GetUUID** - Allows you to get the data field from the specified UUID box.

```
Data = Obj-> [IDLffJPEG2000:]GetUUID(UUID,
LENGTH=length)
```

**IDLffJPEG2000::Init** - Initializes an IDLffJPEG2000 object.

```
Obj = OBJ_NEW('IDLffJPEG2000', Filename
[, PERSISTENT=0 | 1] [, PROPERTY=value])
```

**IDLffJPEG2000::SetData** - Writes data to the IDLffJPEG2000 object.

```
Obj->[IDLffJPEG2000:]SetData ([P1, ..., Pn]
[, COMPONENT=value] [, /ORDER]
[, TILE_INDEX=value])
```

**IDLffJPEG2000::SetProperty** - Sets the properties of an object that are open for writing.

```
Obj->[IDLffJPEG2000:]SetProperty
[, PROPERTY=value]
```

**IDLffJPEG2000::SetUUID** - Allows you to add UUID boxes when creating a new JPEG2000 file.

```
Obj->[IDLffJPEG2000:]SetUUID, uuid, Data
```

**IDLffLangCat** - Object class is used to find and load an XML language catalog. No superclasses. No subclasses

**Properties:** [, APP\_NAME {Get, Init}=string]  
[, APP\_PATH {Get, Init}=string]  
[, AVAILABLE\_LANGUAGES {Get}=string]  
[, DEFAULT\_KEYS {Get}=string]  
[, DEFAULT\_LANGUAGE {Get, Init}=string]  
[, DEFAULT\_N\_KEYS {Get}=variable] [, FILENAME  
{Get, Init}=string] [, KEYS  
{Get}=variable] [, LANGUAGE {Get, Init, Set}=string]  
[, N\_KEYS {Get}=variable] [, VERBOSE {Get, Init,  
Set}=byte]

**IDLffLangCat::AppendCatalog** - Adds keys from a file or files to those used to build the language catalog.

```
Result = Obj->[IDLffLangCat:]AppendCatalog
([, APP_NAME=applicationName]
[, FILENAME=filename]
[, APP_PATH=applicationPath])
```

**IDLffLangCat::Cleanup** - Performs all cleanup operations on the language catalog object.

```
OBJ_DESTROY, Obj
or
```

```
Obj->[IDLffLangCat:]Cleanup
```

**IDLffLangCat::GetProperty** - Retrieves a property or group of properties for an IDLffLangCat object.

```
Obj->[IDLffLangCat:]GetProperty
[, PROPERTY=variable]
```

**IDLffLangCat::Init** - Initializes an IDLffLangCat object.

```
Result = OBJ_NEW('IDLffLangCat', Language
[, PROPERTY=value] [, /CONTINUE_ON_ERROR])
```

**IDLffLangCat::Query** - Returns the string or string array that corresponds to all supplied key values.

```
Result = Obj->[IDLffLangCat:]Query( Key
[, DEFAULT_STRING=string])
```

**IDLffLangCat::SetProperty** - Sets the value of a property or group of properties for an IDLffLangCat object.

```
Obj->[IDLffLangCat:]SetProperty
[, PROPERTY=variable]
```

**IDLffMJPEG2000** - Object class used for reading and writing Motion JPEG2000 animations. No superclasses. No subclasses.

**Properties:** [, BIT\_DEPTH {Get, Init, Set}=integer]  
[, BIT\_RATE {Init, Set}=integer vector]  
[, COLOR\_SPACE {Get, Init, Set}=string]  
[, COMMENT {Get, Init, Set}=string]  
[, CURRENT\_FRAME {Get}=integer] [, DIMENSIONS  
{Get, Init, Set}=[width, height]] [, DURATION  
{Get}=value] [, FILENAME {Get}=string]  
[, FRAME\_BUFFER\_LENGTH {Get, Init, Set}=integer]  
[, FRAMES\_IN\_BUFFER {Get}=integer]  
[, FRAMES\_PERIOD {Get, Init, Set}=integer]  
[, N\_COMPONENTS {Get, Init, Set}=integer]  
[, N\_FRAMES {Get}=variable] [, N\_LAYERS {Get,  
Init, Set}=integer] [, N\_LEVELS {Get, Init,  
Set}=integer] [, N\_TILES {Get}=variable] [, PALETTE  
{Get, Init, Set}=array] [, PROGRESSION {Get, Init,  
Set}=string] [, REVERSIBLE {Get, Init, Set}=0 | 1]  
[, SCAN\_MODE {Get,}=0 | 1 2] [, SIGNED {Get, Init,  
Set}=0 | 1] [, STATE{Get}=value]  
[, TILE\_DIMENSIONS {Get, Init, Set}=vector]  
[, TIMESCALE{Get, Init, Set}=integer]  
[, WRITE {Get, Init,}=0 | 1] [, XML {Get, Init,}=string]  
[, YCC {Get, Init, Set}=0 | 1]

**IDLffMJPEG2000::Cleanup** - Deletes all IDLffMJPEG2000 objects.

```
OBJ_DESTROY, Obj
or
Obj->[IDLffMJPEG2000:]Cleanup
```

**IDLffMJPEG2000::Commit** - Shuts down the processing thread and closes the MJ2 file.

```
Result = Obj->[IDLffMJPEG2000:]Commit(Wait)
```

**IDLffMJPEG2000::GetData** - Returns frame, component, tile or region data from the MJ2 file.

Result = Obj->  
 >[IDLffMJPEG2000::]GetData(*FrameNumber*  
 [, COMPONENT=*value*] [, DISCARD\_LEVELS=*value*]  
 [, MAX\_LAYERS=*value*] [, N\_COMPONENTS=*value*]  
 [, /ORDER] [, REGION=*value*] [, /RGB]  
 [, TILE\_INDEX=*value*])

**IDLffMJPEG2000::GetProperty** - Retrieves the value of a property or group of properties for the IDLffMJPEG2000 object.

Obj->[IDLffMJPEG2000::]GetProperty  
 [, PROPERTY=*variable*]

**IDLffMJPEG2000::GetSequentialData** - Retrieves returns the frame index number and retrieves the associated frame data from the frame buffer.

Result = Obj->[IDLffMJPEG2000::]GetSequentialData  
 (*Data* [, FRAME\_NUMBER=*variable*]  
 [, FRAME\_PERIOD=*value*] [, STEP=*value*])

**IDLffMJPEG2000::Init** - Initializes an IDLffMJPEG2000 object.

Obj = OBJ\_NEW('IDLffMJPEG2000', *Filename*  
 [, PERSISTENT=0 | 1] [, PROPERTY=*value*])  
 or  
 Result = Obj->[IDLffMJPEG2000::]Init(*Filename*  
 [, PERSISTENT=0 | 1] [, PROPERTY=*value*]) (In a  
 lifecycle method only.)

**IDLffMJPEG2000::ReleaseSequentialData** - Releases a frame from the frame buffer.

Obj-> [IDLffMJPEG2000::]ReleaseSequentialData  
 (*Frame\_Index*)

**IDLffMJPEG2000::SetData** - Writes data to the IDLffMJPEG2000 object.

Obj->[IDLffMJPEG2000::]SetData (*A1*, ..., *An*)  
 [, COMPONENT=*value*] [, FRAME\_PERIOD=*value*]  
 [, /ORDER] [, TILE\_INDEX=*value*]

**IDLffMJPEG2000::SetProperty** - Sets object properties for a file that is open for writing.

Obj->[IDLffMJPEG2000::]SetProperty  
 [, PROPERTY=*value*]

**IDLffMJPEG2000::StartSequentialReading** - Initiates a sequential playback of a large number of frames.

Result = Obj->  
 [IDLffMJPEG2000::]StartSequentialReading  
 ([, COMPONENT=*value*] [, DISCARD\_LEVELS=*value*]  
 [, MAX\_LAYERS=*value*] [, N\_COMPONENTS=*value*]  
 [, /ORDER] [, REGION=*vector*] [, /RGB]  
 [, START\_FRAME\_NUMBER=*value*]  
 [, STOP\_FRAME\_NUMBER=*value*]  
 [, STOP\_FRAME\_NUMBER=*value*]  
 [, TILE\_INDEX=*value*])

**IDLffMJPEG2000::StopSequentialReading** - Releases the frame buffer memory and shuts down the background processing thread.

Result =  
 Obj->[IDLffMJPEG2000::]StopSequentialReading ( )

**IDLffMrSID** - Object class used to query information about and load image data from a MrSID (.sid) image file. No superclasses. No subclasses.

**Properties:** [, CHANNELS{Get}=*nChannels*]  
 [, DIMENSIONS{Gets}=*Dims*]  
 [, GEO\_VALID{Get}=*geoValid*]  
 [, GEO\_PROJTYPE{Get}=*geoProjType*]  
 [, GEO\_ORIGIN{Get}=*geoOrigin*]  
 [, GEO\_RESOLUTION{Get}=*geoRes*]  
 [, LEVELS{Get}=*Levels*]  
 [, PIXEL\_TYPE{Get}=*pixelType*] [, /QUIET{Init}]  
 [, TYPE{Get}=*strType*]

**IDLffMrSID::Cleanup** - Deletes all MrSID objects, closing the MrSID file in the process.

OBJ\_DESTROY, *Obj*  
 or  
 Obj->[IDLffMrSID::]Cleanup

**IDLffMrSID::GetDimsAtLevel** - Retrieves the dimensions of the image at a given level.

*Dims* = Obj->[IDLffMrSID::]GetDimsAtLevel ( *Level* )

**IDLffMrSID::GetImageData** - Returns the image data from the MrSID file.

*ImageData* = Obj->[IDLffMrSID::]GetImageData ( [, LEVEL = *lv*] [, SUB\_RECT = *rect*] )

**IDLffMrSID::GetProperty** - Query properties associated with the MrSID image.

Obj->[IDLffMrSID::]GetProperty[, PROPERTY=*variable*]

**IDLffMrSID::Init** - Initializes an IDLffMrSID object containing the image data from a MrSID image file.

Result = OBJ\_NEW('IDLffMrSID', *Filename*  
 [, PROPERTY=*value*])

**IDLffShape** - Contains geometry, connectivity and attributes for graphics primitives accessed from ESRI Shapefiles. No superclass. No subclasses.

**Properties:** [, /DBF\_ONLY{Init}]  
 [, ENTITY\_TYPE{Init}= *Value*] [, /UPDATE{Init}]

**IDLffShape::AddAttribute** - Adds an attribute to a shapefile.

Obj->[IDLffShape::]AddAttribute, *Name*, *Type*, *Width*  
 [, PRECISION=*integer*]

**IDLffShape::Cleanup** - Performs all cleanup on a Shapefile object.

OBJ\_DESTROY, *Obj* or Obj->[IDLffShape::]Cleanup

**IDLffShape::Close** - Closes a Shapefile.

Obj->[IDLffShape::]Close

**IDLffShape::DestroyEntity** - Frees memory associated with the entity structure.

Obj->[IDLffShape::]DestroyEntity, *Entity*

**IDLffShape::GetAttributes** - Retrieves the attributes for the entities you specify from a Shapefile.

*Result = Obj->[IDLffShape::]GetAttributes([Index] [, /ALL] [, /ATTRIBUTE\_STRUCTURE])*

**IDLffShape::GetEntity** - Returns an array of entity structures from a Shapefile.

*Result = Obj->[IDLffShape::]GetEntity([Index] [, /ALL] [, /ATTRIBUTES])*

**IDLffShape::GetProperty** - Returns the values of properties associated with a Shapefile object.

*Obj->[IDLffShape::]GetProperty [ , ATTRIBUTE\_INFO=variable] [ , ATTRIBUTE\_NAMES=variable] [ , ENTITY\_TYPE=variable] [ , FILENAME=variable] [ , IS\_OPEN=variable] [ , N\_ATTRIBUTES=variable] [ , N\_ENTITIES=variable] [ , N\_RECORDS=variable]*

**IDLffShape::Init** - Initializes or constructs a Shapefile object.

*Result = OBJ\_NEW('IDLffShape' [, Filename] [ , PROPERTY=value])*

**IDLffShape::Open** - Opens a specified Shapefile.

*Result = Obj->[IDLffShape::]Open('Filename' [ , /DBF\_ONLY] [ , /UPDATE] [ , ENTITY\_TYPE='value'])*

**IDLffShape::PutEntity** - Inserts an entity into the Shapefile object.

*Obj->[IDLffShape::]PutEntity, Data*

**IDLffShape::SetAttributes** - Modifies the attributes for a specified entity in a Shapefile object.

*Obj->[IDLffShape::]SetAttributes, Index, Attribute\_Num, Value*

or

*Obj->[IDLffShape::]SetAttributes, Index, Attributes*

**IDLffXMLDOM Classes** - Represents classes that provide support for IDL's XML Document Object Model (DOM).

**IDLffXMLDOMAttr** - Represents an attribute that is a part of an element object in an XML document. Subclass of IDLffXMLDOMNode.

**IDLffXMLDOMAttr::Cleanup** - Destroys both the accessing object in the IDL tree and any objects created by that object; does not modify the actual DOM tree. Should not be subclassed.

*OBJ\_DESTROY, Obj*

**IDLffXMLDOMAttr::GetName** - Returns the attribute node's name.

*Result = Obj->[IDLffXMLDOMAttr::]GetName()*

**IDLffXMLDOMAttr::GetSpecified** - Returns a scalar integer indicating how the attribute node's value was set.

*Result = Obj->[IDLffXMLDOMAttr::]GetSpecified()*

**IDLffXMLDOMAttr::GetValue** - Returns the attribute node's value.

*Result = Obj->[IDLffXMLDOMAttr::]GetValue()*

**IDLffXMLDOMAttr::SetValue** - Sets the attribute node's value.

*Obj->[IDLffXMLDOMAttr::]SetValue, Value*

**IDLffXMLDOMCDATASection** - Used to escape blocks of text in an XML document containing text that would otherwise be regarded as markup. Subclass of IDLffXMLDOMNode., IDLffXMLDOMCharacterData, and IDLffXMLDOMText.

**IDLffXMLDOMCDATASection::Cleanup** - Destroys both the accessing object in the IDL tree and any objects created by that object; does not modify the actual DOM tree. Should not be subclassed.

*OBJ\_DESTROY, Obj*

**IDLffXMLDOMCharacterData** - Extends the IDLffXMLDOM class with methods for accessing character data in the DOM tree. Subclass of IDLffXMLDOMNode.

**IDLffXMLDOMCharacterData::AppendData** - Appends a string to the node's character data.

*Obj->[IDLffXMLDOMCharacterData::]AppendData, String*

**IDLffXMLDOMCharacterData::Cleanup** - Destroys both the accessing object in the IDL tree and any objects created by that object; does not modify the actual DOM tree. Should not be subclassed.

*OBJ\_DESTROY, Obj*

**IDLffXMLDOMCharacterData::DeleteData** - Deletes a number of characters from the node's character data, starting at an offset.

*Obj->[IDLffXMLDOMCharacterData::]DeleteData, Offset, Count*

**IDLffXMLDOMCharacterData::GetData** - Returns the node's character data.

*Result = Obj-> [IDLffXMLDOMCharacterData::]GetData()*

**IDLffXMLDOMCharacterData::GetLength** - Returns the number of characters in the node.

*Result = Obj->[IDLffXMLDOMCharacterData::]GetLength()*

**IDLffXMLDOMCharacterData::InsertData** - Inserts a string in the node's character data, starting at an offset.

*Obj->[IDLffXMLDOMCharacterData::]InsertData, Offset, String*

**IDLffXMLDOMCharacterData::ReplaceData** - Replaces a number of characters, starting at an offset in the node's character data, with a string.

*Obj->[IDLffXMLDOMCharacterData::]ReplaceData, Offset, Count, String*

**IDLffXMLDOMCharacterData::SetData** - Sets the node's character data to a string

*Obj->[IDLffXMLDOMCharacterData::]SetData, String*

**IDLffXMLDOMCharacterData::SubstringData** - Returns a string composed of a substring of the node's character data.

*Result = Obj->[IDLffXMLDOMCharacterData::] SubstringData (Offset, Count)*

**IDLffXMLDOMComment** - Represents the content of a comment (characters between "<!--" and "-->!") in an XML document. Subclass of IDLffXMLDOMCharacterData and IDLffXMLDOMText.

**IDLffXMLDOMComment::Cleanup** - Destroys both the accessing object in the IDL tree and any objects created by that object; does not modify the actual DOM tree. Should not be subclassed.

OBJ\_DESTROY, *Obj*

**IDLffXMLDOMDocument** - Represents the entire XML document as the document tree's root and by providing primary access to the document's data. Subclass of IDLffXMLDOMNode.

**Properties:** [, NODE\_DESTRUCTION\_POLICY {Get, Init, Set}=0 | 1]

Destroys both the accessing object in the IDL tree and any objects created by that object; does not modify the actual DOM tree.

OBJ\_DESTROY, *Obj*

or

*Obj*->[IDLffXMLDOMDocument::]Cleanup

**IDLffXMLDOMDocument::CreateAttribute** - Creates and names an attribute node owned by the XML document.

*Result = Obj*->[IDLffXMLDOMDocument::]  
CreateAttribute(*Name*)

**IDLffXMLDOMDocument::CreateCDATASection** - Creates and fills a CDATASection node owned by the XML document.

*Result = Obj*->[IDLffXMLDOMDocument::]  
CreateCDATASection(*String*)

**IDLffXMLDOMDocument::CreateComment** - Creates and fills a comment node owned by the XML document.

*Result = Obj*->[IDLffXMLDOMDocument::]  
CreateComment, (*String*)

**IDLffXMLDOMDocument::CreateDocumentFragment** -

Creates a document fragment node owned by the XML document.

*Result = Obj*->[IDLffXMLDOMDocument::]  
CreateDocumentFragment()

**IDLffXMLDOMDocument::CreateElement** - Creates and names with a tag name an element node owned by the XML document.

*Result = Obj*->[IDLffXMLDOMDocument::]  
CreateElement(*TagName*)

**IDLffXMLDOMDocument::CreateEntityReference** - Creates and names an entity reference node owned by the XML document.

*Result = Obj*->[IDLffXMLDOMDocument::]  
CreateEntityReference(*Name*)

**IDLffXMLDOMDocument::CreateNodeIterator** - Creates and names an instance of an IDLffXMLDOMNodeIterator object.

*Result = Obj*->[IDLffXMLDOMDocument::]  
CreateNodeIterator(*Name* [, FILTER\_NAME=*string*]  
[, FILTER\_USERDATA=*variable*]  
[, WHAT\_TO\_SHOW=*value*])

**IDLffXMLDOMDocument::CreateProcessingInstruction** -

Creates and stores two strings in a ProcessingInstruction node owned by the XML document.

*Result = Obj*->[IDLffXMLDOMDocument::]  
CreateProcessingInstruction(*Target*, *Data*)

**IDLffXMLDOMDocument::CreateTextNode** - Creates and fills a text node owned by the XML document.

*Result = Obj*->[IDLffXMLDOMDocument::]  
CreateTextNode(*String*)

**IDLffXMLDOMDocument::CreateTreeWalker** - Creates an instance of an IDLffXMLDOMTreeWalker object.

*Result = Obj*->[IDLffXMLDOMDocument::]  
CreateTreeWalker(*RootNode* [, FILTER\_NAME=*string*]  
[, FILTER\_USERDATA=*variable*]  
[, WHAT\_TO\_SHOW=*value*])

**IDLffXMLDOMDocument::GetDoctype** - Creates an instance of IDLffXMLDOMDocumentType.

*Result = Obj*->[IDLffXMLDOMDocument::]  
GetDoctype()

**IDLffXMLDOMDocument::GetDocumentElement** - Creates an instance of IDLffXMLDOMElement.

*Result = Obj*->[IDLffXMLDOMDocument::]  
GetDocumentElement()

**IDLffXMLDOMDocument::GetElementsByTagName** -

Creates an IDLffXMLDOMNodeList object containing all element nodes in the XML document with the specified tag name.

*Result = Obj*->[IDLffXMLDOMDocument::]  
GetElementsByTagName(*TagName*)

**IDLffXMLDOMDocument::Init** - Initializes the object.

*Obj = OBJ\_NEW*('IDLffXMLDOMDocument'  
[, /EXCLUDE\_IGNOREABLE\_WHITESPACE]  
[, /EXPAND\_ENTITY\_REFERENCES]  
[, FILENAME=*string* / STRING=*string*] [,  
MSG\_ERROR=*string*] [, MSG\_FATAL=*string*] [,  
MSG\_WARNING=*string*] [, /QUIET]  
[, SCHEMA\_CHECKING=*value*]  
[, VALIDATION\_MODE=*value*])

or

*Result = Obj*->[IDLffXMLDOMDocument::]Init(  
[, /EXCLUDE\_IGNOREABLE\_WHITESPACE]  
[, /EXPAND\_ENTITY\_REFERENCES]  
[, FILENAME=*string* / STRING=*string*] [,  
MSG\_ERROR=*string*] [, MSG\_FATAL=*string*] [,  
MSG\_WARNING=*string*] [, /QUIET]  
[, SCHEMA\_CHECKING=*value*]  
[, VALIDATION\_MODE=*value*])

(Only in a subclass's Init method)

**IDLffXMLDOMDocument::Load** - Loads and parses XML data from a specified source; creates a DOM document tree (accessed through this object) from the data.

*Obj*->[IDLffXMLDOMDocument::]Load  
[, /EXCLUDE\_IGNOREABLE\_WHITESPACE]  
[, /EXPAND\_ENTITY\_REFERENCES]  
[, FILENAME=*string* / STRING=*string*][,  
MSG\_ERROR=*string*] [, MSG\_FATAL=*string*] [,  
MSG\_WARNING=*string*] [, /QUIET]  
[, SCHEMA\_CHECKING=*value*]  
[, VALIDATION\_MODE=*value*]

**IDLffXMLDOMDocument::Save** - Serializes the current DOM document and writes it to an output source.

```
Obj->[IDLffXMLDOMDocument::]Save
[, ENCODING=string]
[, /EXPAND_ENTITY_REFERENCES]
[, FILENAME=string / STRING=variable][,
/PRETTY_PRINT]
```

**IDLffXMLDOMDocumentFragment** - References a document fragment node. Subclass of IDLffXMLDOMNode.

**IDLffXMLDOMDocumentFragment::Cleanup** - Destroys both the accessing object in the IDL tree and any objects created by that object; does not modify the actual DOM tree. Should not be subclassed.

```
OBJ_DESTROY, Obj
```

**IDLffXMLDOMDocumentType** - References a DocumentType node. Subclass of IDLffXMLDOMNode.

**IDLffXMLDOMDocumentType::Cleanup** - Destroys both the accessing object in the IDL tree and any objects created by that object; does not modify the actual DOM tree. Should not be subclassed.

```
OBJ_DESTROY, Obj
```

**IDLffXMLDOMDocumentType::GetEntities** - Returns the external and internal entities declared in the DTD.

```
Result = Obj->[IDLffXMLDOMDocumentType::]
GetEntities()
```

**IDLffXMLDOMDocumentType::GetName** - Returns the DTD's name.

```
Result = Obj->[IDLffXMLDOMDocumentType::]
GetName()
```

**IDLffXMLDOMDocumentType::GetNotations** - Returns the notations declared in the DTD.

```
Result = Obj->[IDLffXMLDOMDocumentType::]
GetNotations()
```

**IDLffXMLDOMELEMENT** - References an element node. Subclass of IDLffXMLDOMNode.

**IDLffXMLDOMELEMENT::Cleanup** - Destroys both the accessing object in the IDL tree and any objects created by that object; does not modify the actual DOM tree. Should not be subclassed.

```
OBJ_DESTROY, Obj
```

**IDLffXMLDOMELEMENT::GetAttribute** - Returns the value of the named attribute.

```
Result = Obj->[IDLffXMLDOMELEMENT::]GetAttribute
(Name)
```

**IDLffXMLDOMELEMENT::GetAttributeNode** - Creates an named IDLffXMLDOMAttr object.

```
Result = Obj->[IDLffXMLDOMELEMENT::]
GetAttributeNode(Name)
```

**IDLffXMLDOMELEMENT::GetElementsByTagName** - Creates an IDLffXMLDOMNodeList object containing all element nodes in the XML document with the specified tag name.

```
Result = Obj->[IDLffXMLDOMELEMENT::]
GetElementsByTagName(TagName)
```

**IDLffXMLDOMELEMENT::GetTagName** - Returns the element's name.

```
Result = Obj->[IDLffXMLDOMELEMENT::]GetTagName()
```

**IDLffXMLDOMELEMENT::RemoveAttribute** - Removes the named attribute from the element node.

```
Result = Obj->[IDLffXMLDOMELEMENT::]
RemoveAttribute(Name)
```

**IDLffXMLDOMELEMENT::RemoveAttributeNode** -

Removes the named attribute node from the element node.

```
Result = Obj->[IDLffXMLDOMELEMENT::]
RemoveAttributeNode(OldAttr)
```

**IDLffXMLDOMELEMENT::SetAttribute** - Adds and sets a new attribute to the element node.

```
Result = Obj->[IDLffXMLDOMELEMENT::]
SetAttribute(Name, Value)
```

**IDLffXMLDOMELEMENT::SetAttributeNode** - Adds a new attribute to the element node.

```
Result = Obj->[IDLffXMLDOMELEMENT::]
SetAttributeNode(NewAttr)
```

**IDLffXMLDOMENTITY** - References a parsed or unparsed entity in an XML document. Subclass of IDLffXMLDOMNode.

**IDLffXMLDOMENTITY::Cleanup** - Destroys both the accessing object in the IDL tree and any objects created by that object; does not modify the actual DOM tree. Should not be subclassed.

```
OBJ_DESTROY, Obj
```

**IDLffXMLDOMENTITY::GetNotationName** - Returns the notation's name for the entity.

```
Result = Obj->[IDLffXMLDOMENTITY::]
GetNotationName()
```

**IDLffXMLDOMENTITY::GetPublicId** - Returns the entity's public ID if specified.

```
Result = Obj->[IDLffXMLDOMENTITY::]GetPublicId()
```

**IDLffXMLDOMENTITY::GetSystemId** - Returns the entity's system ID if specified.

```
Result = Obj->[IDLffXMLDOMENTITY::]GetSystemId()
```

**IDLffXMLDOMENTITYREFERENCE** - References an entity reference node in an XML document. Subclass of IDLffXMLDOMNode.

**IDLffXMLDOMENTITYREFERENCE::Cleanup** - Destroys both the accessing object in the IDL tree and any objects created by that object; does not modify the actual DOM tree. Should not be subclassed.

```
OBJ_DESTROY, Obj
```

**IDLffXMLDOMNamedNodeMap** - Container for nodes that uses node names for access. Subclass of IDLffXMLDOMNode.

**IDLffXMLDOMNamedNodeMap::Cleanup** - Destroys both the accessing object in the IDL tree and any objects created by that object; does not modify the actual DOM tree. Should not be subclassed.

```
OBJ_DESTROY, Obj
```

**IDLffXMLDOMNamedNodeMap::GetLength** - Returns the number of nodes in the named node map.

```
Result = Obj->[IDLffXMLDOMNamedNodeMap::]
GetLength()
```

**IDLffXMLDOMNamedNodeMap::GetNamedItem** - Returns an object reference to the node that contains the named item.

*Result = Obj->[IDLffXMLDOMNamedNodeMap:]*  
GetNamedItem(*Name*)

**IDLffXMLDOMNamedNodeMap::Item** - Returns an object reference to the node that contains the indexed item.

*Result = Obj->[IDLffXMLDOMNamedNodeMap:]*  
Item(*Index*)

**IDLffXMLDOMNamedNodeMap::RemoveNamedItem** - Removes the specified node from the named node map.

*Result = Obj->[IDLffXMLDOMNamedNodeMap:]*  
RemoveNamedItem(*Name*)

**IDLffXMLDOMNamedNodeMap::SetNamedItem** - Adds and names a node to the named node map, replacing an existing node with the specified name if necessary.

*Result = Obj->[IDLffXMLDOMNamedNodeMap:]*  
SetNamedItem(*Name*)

**IDLffXMLDOMNode** - Abstract superclass for other IDLffXMLDOM node classes. No superclasses.

**IDLffXMLDOMNode::AppendChild** - Adds a new node at the end of the calling node's children list.

*Result = Obj->[IDLffXMLDOMNode:]*  
AppendChild(*NewChild*)

**IDLffXMLDOMNode::Cleanup** - Destroys both the accessing object in the IDL tree and any objects created by that object; does not modify the actual DOM tree. Should not be subclassed.

OBJ\_DESTROY, *Obj*

**IDLffXMLDOMNode::CloneNode** - Creates a copy of the calling node; the cloned node has no parent.

*Result = Obj->[IDLffXMLDOMNode:]*  
CloneNode([, DEEP=*value*])

**IDLffXMLDOMNode::GetAttributes** - Creates a named node map used to access an element object's attributes.

*Result = Obj->[IDLffXMLDOMNode:]*GetAttributes()

**IDLffXMLDOMNode::GetChildNodes** - Creates a node list used to access the calling node's children.

*Result = Obj->[IDLffXMLDOMNode:]*GetChildNodes()

**IDLffXMLDOMNode::GetFirstChild** - Creates an IDLffXMLDOM node that refers to the first child in the DOM tree.

*Result = Obj->[IDLffXMLDOMNode:]*GetFirstChild()

**IDLffXMLDOMNode::GetLastChild** - Creates an IDLffXMLDOM node that refers to the last child in the DOM tree.

*Result = Obj->[IDLffXMLDOMNode:]*GetLastChild()

**IDLffXMLDOMNode::GetNextSibling** - Creates an IDLffXMLDOM node that refers to the next sibling in the DOM tree.

*Result = Obj->[IDLffXMLDOMNode:]*GetNextSibling()

**IDLffXMLDOMNode::GetNodeName** - Returns the node's name, depending on the subclass type.

*Result = Obj->[IDLffXMLDOMNode:]*GetNodeName()

**IDLffXMLDOMNode::GetNodeType** - Returns the node's type.

*Result = Obj->[IDLffXMLDOMNode:]*GetNodeType()

**IDLffXMLDOMNode::GetNodeValue** - Returns the node's value, depending on the subclass type.

*Result = Obj->[IDLffXMLDOMNode:]*GetNodeValue()

**IDLffXMLDOMNode::GetOwnerDocument** - Returns an object reference to the IDLffXMLDOMDocument used to create the node.

*Result = Obj->[IDLffXMLDOMNode:]*  
GetOwnerDocument()

**IDLffXMLDOMNode::GetParentNode** - Creates an IDLffXMLDOM node that refers to the parent in the DOM tree.

*Result = Obj->[IDLffXMLDOMNode:]*GetParentNode()

**IDLffXMLDOMNode::GetPreviousSibling** - Creates an IDLffXMLDOM node that refers to the previous sibling in the DOM tree.

*Result = Obj->[IDLffXMLDOMNode:]*  
GetPreviousSibling()

**IDLffXMLDOMNode::HasChildNodes** - Returns a value indicating whether the calling node has children.

*Result = Obj->[IDLffXMLDOMNode:]*HasChildNodes()

**IDLffXMLDOMNode::InsertBefore** - Inserts a new node into the calling node's children before a reference node.

*Result = Obj->[IDLffXMLDOMNode:]*  
InsertBefore(*NewChild* [, *RefChild*])

**IDLffXMLDOMNode::RemoveChild** - Removes an existing node from the calling node's child list.

*Result = Obj->[IDLffXMLDOMNode:]*RemoveChild(*OldChild*)

**IDLffXMLDOMNode::ReplaceChild** - Replaces an existing child node with a new node in the calling node's child list.

*Result = Obj->[IDLffXMLDOMNode:]*  
ReplaceChild(*NewChild*, *OldChild*)

**IDLffXMLDOMNode::SetNodeValue** - Sets the node's value to the contents of a string.

*Result = Obj->[IDLffXMLDOMNode:]*  
SetNodeValue(*NodeValue*)

**IDLffXMLDOMNodeIterator** - Allows iterative navigation of an IDLffXMLDOM tree. No superclasses.

**IDLffXMLDOMNodeIterator::Cleanup** - Destroys both the node-iterator object and any objects created by that object; does not modify the actual DOM tree. Should not be subclassed.

OBJ\_DESTROY, *Obj*

**IDLffXMLDOMNodeIterator::NextNode** - Returns an object reference to the DOM tree node that appears after the iterator's current position in document-order traversal.

*Result =*  
*Obj->[IDLffXMLDOMNodeIterator:]*NextNode()

**IDLffXMLDOMNodeIterator::PreviousNode** - Returns an object reference to the DOM tree node that appears before the iterator's current position in document-order traversal.

*Result =*  
*Obj->[IDLffXMLDOMNodeIterator:]*PreviousNode()

**IDLffXMLDOMNodeList** - Container for nodes. No superclasses.

**IDLffXMLDOMNodeList::Cleanup** - Destroys both the accessing object in the IDL tree and any objects created by that object; does not modify the actual DOM tree. Should not be subclassed.  
OBJ\_DESTROY, *Obj*

**IDLffXMLDOMNodeList::GetLength** - Returns the number of nodes in the node list.

*Result = Obj->*[IDLffXMLDOMNodeList::]GetLength()

**IDLffXMLDOMNodeList::Item** - Returns an object reference to the node that contains the indexed item.

*Result = Obj->*[IDLffXMLDOMNodeList::]Item(*Index*)

**IDLffXMLDOMNotation** - References a notation in the DTD. Subclass of IDLffXMLDOMNode.

**IDLffXMLDOMNotation::Cleanup** - Destroys both the accessing object in the IDL tree and any objects created by that object; does not modify the actual DOM tree. Should not be subclassed.

OBJ\_DESTROY, *Obj*

**IDLffXMLDOMNotation::GetPublicID** - Returns the notation's public ID.

*Result = Obj->*[IDLffXMLDOMNotation::]GetPublicID()

**IDLffXMLDOMNotation::GetSystemID** - Returns the notation's system ID.

*Result = Obj->*[IDLffXMLDOMNotation::]GetSystemID()

**IDLffXMLDOMProcessingInstruction** - References a processing instruction node in the XML document. Subclass of IDLffXMLDOMNode.

**IDLffXMLDOMProcessingInstruction::Cleanup** - Destroys both the accessing object in the IDL tree and any objects created by that object; does not modify the actual DOM tree. Should not be subclassed.

OBJ\_DESTROY, *Obj*

**IDLffXMLDOMProcessingInstruction::GetData** - Returns the content of the processing instruction.

*Result = Obj->*[IDLffXMLDOMProcessingInstruction::]GetData()

**IDLffXMLDOMProcessingInstruction::GetTarget** -

Returns the target of the processing instruction.

*Result = Obj->*[IDLffXMLDOMProcessingInstruction::]GetTarget()

**IDLffXMLDOMProcessingInstruction::SetData** - Sets the content of the processing instruction.

*Obj->*[IDLffXMLDOMProcessingInstruction::]SetData, *Content*

**IDLffXMLDOMText** - References a text node in the XML document. Subclass of IDLffXMLDOMNode and IDLffXMLDOMCharacterData.

**IDLffXMLDOMText::Cleanup** - Destroys both the accessing object in the IDL tree and any objects created by that object; does not modify the actual DOM tree. Should not be subclassed.

OBJ\_DESTROY, *Obj*

**IDLffXMLDOMText::IsIgnorableWhitespace** - Indicates whether the text node contains ignorable whitespace.

*Result = Obj->*[IDLffXMLDOMText::]IsIgnorableWhitespace()

**IDLffXMLDOMText::SplitText** - Breaks a text node into two sibling nodes in the same tree, splitting the text at the specified offset.

*Result = Obj->*[IDLffXMLDOMText::]SplitText(*Offset*)

**IDLffXMLDOMTreeWalker** - Allows tree-walking navigation of an IDLffXMLDOM tree. No superclasses.

**IDLffXMLDOMTreeWalker::Cleanup** - Destroys both the node-iterator object and any objects created by that object; does not modify the actual DOM tree. Should not be subclassed.

OBJ\_DESTROY, *Obj*

**IDLffXMLDOMTreeWalker::FirstChild** - Returns an object reference to the DOM tree node that is the first child of the node to which the walker is currently pointing.

*Result = Obj->*[IDLffXMLDOMTreeWalker::]FirstChild()

**IDLffXMLDOMTreeWalker::GetCurrentNode** - Returns an object reference to the DOM tree node to which the walker is currently pointing.

*Result = Obj->*[IDLffXMLDOMTreeWalker::]GetCurrentNode()

**IDLffXMLDOMTreeWalker::LastChild** - Returns an object reference to the DOM tree node that is the last child of the node to which the walker is currently pointing.

*Result = Obj->*[IDLffXMLDOMTreeWalker::]LastChild()

**IDLffXMLDOMTreeWalker::NextNode** - Returns an object reference to the DOM tree node visited next after the walker's current node in document-order traversal.

*Result = Obj->*[IDLffXMLDOMTreeWalker::]NextNode()

**IDLffXMLDOMTreeWalker::NextSibling** - Returns an object reference to the DOM tree node that is the next sibling of the node to which the walker is currently pointing.

*Result = Obj->*[IDLffXMLDOMTreeWalker::]NextSibling()

**IDLffXMLDOMTreeWalker::ParentNode** - Returns an object reference to the DOM tree node that is the parent of the node to which the walker is currently pointing.

*Result = Obj->*[IDLffXMLDOMTreeWalker::]ParentNode()

**IDLffXMLDOMTreeWalker::PreviousNode** - Returns an object reference to the DOM tree node visited before the walker's current node in document-order traversal.

*Result = Obj->*[IDLffXMLDOMTreeWalker::]PreviousNode()

**IDLffXMLDOMTreeWalker::PreviousSibling** - Returns an object reference to the DOM tree node that is the previous sibling of the node to which the walker is currently pointing.

*Result = Obj->*[IDLffXMLDOMTreeWalker::]PreviousSibling()

**IDLffXMLDOMTreeWalker::SetCurrentNode** - Sets the walker's current node to the specified node.

*Obj->*[IDLffXMLDOMTreeWalker::]SetCurrentNode(*CurrentNode*)

**IDLffXMLSAX** - Represents an XML SAX Level 2 parser. No super-classes. In order to use this class, you *must* write your own subclass.

**Properties:** [, FILENAME{Get}=variable]  
 [, /NAMESPACE\_PREFIXES{Get, Init, Set}]  
 [, /PARSER\_LOCATION{Get}=variable]  
 [, /PARSER\_PUBLICID{Get}=variable]  
 [, /PARSER\_URI{Get}=variable]  
 [, /SCHEMA\_CHECKING{Get, Init, Set}=[0,1,2]]  
 [, /VALIDATION\_MODE{Get, Init, Set}=[0,1,2]]

**IDLffXMLSAX::AttributeDecl** - Called when the parser detects an <!ATTLIST ...> declaration in a DTD.

*Obj*->[IDLffXMLSAX::]AttributeDecl, *eName*, *aName*,  
*Type*, *Mode*, *Value*

**IDLffXMLSAX::Characters** - Called when the parser detects text in the parsed document.

*Obj*->[IDLffXMLSAX::]Characters, *Chars*

**IDLffXMLSAX::Cleanup** - Performs all cleanup on the object.

OBJ\_DESTROY, *Obj*

or

*Obj*->[IDLffXMLSAX::]Cleanup

**IDLffXMLSAX::Comment** - Called when the parser detects a comment section of the form <!-- ... -->.

*Obj*->[IDLffXMLSAX::]Comment, *Comment*

**IDLffXMLSAX::ElementDecl** - Called when the parser detects an <!ELEMENT ...> declaration in the DTD.

*Obj*->[IDLffXMLSAX::]ElementDecl, *Name*, *Model*

**IDLffXMLSAX::EndCDATA** - Called when the parser detects the end of a <![CDATA[...]]> text section.

*Obj*->[IDLffXMLSAX::]EndCDATA

**IDLffXMLSAX::EndDocument** - Called when the parser detects the end of the XML document.

*Obj*->[IDLffXMLSAX::]EndDocument

**IDLffXMLSAX::EndDTD** - Called when the parser detects the end of a Document Type Definition (DTD).

*Obj*->[IDLffXMLSAX::]EndDTD

**IDLffXMLSAX::EndElement** - Called when the parser detects the end of an element.

*Obj*->[IDLffXMLSAX::]EndElement, *URI*, *Local*, *qName*

**IDLffXMLSAX::EndEntity** - Called when the parser detects the end of an internal or external entity expansion.

*Obj*->[IDLffXMLSAX::]EndEntity, *Name*

**IDLffXMLSAX::EndPrefixMapping** - Called when a previously declared prefix mapping goes out of scope.

*Obj*->[IDLffXMLSAX::]EndPrefixMapping, *Prefix*

**IDLffXMLSAX::Error procedure** - Called when the parser detects an error that is not expected to be fatal.

*Obj*->[IDLffXMLSAX::]Error, *SystemID*, *LineNumber*,  
*ColumnNumber*, *Message*

**IDLffXMLSAX::ExternalEntityDecl** - Called when the parser detects an <!ENTITY ...> declarations in the DTD for a parsed external entity.

*Obj*->[IDLffXMLSAX::]ExternalEntityDecl, *Name*,  
*PublicID*, *SystemID*

**IDLffXMLSAX::FatalError** - Called when the parser detects a fatal error.

*Obj*->[IDLffXMLSAX::]FatalError, *SystemID*,  
*LineNumber*, *ColumnNumber*, *Message*

**IDLffXMLSAX::GetProperty** - Used to get the values of various properties of the parser.

*Obj*->[IDLffXMLSAX::]GetProperty  
 [, *PROPERTY*=variable]

**IDLffXMLSAX::IgnorableWhitespace** - Called when the parser detects whitespace that separates elements in an element content model.

*Obj*->[IDLffXMLSAX::]IgnorableWhitespace, *Chars*

**IDLffXMLSAX::InitI** - Initializes an XML parser object.

*Obj* = OBJ\_NEW('IDLffXMLSAX'  
 [, *PROPERTY*=value]) or *Result* = *Obj*-  
 >[IDLffXMLSAX::]InitI( [*PROPERTY*=value])

**IDLffXMLSAX::InternalEntityDecl** - Called when the parser detects an <!ENTITY ...> declaration in a DTD for (parsed) internal entities.

*Obj*->[IDLffXMLSAX::]InternalEntityDecl, *Name*, *Value*

**IDLffXMLSAX::NotationDecl** - Called when the parser detects a <!NOTATION ...> declaration in a DTD.

*Obj*->[IDLffXMLSAX::]NotationDecl, *Name*, *PublicID*,  
*SystemID*

**IDLffXMLSAX::ParseFile** - Parses the specified XML file.

*Obj*->[IDLffXMLSAX::]ParseFile, *Input* [, /URL]  
 [, /XML\_STRING]

**IDLffXMLSAX::ProcessingInstruction** - Called when the parser detects a processing instruction.

*Obj*->[IDLffXMLSAX::]ProcessingInstruction, *Target*,  
*Data*

**IDLffXMLSAX::SetProperty** - Used to set the values of various properties of the parser.

*Obj*->[IDLffXMLSAX::]SetProperty  
 [, *PROPERTY*=value]

**IDLffXMLSAX::SkippedEntity** - Called when the parser skips an entity and validation is not being performed.

*Obj*->[IDLffXMLSAX::]SkippedEntity, *Name*

**IDLffXMLSAX::StartCDATA** - Called when the parser detects the beginning of a <![CDATA[...]]> text section.

*Obj*->[IDLffXMLSAX::]StartCDATA

**IDLffXMLSAX::StartDocument** - Called when the parser begins processing a document, and before any data is processed.

*Obj*->[IDLffXMLSAX::]StartDocument



**IDLffXMLSAX::StartDTD** - Called when the parser detects the beginning of a Document Type Definition (DTD).

*Obj->*[IDLffXMLSAX::]StartDTD, *Name*, *PublicID*, *SystemID*

**IDLffXMLSAX::StartElement** - Called when the parser detects the beginning of an element.

*Obj->*[IDLffXMLSAX::]StartElement, *URI*, *Local*, *qName* [, *attName*, *attValue*]

**IDLffXMLSAX::StartEntity** - Called when the parser detects the start of an internal or external entity expansion.

*Obj->*[IDLffXMLSAX::]StartEntity, *Name*

**IDLffXMLSAX::StartPrefixmapping** - Called when the parser detects the beginning of a namespace declaration.

*Obj->*[IDLffXMLSAX::]StartPrefixmapping, *Prefix*, *URI*

**IDLffXMLSAX::StopParsing** - Used during a parse operation to halt the operation and cause the ParseFile method to return.

*Obj->*[IDLffXMLSAX::]StopParsing

**IDLffXMLSAX::UnparsedEntityDecl** - Called when the parser detects an `<!ENTITY ...>` declaration that includes the NDATA keyword, indicating that the entity is not meant to be parsed.

*Obj->*[IDLffXMLSAX::]UnparsedEntityDec, *Name*, *PublicID*, *SystemID*, *Notation*

**IDLffXMLSAX::Warning** - Called when the parser detects a problem during processing.

*Obj->*[IDLffXMLSAX::]Warning, *SystemID*, *LineNumber*, *ColumnNumber*, *Message*

**IDLgrAxis** - Represents a single vector that may include a set of tick marks, tick labels, and a title.

**Properties:** [, ALL{Get}=variable]  
 [, ALPHA\_CHANNEL{Get, Init, Set}=value]  
 [, AM\_PM{Get, Init, Set}=array]  
 [, CLIP\_PLANES{Get, Init, Set}=array] [, COLOR{Get, Init, Set}=index or RGB\_vector]  
 [, CRANGE{Get}=variable] [, DAYS\_OF\_WEEK{Get, Init, Set}=array] [, DEPTH\_TEST\_DISABLE{Get, Init, Set}={0 | 1 | 2}] [, DEPTH\_TEST\_FUNCTION{Get, Init, Set}={0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8}]  
 [, DEPTH\_WRITE\_DISABLE{Get, Init, Set}={0 | 1 | 2}]  
 [, DIRECTION{Get, Init, Set}=integer] [, /EXACT{Get, Init, Set}] [, /EXTEND{Get, Init, Set}]  
 [, GRIDSTYLE{Get, Init, Set}=integer{0 to 6} or [repeat{1 to 255}, bitmask]] [, /HIDE{Get, Init, Set}]  
 [, LOCATION{Get, Init, Set}=[x, y] or [x, y, z]]  
 [, /LOG{Get, Init, Set}] [, MAJOR{Get, Init, Set}=integer] [, MINOR{Get, Init, Set}=integer]  
 [, MONTHS{Get, Init, Set}=array] [, /NOTEXT{Get, Init, Set}] [, /PALETTE{Get, Init, Set}=objref]  
 [, PARENT{Get}=variable] [, RANGE{Get, Init, Set}=[min, max]] [, /REGISTER\_PROPERTIES{Get, Init, Set}] [, /SHADER{Get, Init, Set}=objref]  
 [, SUBTICKLEN{Get, Init, Set}=value]  
 [, TEXTALIGNMENTS{Get, Init, Set}=[horiz{0.0 to 1.0}, vert{0.0 to 1.0}]] [, TEXTBASELINE{Get, Init,

Set}=vector] [, TEXTPOS{Get, Init, Set}={0 | 1}]  
 [, TEXTUPDIR{Get, Init, Set}=vector] [, THICK{Get, Init, Set}=points{1.0 to 10.0}] [, TICKDIR{Get, Init, Set}={0 | 1}] [, TICKFORMAT{Get, Init, Set}=string or array of strings]

**IDLgrAxis Properties - continued**

[, TICKFRMTDATA{Get, Init, Set}=value]  
 [, TICKINTERVAL{Get, Init, Set}=value]  
 [, TICKLAYOUT{Get, Init, Set}=scalar]  
 [, TICKLEN{Get, Init, Set}=value] [, TICKTEXT{Get, Init, Set}=objref or vector] [, TICKUNITS{Get, Init, Set}=string or a vector of strings] [, TICKVALUES{Get, Init, Set}=vector] [, TITLE{Get, Init, Set}=objref]  
 [, /USE\_TEXT\_COLOR{Get, Init, Set}]  
 [, XCOORD\_CONV{Get, Init, Set}=vector]  
 [, XRANGE{Get}=variable] [, YCOORD\_CONV{Get, Init, Set}=vector] [, YRANGE{Get}=variable]  
 [, ZCOORD\_CONV{Get, Init, Set}=vector]  
 [, ZRANGE{Get}=variable]

**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 [, *PROPERTY=variable*]

**IDLgrAxis::Init** - Initializes an axis object.

*Obj =* OBJ\_NEW('IDLgrAxis' [, *Direction*]  
 [, *PROPERTY=value*]) or  
*Result = Obj->*[IDLgrAxis::]Init( [*Direction*]  
 [, *PROPERTY=value*])

**IDLgrAxis::SetProperty** - Sets the value of a property or group of properties for the axis.

*Obj->*[IDLgrAxis::]SetProperty [, *PROPERTY=value*]

**IDLgrBuffer** - An in-memory, off-screen destination object.

**Properties:** [, ALL{Get}=variable]  
 [, COLOR\_MODEL{Get, Init}={0 | 1}]  
 [, DIMENSIONS{Get, Init, Set}=[width, height]]  
 [, GRAPHICS\_TREE{Get, Init, Set}=objref]  
 [, IMAGE\_DATA{Get}=variable] [, N\_COLORS{Get, Init}=integer{2 to 256}] [, /PALETTE{Get, Init, Set}=objref] [, /QUALITY{Get, Init, Set}={0 | 1 | 2}]  
 [, /REGISTER\_PROPERTIES{Get, Init, Set}]  
 [, RESOLUTION{Get, Init, Set}={xres, yres}]  
 [, SCREEN\_DIMENSIONS{Get}=variable]  
 [, UNITS{Get, Init, Set}={0 | 1 | 2 | 3}]  
 [, ZBUFFER\_DATA{Get}=variable]

**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={ 1 | 2 }]

**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.

*Obj*->[IDLgrBuffer::]GetDeviceInfo [, ALL=*variable*]  
[, MAX\_NUM\_CLIP\_PLANES=*variable*]  
[, MAX\_TEXTURE\_DIMENSIONS=*variable*]  
[MAX\_TILE\_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  
[, PROPERTY=*variable*]

**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'[, PROPERTY=*value*])  
or Result = *Obj*->[IDLgrBuffer::]Init(  
[, PROPERTY=*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*  
[, DIMENSIONS=*width,height*] [, PATH=*objref(s)*]  
[, PICK\_STATUS=*variable*])

**IDLgrBuffer::QueryRequiredTiles** - Returns an array of named structures containing information regarding which tile data is needed for display. Used with a tiled IDLgrImage object.

Result = *Obj*->[IDLgrBuffer::]QueryRequiredTiles (*View*,  
*Image* [, COUNT=*variable*] [, ALL\_VISIBLE=*value*]  
[, 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*] [, /ORDER]  
[, SUB\_SELECTION=*variable*]  
[, UNITS={ 0 | 1 | 2 | 3 }])

**IDLgrBuffer::SetProperty** - Sets the value of a property or group of properties for the buffer.

*Obj*->[IDLgrBuffer::]SetProperty[, PROPERTY=*value*]

**IDLgrClipboard** - A destination object representing the native clipboard.

**Properties:** [, ALL{Get}=*variable*]  
[, COLOR\_MODEL{Get, Init}={ 0 | 1 }]  
[, DIMENSIONS{Get, Init, Set}=/*width,height*]  
[, GRAPHICS\_TREE{Get, Init, Set}=*objref*]  
[, N\_COLORS{Get, Init}=integer{2 to 256}]  
[, PALETTE{Get, Init, Set}=*objref*]  
[, QUALITY{Get, Init, Set}={ 0 | 1 | 2 }]  
[, /REGISTER\_PROPERTIES{Get, Init, Set}]  
[, RESOLUTION{Get, Init, Set}=*xres,yres*]  
[, SCREEN\_DIMENSIONS{Get}=*variable*]  
[, UNITS{Get, Init, Set}={ 0 | 1 | 2 | 3 }]

**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*] [, /CMYK]  
[, FILENAME=*string*] [, /ISOLATIN1]  
[, POSTSCRIPT=*value*] [, VECT\_SHADING={ 0 | 1 }]  
[, VECT\_SORTING={ 0 | 1 }]  
[, VECT\_TEXT\_RENDER\_METHOD={ 0 | 1 }]  
[, 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.

*Obj*->[IDLgrClipboard::]GetDeviceInfo [, ALL=*variable*]  
[, MAX\_NUM\_CLIP\_PLANES=*variable*]  
[, MAX\_TEXTURE\_DIMENSIONS=*variable*]  
[MAX\_TILE\_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  
[, *PROPERTY=variable*]

**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'[, *PROPERTY=value*])  
or *Result = Obj*->[IDLgrClipboard::]Init(  
[*PROPERTY=value*])

**IDLgrClipboard::QueryRequiredTiles** - Returns an array of named structures containing information regarding which tile data is needed for display. Used with a tiled IDLgrImage object.

*Result = Obj*->[IDLgrClipboard::]QueryRequiredTiles  
(*View, Image* [, COUNT=*variable*]  
[, ALL\_VISIBLE=*value*] [, PATH=*objref(s)*])

**IDLgrClipboard::SetProperty** - Sets the value of a property or group of properties for the clipboard buffer.

*Obj*->[IDLgrClipboard::]SetProperty  
[, *PROPERTY=value*]

**IDLgrColorbar** - Consists of a color-ramp with an optional framing box and annotation axis.

**Properties:** [, ALL{Get}=*variable*]  
[, BLUE\_VALUES{Get, Init, Set}=*vector*]  
[, COLOR{Get, Init, Set}=*index or RGB vector*]  
[, DIMENSIONS{Get, Init, Set}=*[dx, dy]*]  
[, GREEN\_VALUES{Get, Init, Set}=*vector*]  
[, /HIDE{Get, Init, Set}] [, MAJOR{Get,  
Init,Set}=*integer*] [, MINOR{Get, Init, Set}=*integer*]  
[, PALETTE{Get, Init, Set}=*objref*]  
[, PARENT{Get}=*variable*] [, RED\_VALUES{Get, Init,  
Set}=*vector*] [, SHOW\_AXIS{Get, Init, Set}={0 | 1 | 2}]  
[, /SHOW\_OUTLINE{Get, Init, Set}]  
[, SUBTICKLEN{Get, Init, Set}=*minorTickLength*  
/*majorTickLength*] [, THICK{Get, Init,  
Set}=*points*{1.0 to 10.0}] [, /THREED{Get, Init}]  
[, TICKFORMAT{Get, Init, Set}=*string*]  
[, TICKFRMTDATA{Get, Init, Set}=*value*]  
[, TICKLEN{Get, Init, Set}=*value*] [, TICKTEXT{Get,  
Init, Set}=*objref(s)*] [, TICKVALUES{Get, Init,  
Set}=*vector*] [, TITLE{Get, Init, Set}=*objref*]  
[, XCOORD\_CONV{Get, Init, Set}=*vector*]  
[, XRANGE{Get}=*variable*] [, YCOORD\_CONV{Get,  
Init, Set}=*vector*] [, YRANGE{Get}=*variable*]  
[, ZCOORD\_CONV{Get, Init, Set}=*vector*]  
[, ZRANGE{Get}=*variable*]

**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  
[, *PROPERTY=variable*]

**IDLgrColorbar::Init** - Initializes the colorbar object.

*Obj = OBJ\_NEW*('IDLgrColorbar'[, *aRed, aGreen, aBlue*]  
[, *PROPERTY=value*]) or  
*Result = Obj*->[IDLgrColorbar::]Init(*aRed, aGreen,*  
*aBlue*) [, *PROPERTY=value*])

**IDLgrColorbar::SetProperty** - Sets the value of a property or group of properties for the colorbar.

*Obj*->[IDLgrColorbar::]SetProperty [, *PROPERTY=value*]

**IDLgrContour** - Draws a contour plot from an array or unstructured point data. No superclasses. No subclasses.

**Properties:** [, ALL{Get}=*variable*]  
[, ALPHA\_CHANNEL{Get, Init, Set}=*value*]  
[, AM\_PM{Get, Init, Set}=*vector of two strings*]  
[, ANISOTROPY{Get, Init, Set}=*[x, y, z]*]  
[, C\_COLOR{Get, Init, Set}=*vector*]  
[, C\_FILL\_PATTERN{Get, Init, Set}=*array of*  
*IDLgrPattern objects*] [, C\_LABEL\_INTERVAL{Get,  
Init, Set}=*vector*] [, C\_LABEL\_NOGAPS{Get, Init,  
Set}=*array of object references*]  
[, C\_LABEL\_SHOW{Get, Init, Set}=*vector of integers*]  
[, C\_LINestyle{Get, Init, Set}=*array of linestyles*]  
[, C\_THICK{Get, Init, Set}=*float array*{each element  
1.0 to 10.0}] [, C\_USE\_LABEL\_COLOR{Get, Init,  
Set}= *vector of values*]  
[, C\_USE\_LABEL\_ORIENTATION {Get, Init,  
Set}=*vector of values*] [, C\_VALUE{Get, Init,  
Set}=*scalar or vector*] [, CLIP\_PLANES{Get, Init,  
Set}=*array*] [, COLOR{Get, Init, Set}=*index or RGB*  
*vector*] [, DATA\_VALUES{Get, Init, Set}=*vector or 2D*  
*array*] [, DAYS\_OF\_WEEK{Get, Init, Set}=*vector of*  
*seven strings*] [, DEPTH\_OFFSET{Get, Init, Set}=*value*]  
[, DEPTH\_TEST\_DISABLE{Get, Init, Set}={0 | 1 | 2}]  
[, DEPTH\_TEST\_FUNCTION{Get, Init, Set}={0 | 1 | 2 |  
3 | 4 | 5 | 6 | 7 | 8}] [, DEPTH\_WRITE\_DISABLE{Get,  
Init, Set}={0 | 1 | 2}] [, /DOUBLE\_DATA{Init}]  
[, /DOUBLE\_GEOM{Init}] [, /DOWNHILL{Get, Init,  
Set}] [, /FILL{Get, Init, Set}] [, GEOM{Get}=*variable*]  
[, GEOMX{Init, Set}=*vector or 2D array*]  
[, GEOMY{Init, Set}=*vector or 2D array*]  
[, GEOMZ{Init, Set}=*scalar, vector, or 2D array*]  
[, /HIDE{Get, Init, Set}] [, LABEL\_FONT{Get, Init,  
Set}=*objref*] [, LABEL\_FORMAT{Get, Init, Set}=*string*]  
[, LABEL\_FRMTDATA{Init}=*value*]  
[, LABEL\_UNITS{Get, Init, Set}=*string*]  
[, MAX\_VALUE{Get, Init, Set}=*value*]

[, MIN\_VALUE{Get, Init, Set}=value] [, MONTHS{Get, Init, Set}=vector of 12 values] [, N\_LEVELS{Get, Init, Set}=value] [, PALETTE{Get, Init, Set}=objref] [, PARENT{Get}=variable] [, /PLANAR{Get, Init, Set}]

### IDLgrContour Properties - continued

[, POLYGONS{Get, Init, Set}=array of polygon descriptions] [, /REGISTER\_PROPERTIES{Get, Init, Set}] [, SHADE\_RANGE{Get, Init, Set}=[min, max]] [, SHADER{Get, Init, Set}=objref] [, SHADING{Get, Init, Set}={0 | 1}] [, TICKINTERVAL{Get, Set}=value] [, TICKLEN{Get, Init, Set}=value] [, USE\_TEXT\_ALIGNMENTS{Get, Init, Set}=value] [, XCOORD\_CONV{Get, Init, Set}=vector] [, XRANGE{Get}=variable] [, YCOORD\_CONV{Get, Init, Set}=vector] [, YRANGE{Get}=variable] [, ZCOORD\_CONV{Get, Init, Set}=vector] [, ZRANGE{Get}=variable]

**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::GetLabelInfo** - Retrieves information about the labels for a contour  
*Obj*->[IDLgrContour::]GetLabelInfo, *Destination*,  
*LevelIndex* [, LABEL\_OFFSETS=variable]  
[, LABEL\_POLYS=variable]  
[, LABEL\_OBJECTS=variable]

**IDLgrContour::GetProperty** - Retrieves the value of a property or group of properties for the contour.  
*Obj*->[IDLgrContour::]GetProperty  
[, PROPERTY=variable]

**IDLgrContour::Init** - Initializes the contour object.  
*Obj* = OBJ\_NEW('IDLgrContour'[, *Values*]  
[, PROPERTY=value]) or  
*Result* = *Obj*->[IDLgrContour::]Init([*Values*]  
[, PROPERTY=value])

**IDLgrContour::SetProperty** - Sets the value of a property or group of properties for the contour.  
*Obj*->[IDLgrContour::]SetProperty[, PROPERTY=value]

**IDLgrFilterChain** - Provides a way to apply a sequence of image filtering shader programs in the order that they are added to this specialized container object (if there is sufficient graphics hardware support).

**Properties:** [, /FORCE\_FILTER{Get, Init}]

**IDLgrFilterChain::Cleanup** - Performs all cleanup on the object.  
OBJ\_DESTROY, *Obj* or *Obj*->[IDLgrFilterChain::]  
Cleanup

**IDLgrFilterChain::GetProperty** - Retrieves the value of a property or group of properties for the filter chain object.  
*Obj*->[IDLgrFilterChain::]GetProperty  
[, PROPERTY=variable]

**IDLgrFilterChain::Init** - Initializes the filter chain object.  
*Obj* = OBJ\_NEW('IDLgrFilterChain'  
[, PROPERTY=value]) or  
*Result* = *Obj*->[IDLgrFilterChain::]Init  
([, PROPERTY=value])

**IDLgrFilterChain::SetProperty** - Sets the value of a property or group of properties for the filter chain object.  
*Obj*->[IDLgrFilterChain::]SetProperty  
[, PROPERTY=value]

**IDLgrFont** - Represents a typeface, style, weight, and point size that may be associated with text objects.

**Properties:** [, ALL{Get}=variable] [, SIZE{Get, Init, Set}=points] [, SUBSTITUTE{Get, Init, Set}={'Helvetica' | 'Courier' | 'Times' | 'Symbol' | 'Hershey'}] [, THICK{Get, Init, Set}=points{1.0 to 10.0}] [, UVALUE{Get, Init, Set}=value]

**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 [, PROPERTY=variable]

**IDLgrFont::Init** - Initializes the font object.  
*Obj* = OBJ\_NEW('IDLgrFont'[, *Fontname*]  
[, PROPERTY=value])  
or  
*Result* = *Obj*->[IDLgrFont::]Init([*Fontname*]  
[, PROPERTY=value])

**IDLgrFont::SetProperty** - Sets the value of a property or group of properties for the font.  
*Obj*->[IDLgrFont:]SetProperty[, PROPERTY=value]

**IDLgrImage** - Represents a mapping from a 2D array of data values to a 2D array of pixel colors.

**Properties:** [, ALL{Get}=variable]  
[, ALPHA\_CHANNEL=value]  
[, BLEND\_FUNCTION{Get, Init, Set}=vector]  
[, CHANNEL{Get, Init, Set}=hexadecimal bitmask]  
[, CLIP\_PLANES{Get, Init, Set}=array] [, DATA{Get, Init, Set}=nxm, 2xnxm, 3xnxm, or 4xnxm array of image data] [, DEPTH\_OFFSET=value]  
[, DEPTH\_TEST\_DISABLE{Get, Init, Set}={0 | 1 | 2}]  
[, DEPTH\_TEST\_FUNCTION{Get, Init, Set}={0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8}] [, DEPTH\_WRITE\_DISABLE{Get, Init, Set}={0 | 1 | 2}] [, DIMENSIONS{Get, Init, Set}=[width, height]] [, /GREYSCALE{Get, Init, Set}] [, /HIDE{Get, Init, Set}] [, IMAGE\_ID{Get, Init, Set}=value] [, INTERLEAVE{Get, Init, Set}={0 | 1 | 2}]

```
[, INTERNAL_DATA_TYPE{Get, Init, Set}=value]
[, /INTERPOLATE{Get, Init, Set}] [LOCATION{Get,
Init, Set}=[x, y] or [x, y, z]] [, /ORDER{Get, Init, Set}]
[, PALETTE{Get, Init, Set}=objref] [, PARENT
{Get}=variable] [, /REGISTER_PROPERTIES{Get, Init,
Set}] [, RENDER_METHOD=value]
[, /RESET_DATA{Init, Set}] [, SHADER{Get, Init,
Set}=objref] [, SHARE_DATA{Init, Set}=objref]
[, SUB_RECT{Get, Init, Set}=[x, y, xdim, ydim]]
[, TILE_BORDER_SIZE{Get, Init, Set}=value]
[, TILE_COLOR{Get, Init, Set}=[red, green, blue]]
[, TILE_CURRENT_LEVEL{Get, Init, Set}=value]
[, TILE_DIMENSIONS{Get, Init, Set}=[width, height]]
[, TILE_LEVEL_MODE{Get, Init, Set}={0 | 1}]
[, TILE_NUM_LEVELS{Get, Init, Set}=value]
[, TILE_SHOW_BOUNDARIES{Get, Init, Set}={0 | 1}]
[, TILED_IMAGE_DIMENSIONS{Get, Init,
Set}=[width, height]] [, /TILING{Get, Init, Set}]
[, TRANSFORM_MODE={0 | 1}]
[, XCOORD_CONV{Get, Init, Set}=vector]
[, XRANGE{Get}=variable] [YCOORD_CONV{Get,
Init, Set}=vector] [, YRANGE{Get}=variable]
[, ZCOORD_CONV{Get, Init, Set}=vector]
[, ZRANGE{Get}=variable]
```

**IDLgrImage::Cleanup** - Performs all cleanup on the object.

*Obj\_DESTROY*, *Obj* or *Obj*->[IDLgrImage::]Cleanup

**IDLgrImage::DeleteTileData** - Removes tile data stored in the cache.

*Obj*->[IDLgrImage::]DeleteTileData, *TileInfo*, [, /ALL]

**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
[, PROPERTY=variable]

**IDLgrImage::Init** - Initializes the image object.

*Obj* = OBJ\_NEW('IDLgrImage', *ImageData*)
[, PROPERTY=value] [, /NO\_COPY]) or
*Result* = *Obj*->[IDLgrImage::]Init([*ImageData*]
[, PROPERTY=value] [, /NO\_COPY])

**IDLgrImage::ReadFilteredData** - Returns the image data after the application of one or more IDLgrShader programs to the image.

*data* = *Obj*->[IDLgrImage::]ReadFilteredData(*Dest*
[, DATA\_TYPE=value] [, READ\_POSITION=value]
[, TILE=tileinfo])

**IDLgrImage::SetProperty** - Sets the value of the property or group of properties for the image.

*Obj*->[IDLgrImage::]SetProperty[, PROPERTY=value]
[, /NO\_COPY]

**IDLgrImage::SetTileData** - Stores tile data in the tile data cache.

*Obj*->[IDLgrImage::]SetTileData, *TileInfo*, *TileData*,
[, NO\_FREE=value]

**IDLgrImage::TileDataLoaded** - Queries an image tile to determine whether data is loaded.

*Result* = *Obj*->[IDLgrImage::]TileData(*TileInfo*)

**IDLgrLegend** - Provides a simple interface for displaying a legend.

**Properties:** [, ALL{Get}=variable]
[, BORDER\_GAP{Get, Init, Set}=value]
[, COLUMNS{Get, Init, Set}=integer]
[, FILL\_COLOR{Get, Init, Set}=index or RGB vector]
[, FONT{Get, Init, Set}=objref] [, GAP{Get, Init,
Set}=value] [, GLYPH\_WIDTH{Get, Init, Set}=value]
[, /HIDE{Get, Init, Set}] [, ITEM\_COLOR{Get, Init,
Set}=array of colors] [, ITEM\_LINestyle{Get, Init,
Set}=integer array] [, ITEM\_OBJECT{Get, Init,
Set}=array of IDLgrSymbol or IDLgrPattern objrefs]
[, ITEM\_THICK{Get, Init, Set}=float array{each
element 1.0 to 10.0}] [, ITEM\_TYPE{Get, Init, Set}=int
array{each element 0 or 1}] [, OUTLINE\_COLOR{Get,
Init, Set}=index or RGB vector] [, OUTLINE\_THICK
{Get, Init, Set}=points{1.0 to 10.0}] [, PARENT
{Get}=variable] [, RECOMPUTE{Set}={0 | 1}{0
prevents recompute, 1 is the default}]
[, /SHOW\_FILL{Get, Init, Set}]
[, /SHOW\_OUTLINE{Get, Init, Set}]
[, TEXT\_COLOR{Get, Init, Set}=index or RGB vector]
[, TITLE{Get, Init, Set}=objref]
[, XCOORD\_CONV{Get, Init, Set}=vector]
[, XRANGE{Get}=variable] [, YCOORD\_CONV{Get,
Init, Set}=vector] [, YRANGE{Get}=variable]
[, ZCOORD\_CONV{Get, Init, Set}=vector]
[, ZRANGE{Get}=variable]

**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
[, PROPERTY=variable]

**IDLgrLegend::Init** - Initializes the legend object.

*Obj* = OBJ\_NEW('IDLgrLegend', *altNames*)
[, PROPERTY=value] or
*Result* = *Obj*->[IDLgrLegend::]Init([*altNames*]
[, PROPERTY=value])

**IDLgrLegend::SetProperty** - Sets the value of a property or group of properties for the legend.  
*Obj*->[IDLgrLegend::]SetProperty[, *PROPERTY=value*]

**IDLgrLight** - Represents a source of illumination for 3D graphic objects.

**Properties:** [, ALL{Get}=variable]  
 [, ATTENUATION{Get, Init, Set}=[constant, linear, quadratic]] [, COLOR{Get, Init, Set}=[R, G, B]]  
 [, CONEANGLE{Get, Init, Set}=degrees]  
 [, DIRECTION{Get, Init, Set}=3-element vector]  
 [, FOCUS{Get, Init, Set}=value] [, /HIDE{Get, Init, Set}] [, INTENSITY{Get, Init, Set}=value{0.0 to 1.0}]  
 [, LIGHT\_INDEX{Get, Init, Set}=value]  
 [, LOCATION{Get, Init, Set}={x, y, z}]  
 [, PARENT{Get}=variable]  
 [, /REGISTER\_PROPERTIES{Get, Init, Set}]  
 [, SHADER{Get, Init, Set}=objref] [, TYPE{Get, Init, Set}={0 | 1 | 2 | 3}] [, XCOORD\_CONV{Get, Init, Set}=vector] [, YCOORD\_CONV{Get, Init, Set}=vector] [, ZCOORD\_CONV{Get, Init, Set}=vector]

**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 [, *PROPERTY=variable*]

**IDLgrLight::Init** - Initializes the light object.  
*Obj* = OBJ\_NEW('IDLgrLight'[ , *PROPERTY=value*]) or  
*Result* = *Obj*->[IDLgrLight::]Init([*PROPERTY=value*])

**IDLgrLight::SetProperty** - Sets the value of a property or group of properties for the light.  
*Obj*->[IDLgrLight::]SetProperty[, *PROPERTY=value*]

**IDLgrModel** - Represents a graphical item or group of items that can be transformed (rotated, scaled, and/or translated).

**Properties:** [, ACTIVE\_POSITION{Get, Init, Set}=integer] [, ALL{Get}=variable] [, /HIDE{Get, Init, Set}] [, LIGHTING{Get, Init, Set}={0 | 1 | 2}]  
 [, PARENT{Get}=variable]  
 [, /REGISTER\_PROPERTIES{Get, Init, Set}]  
 [, RENDER\_METHOD{Get, Init, Set}=integer]  
 [, /SELECT\_TARGET{Get, Init, Set}]  
 [, TRANSFORM{Get, Init, Set}=4x4 transformation matrix]

**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 subclassing 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  
 [, *PROPERTY=variable*]

**IDLgrModel::Init** - Initializes the model object.  
*Obj* = OBJ\_NEW('IDLgrModel'[ , *PROPERTY=value*]) or  
*Result* = *Obj*->[IDLgrModel::]Init([*PROPERTY=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[, *PROPERTY=value*]

**IDLgrModel::Translate** - Translates model by specified offsets.  
*Obj*->[IDLgrModel::]Translate, *Tx*, *Ty*, *Tz*  
 [, /PREMULTIPLY]

**IDLgrMPEG** - Creates an MPEG movie from an array of image frames.  
**Properties:** [, ALL{Get}=variable] [, BITRATE{Get, Init, Set}=value] [, DIMENSIONS{Get, Init, Set}=2-element array] [, FILENAME{Get, Init, Set}=string]  
 [, FORMAT{Get, Init, Set}={0 | 1}] [, FRAME\_RATE{Get, Init, Set}={1 | 2 | 3 | 4 | 5 | 6 | 7 | 8}]  
 [, IFRAME\_GAP{Get, Init, Set}=integer value]  
 [, /INTERLACED{Get, Init, Set}]  
 [, MOTION\_VEC\_LENGTH{Get, Init, Set}={1 | 2 | 3}]  
 [, QUALITY{Get, Init, Set}=value{0 to 100}]  
 [, SCALE{Get, Init, Set}={xscale, yscale}]  
 [, /STATISTICS{Get, Init, Set}]  
 [, TEMP\_DIRECTORY{Init}=string]

**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  
[, *PROPERTY=variable*]

**IDLgrMPEG::Init** - Initializes the MPEG object.

*Obj* = OBJ\_NEW('IDLgrMPEG', [ *PROPERTY=value*]) or  
*Result* = *Obj*->[IDLgrMPEG::]Init([*PROPERTY=value*])

**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*]

**IDLgrMPEG::SetProperty** - Sets the value of a property or group of properties for the MPEG object.

*Obj*->[IDLgrMPEG::]SetProperty[, *PROPERTY=value*]

**IDLgrPalette** - Represents a color lookup table that maps indices to red, green, and blue values.

**Properties:** [, ALL{Get}=*variable*]  
[, BLUE\_VALUES{Get, Init, Set}=*vector*]  
[, BOTTOM\_STRETCH{Get, Init, Set}=*value*{0 to 100}] [, GAMMA{Get, Init, Set}=*value*{0.1 to 10.0}]  
[, GREEN\_VALUES{Get, Init, Set}=*vector*]  
[, N\_COLORS{Get}=*variable*] [, NAME{Get, Init, Set}=string] [, RED\_VALUES{Get, Init, Set}=*vector*]  
[, TOP\_STRETCH{Get, Init, Set}=*value*{0 to 100}]  
[, UVALUE{Get, Init, Set}=*value*]

**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  
[, *PROPERTY=variable*]

**IDLgrPalette::Init** - Initializes a palette object.

*Obj* = OBJ\_NEW('IDLgrPalette', *aRed*, *aGreen*, *aBlue*  
[, *PROPERTY=value*]) or  
*Result* = *Obj*->[IDLgrPalette::]Init([*aRed*, *aGreen*, *aBlue*]  
[, *PROPERTY=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[, *PROPERTY=value*]

**IDLgrPattern** - Describes which pixels are filled and which are left blank when an area is filled.

**Properties:** [, ALL{Get}=*variable*] [, NAME{Get, Init, Set}=string] [, ORIENTATION{Get, Init, Set}=ccw degrees from horiz] [, PATTERN{Get, Init, Set}=32 x 32 bit array] [, SPACING{Get, Init, Set}=points] [, STYLE{Get, Init, Set}={0 | 1 | 2}] [, THICK{Init}=points{1.0 to 10.0}] [, UVALUE{Get, Init, Set}=value]

**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  
[, *PROPERTY=variable*]

**IDLgrPattern::Init** - Initializes the pattern object.

*Obj* = OBJ\_NEW('IDLgrPattern', [ *Style*]  
[, *PROPERTY=value*]) or  
*Result* = *Obj*->[IDLgrPattern::]Init([*Style*]  
[, *PROPERTY=value*])

**IDLgrPattern::SetProperty** - Sets the value of a property or group of properties for the pattern.

*Obj*->[IDLgrPattern::]SetProperty[, *PROPERTY=value*]

**IDLgrPlot** - Creates set of polylines connecting data points in 2D space.

**Properties:** [, ALL{Get}=*variable*]  
[, ALPHA\_CHANNEL{Get, Init, Set}=value]  
[, CLIP\_PLANES{Get, Init, Set}=array] [, COLOR{Get, Init, Set}=index or RGB vector] [, VERT\_COLORS{Get, Init, Set}=vector] [, DATA{Get}=variable] [, DATAX{Init, Set}=vector] [, DATAY{Init, Set}=vector]  
[, DEPTH\_TEST\_DISABLE{Get, Init, Set}={0 | 1 | 2}]  
[, DEPTH\_TEST\_FUNCTION{Get, Init, Set}={0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8}] [, DEPTH\_WRITE\_DISABLE{Get, Init, Set}={0 | 1 | 2}] [, DOUBLE{Get, Init, Set}=value][, /HIDE{Get, Init, Set}]  
[, /HISTOGRAM{Get, Set}] [, LINESSTYLE{Get, Init, Set}=integer or two-element vector]  
[, MAX\_VALUE{Get, Init, Set}=value]  
[, MIN\_VALUE{Get, Init, Set}=value] [, NSUM{Get, Init, Set}=value] [, PALETTE{Get, Init, Set}=objref]  
[, PARENT{Get}=variable] [, /POLAR{Get, Init, Set}]  
[, /REGISTER\_PROPERTIES{Get, Init, Set}]  
[, /RESET\_DATA{Init, Set}] [, SHADER{Get, Init,

Set)=*objref*] [, SHARE\_DATA{Init, Set}=*objref*]  
 [, SYMBOL{Get, Init, Set}=*objref(s)*] [, THICK{Get,  
 Init, Set}=*points*{1.0 to 10.0}] [, /USE\_ZVALUE{Init}]  
 [, XCOORD\_CONV{Get, Init, Set}=*vector*]  
 [, XRANGE{Get, Init, Set}=*[xmin, xmax]*]  
 [, YCOORD\_CONV{Get, Init, Set}=*vector*]  
 [, YRANGE{Get, Init, Set}=*[ymin, ymax]*]  
 [, ZCOORD\_CONV{Get, Init, Set}=*vector*]  
 [, ZRANGE{Get}=*variable*] [, ZVALUE{Get, Init,  
 Set}=*value*]

**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[, PROPERTY=*variable*]

**IDLgrPlot::GetVertexAttributeData** - Retrieves the value of the attribute variable specified in the *Name* argument (a case-sensitive string).

*Obj*->[IDLgrPlot::]GetVertexAttributeData, *Name*, *Value*

**IDLgrPlot::Init** - Initializes the plot object.

*Obj* = OBJ\_NEW('IDLgrPlot', [X, Y]  
 [, PROPERTY=*value*]) or  
 Result = *Obj*->[IDLgrPlot::]Init([[X, Y]  
 [, PROPERTY=*value*])

**IDLgrPlot::SetProperty** - Sets the value of the property or group of properties for the plot.

*Obj*->[IDLgrPlot::]SetProperty[, PROPERTY=*value*]

**IDLgrPlot::SetVertexAttributeData** - Sets the value of the vertex attribute data variable specified in the *Name* argument (a case-sensitive string).

*Obj*->[IDLgrPlot::]SetVertexAttributeData, *Name*, *Value*

**IDLgrPolygon** - Represents one or more polygons that share a set of vertices and rendering attributes.

**Properties:** [, ALL{Get}=*variable*]  
 [, ALPHA\_CHANNEL{Get, Init, Set}=*value*]  
 [, AMBIENT{Get, Init, Set}=*RGB vector*]  
 [, BOTTOM{Get, Init, Set}=*index or RGB vector*]  
 [, CLIP\_PLANES{Get, Init, Set}=*array*] [, COLOR{Get,  
 Init, Set}=*index or RGB vector* / , VERT\_COLORS{Get,  
 Init, Set}=*vector*] [, DATA{Get, Init, Set}=*array*]  
 [, DEPTH\_OFFSET{Get, Init, Set}=*value*]  
 [, DEPTH\_TEST\_DISABLE{Get, Init, Set}={0 | 1 | 2}]  
 [, DEPTH\_TEST\_FUNCTION{Get, Init, Set}={0 | 1 | 2 |  
 3 | 4 | 5 | 6 | 7 | 8}] [, DEPTH\_WRITE\_DISABLE{Get,  
 Init, Set}={0 | 1 | 2}] [, DIFFUSE{Get, Init, Set}=*RGB  
 vector*] [, DOUBLE{Get, Init, Set}=*value*]  
 [, EMISSION{Get, Init, Set}=*RGB vector*]

[, FILL\_PATTERN{Get, Init, Set}=*objref* to  
 IDLgrPattern object] [, /HIDDEN\_LINES{Init}]  
 [, /HIDE{Get, Init, Set}] [, LINSTYLE{Get, Init,  
 Set}=*value*] [, NAME{Get, Init, Set}=*string*]  
 [, NORMALS{Get, Init, Set}=*array*]  
 [, PALETTE{Init}=*objref*] [, PARENT{Get}=*variable*]  
 [, PART\_COLORS=*vector*] [, PART\_HIDE=*vector*]  
 [, POLYGONS{Get, Init, Set}=*array of polygon  
 descriptions*] [, /REGISTER\_PROPERTIES{Get, Init,  
 Set}] [, REJECT{Get, Init, Set}={0 | 1 | 2}]  
 [, /RESET\_DATA{Init, Set}] [, SHADE\_RANGE{Get,  
 Init, Set}=*array*] [, SHADER{Get, Init, Set}=*objref*]  
 [, SHADING{Get, Init, Set}={0 | 1}]  
 [, SHARE\_DATA{Init, Set}=*objref*] [, SHININESS{Get,  
 Init, Set}=*value*] [, SPECULAR{Get, Init, Set}= *RGB  
 vector*] [, STYLE{Get, Init, Set}={0 | 1 | 2}]  
 [, TEXTURE\_COORD{Get, Init, Set}=*array*]  
 [, /TEXTURE\_INTERP{Get, Init, Set}]  
 [, TEXTURE\_MAP{Get, Init, Set}=*objref* to IDLgrImage  
 object] [, THICK{Get, Init, Set}=*points*{1.0 to 10.0}]  
 [, XCOORD\_CONV{Get, Init, Set}=*vector*]  
 [, XRANGE{Get}=*variable*] [, YCOORD\_CONV{Get,  
 Init, Set}=*vector*] [, ZCOORD\_CONV{Get, Init,  
 Set}=*vector*] [, YRANGE{Get}=*variable*]  
 [, ZERO\_OPACITY\_SKIP{Get, Init, Set}={0 | 1}]  
 [, ZRANGE{Get}=*variable*]

**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  
 [, PROPERTY=*variable*]

**IDLgrPolygon::GetMultiTextureCoord** - Retrieves the texture coordinates associated with the specified *Unit* argument.

*Obj*->[IDLgrPolygon::]GetMultiTextureCoord, *Unit*,  
*TexCoord*

**IDLgrPolygon::GetVertexAttributeData** - Retrieves the value of the attribute variable specified in the *Name* argument (a case-sensitive string).

*Obj*->[IDLgrPolygon::]GetVertexAttributeData, *Name*,  
*Value*

**IDLgrPolygon::Init** - Initializes the polygons object.

*Obj* = OBJ\_NEW('IDLgrPolygon', X [, Y, Z])  
 [, PROPERTY=*value*]) or  
 Result = *Obj*->[IDLgrPolygon::]Init([X, [Y, [Z]]  
 [, PROPERTY=*value*])



**IDLgrPolygon::SetMultiTextureCoord** - Sets the texture coordinates associated with the specified *Unit* argument.

*Obj*->[IDLgrPolygon::]SetMultiTextureCoord, *Unit*,  
*TexCoord*

**IDLgrPolygon::SetProperty** - Sets the value of the property or group of properties for the polygons.

*Obj*->[IDLgrPolygon::]SetProperty [, *PROPERTY=value*]

**IDLgrPolygon::SetVertexAttributeData** - Sets the value of the vertex attribute data variable specified in the *Name* argument (a case-sensitive string).

*Obj*->[IDLgrPolygon::]SetVertexAttributeData, *Name*,  
*Value*

**IDLgrPolyline** - Represents one or more polylines that share a set of vertices and rendering attributes.

**Properties:** [, ALL{Get}=*variable*]  
[, ALPHA\_CHANNEL{Get, Init, Set}=*value*]  
[, CLIP\_PLANES{Get, Init, Set}=*array*] [, COLOR{Get, Init, Set}=*index* or *RGB vector*] [, VERT\_COLORS{Get, Init, Set}=*vector*] [, DATA{Get, Init, Set}=*array*]  
[, DEPTH\_TEST\_DISABLE{Get, Init, Set}={0 | 1 | 2}]  
[, DEPTH\_TEST\_FUNCTION{Get, Init, Set}={0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8}] [, DEPTH\_WRITE\_DISABLE{Get, Init, Set}={0 | 1 | 2}] [, DOUBLE{Get, Init, Set}=*value*]  
[, /HIDE{Get, Init, Set}] [, LABEL\_NOGAPS{Get, Init, Set}=*vector*] [, LABEL\_OBJECTS{Get, Init, Set}=*objref*] [, LABEL\_OFFSETS{Get, Init, Set}=*scalar* or *vector of offsets*] [, LABEL\_POLYLINES{Get, Init, Set}=*scalar* or *vector of polyline indices*]  
[, LABEL\_USE\_VERTEX\_COLOR{Get, Init, Set}=*value*] [, LINSTYLE{Get, Init, Set}=*value*]  
[, PALETTE{Get, Init, Set}=*objref*]  
[, PARENT{Get}=*variable*] [, PART\_COLORS=*vector*]  
[, PART\_HIDE=*vector*] [, POLYLINES{Get, Init, Set}=*array of polyline descriptions*]  
[, /REGISTER\_PROPERTIES{Get, Init, Set}]  
[, /RESET\_DATA{Init, Set}] [, SHADER{Get, Init, Set}=*objref*] [, SHADING{Get, Init, Set}={0 | 1}]  
[, SHARE\_DATA{Init, Set}=*objref*] [, SYMBOL{Get, Init, Set}=*objref(s)*] [, THICK{Get, Init, Set}=*points*{1.0 to 10.0}] [, USE\_LABEL\_COLOR{Get, Init, Set}=*vector*] [, USE\_LABEL\_ORIENTATION{Get, Init, Set}=*vector*] [, /USE\_TEXT\_ALIGNMENTS{Get, Init, Set}] [, XCOORD\_CONV{Get, Init, Set}=*vector*]  
[, XRANGE{Get}=*variable*] [, YCOORD\_CONV{Get, Init, Set}=*vector*] [, YRANGE{Get}=*variable*]  
[, ZCOORD\_CONV{Get, Init, Set}=*vector*]  
[, ZRANGE{Get}=*variable*]

**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  
[, *PROPERTY=variable*]

**IDLgrPolyline::GetVertexAttributeData** - Retrieves the value of the attribute variable specified in the *Name* argument (a case-sensitive string).

*Obj*->[IDLgrPolyline::]GetVertexAttributeData, *Name*,  
*Value*

**IDLgrPolyline::Init** - Initializes the polylines object.

*Obj* = OBJ\_NEW('IDLgrPolyline' [, X [, Y[, Z]])  
[, *PROPERTY=value*] or  
*Result* = *Obj*->[IDLgrPolyline::]Init( [X, [Y, [Z]]]  
[, *PROPERTY=value*])

**IDLgrPolyline::SetProperty** - Sets the value of a property or group of properties for the polylines.

*Obj*->[IDLgrPolyline::]SetProperty [, *PROPERTY=value*]

**IDLgrPolyline::SetVertexAttributeData** - Sets the value of the vertex attribute data variable specified in the *Name* argument (a case-sensitive string).

*Obj*->[IDLgrPolyline::]SetVertexAttributeData, *Name*,  
*Value*

**IDLgrPrinter** - Represents a hardcopy graphics destination.

**Properties:** [, ALL{Get}=*variable*]  
[, COLOR\_MODEL{Get, Init}={0 | 1}]  
[, DIMENSIONS{Get}=*variable*]  
[, GRAPHICS\_TREE{Get, Init, Set}=*objref of type IDLgrScene, IDLgrViewgroup, or IDLgrView*]  
[, /LANDSCAPE{Get, Init, Set}] [, N\_COLORS{Get, Init, Set}=*integer*{2 to 256}] [, N\_COPIES{Get, Init, Set}=*integer*] [, NAME{Get, Init, Set}=*string*]  
[, PALETTE{Get, Init, Set}=*objref*]  
[, PRINT\_QUALITY{Get, Init, Set}={0 | 1 | 2}]  
[, /PRINTER\_NAME{Get}=*variable*] [, QUALITY{Get, Init, Set}={0 | 1 | 2}] [, /REGISTER\_PROPERTIES{Get, Init, Set}] [, RESOLUTION{Get}=*variable*]  
[, UNITS{Get, Init, Set}={0 | 1 | 2 | 3}]

**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*]  
[, VECT\_SORTING={0 | 1}]  
[, VECT\_TEXT\_RENDER\_METHOD={0 | 1}]  
[, 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  
[, PROPERTY=*variable*]

**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'[, PROPERTY=*value*])* or  
*Result = Obj->*[IDLgrPrinter::]Init([PROPERTY=*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::QueryRequiredTiles** - Returns an array of named structures containing information regarding which tile data is needed for display. Used with a tiled IDLgrImage object.

*Result = Obj->*[IDLgrPrinter::]QueryRequiredTiles (*View*,  
*Image* [, COUNT=*variable*] [, ALL\_VISIBLE=*value*]  
[, PATH=*objref(s)*])

**IDLgrPrinter::SetProperty** - Sets the value of a property or group of properties for the printer.

*Obj->*[IDLgrPrinter::]SetProperty[, PROPERTY=*value*]

**IDLgrROI** - Object graphics representation of a region of interest.

**Properties:** [, ALL{Get}=*variable*]  
[, ALPHA\_CHANNEL{Get, Init, Set}=*value*]  
[, CLIP\_PLANES{Get, Init, Set}=array]  
[, COLOR{Get, Init, Set}=vector]  
[, DEPTH\_TEST\_DISABLE{Get, Init, Set}={0 | 1 | 2}]  
[, DEPTH\_TEST\_FUNCTION{Get, Init, Set}={0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8}] [, DEPTH\_WRITE\_DISABLE{Get, Init, Set}={0 | 1 | 2}] [, DOUBLE{Get, Init, Set}=*value*]  
[, /HIDE{Get, Init, Set}] [, LINSTYLE{Get, Init, Set}=*value*] [, PALETTE{Get, Init, Set}=objref]  
[, /REGISTER\_PROPERTIES{Get, Init, Set}]  
[, SHADER{Get, Init, Set}=objref] [, STYLE{Get, Init, Set}={0 | 1 | 2 | 3}] [, SYMBOL{Get, Init, Set}=objref]  
[, THICK{Get, Init, Set}=points{1.0 to 10.0}]

[, XCOORD\_CONV{Get, Init, Set}={s<sub>0</sub>, s<sub>1</sub>}]

[, XRANGE{Get}=*variable*]

[, YCOORD\_CONV{Get, Init, Set}={s<sub>0</sub>, s<sub>1</sub>}]

[, YRANGE{Get}=*variable*]

[, ZCOORD\_CONV{Get, Init, Set}={s<sub>0</sub>,

s<sub>1</sub>} [, ZRANGE{Get}=*variable*]

**IDLgrROI::Cleanup** - Performs all cleanup for the object.

OBJ\_DESTROY, *Obj* or *Obj ->* [IDLgrROI::]Cleanup

**IDLgrROI::GetProperty** - Retrieves the value of a property or group of properties for the Object Graphics region.

*Obj ->* [IDLgrROI::]GetProperty[, PROPERTY=*variable*]

**IDLgrROI::Init** - Initializes an Object Graphics region of interest.

*Obj = OBJ\_NEW('IDLgrROI' [, X[, Y[, Z]]*  
[, PROPERTY=*value*]) or

*Result = Obj ->* [IDLgrROI::]Init([X[, Y[, Z]]  
[, PROPERTY=*value*])

**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[, PROPERTY=*value*]

**IDLgrROIgroup** - Object Graphics representation of a group of regions of interest.

**Properties:** [, ALL{Get}=*variable*]  
[, CLIP\_PLANES{Get, Init, Set}=array]  
[, DEPTH\_TEST\_DISABLE{Get, Init, Set}={0 | 1 | 2}]  
[, DEPTH\_TEST\_FUNCTION{Get, Init, Set}={0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8}] [, DEPTH\_WRITE\_DISABLE{Get, Init, Set}={0 | 1 | 2}] [, /HIDE{Get, Init, Set}]  
[, SHADER{Get, Init, Set}=objref]  
[, PARENT{Get}=*variable*]  
[, XCOORD\_CONV{Get, Init, Set}={s<sub>0</sub>, s<sub>1</sub>}]  
[, XRANGE{Get}=*variable*]  
[, YCOORD\_CONV{Get, Init, Set}={s<sub>0</sub>, s<sub>1</sub>}]  
[, YRANGE{Get}=*variable*]  
[, ZCOORD\_CONV{Get, Init, Set}={s<sub>0</sub>, s<sub>1</sub>}]  
[, ZRANGE{Get}=*variable*]

**IDLgrROIgroup::Add** - Adds a region to the region group.

*Obj ->* [IDLgrROIgroup::]Add, *ROI*

**IDLgrROIgroup::Cleanup** - Performs all cleanup for the object.

OBJ\_DESTROY, *Obj* or  
*Obj ->* [IDLgrROIgroup::]Cleanup

**IDLgrROIgroup::GetProperty** - Retrieves the value of a property or group of properties for the region group.

*Obj ->* [IDLgrROIgroup::]GetProperty  
[, PROPERTY=*variable*]

**IDLgrROIgroup::Init** - Initializes an Object Graphics region of interest group object.

*Obj* =  
OBJ\_NEW('IDLgrROIgroup'[, *PROPERTY=value*]) or  
*Result* = *Obj*->[IDLgrROIgroup::]Init(  
[, *PROPERTY=value*])

**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*] )

**IDLgrROIgroup::Remove** - Removes an object from the ROI group.

*Obj*->[IDLgrROIgroup::]Remove [, *Child\_object*]  
[, *POSITION=index*] [, /ALL]

**IDLgrROIgroup::SetProperty** - Sets the value of a property or group of properties for the region group.

*Obj* -> [IDLgrROIgroup::]SetProperty  
[, *PROPERTY=value*]

**IDLgrScene** - Represents the entire scene to be drawn and serves as a container of IDLgrView or IDLgrViewgroup objects.

**Properties:** [, ALL{Get}=*variable*] [, COLOR{Get, Init,  
Set}=*index* or *RGB vector*] [, /HIDE{Get, Init, Set}]  
[, /REGISTER\_PROPERTIES{Get, Init, Set}]  
[, /TRANSPARENT{Get, Init, Set}]

**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 [, *PROPERTY=variable*]

**IDLgrScene::Init** - Initializes the scene object.

*Obj* = OBJ\_NEW('IDLgrScene'[, *PROPERTY=value*]) or  
*Result* = *Obj*->[IDLgrScene::]Init([, *PROPERTY=value*])

**IDLgrScene::SetProperty** - Sets the value of one or more properties for the scene.

*Obj*->[IDLgrScene::]SetProperty[, *PROPERTY=value*]

**IDLgrShader** - Exposes the OpenGL shading language to IDL object graphics. This lets you associate a shader program (which is executed on the graphics card processing unit) with a graphic object, increasing performance and interactivity.

**Properties:** [, CACHE\_RESULT{Get, Init, Set}=*value*]  
[, /FORCE\_FILTER{Get, Init}]  
[, FRAGMENT\_PROGRAM\_FILENAME{Get, Init,  
Set}=*string*]  
[, FRAGMENT\_PROGRAM\_STRING{Get, Init,  
Set}=*string*] [, OUTPUT\_DATA\_TYPE{Get, Init,

Set}=*value*] [, VERTEX\_PROGRAM\_FILENAME{Get,  
Init, Set}=*string*]  
[, VERTEX\_PROGRAM\_STRING{Get, Init,  
Set}=*string*]

**IDLgrShader::Cleanup** - Performs all cleanup on the object.

OBJ\_DESTROY, *Obj* or *Obj*->[IDLgrShader::]Cleanup

**IDLgrShader::Filter** - provides a software fallback for an image filtering operation when there is not support for the shader program. Implemented *only* in an IDLgrShader subclass.

*Result* = *Obj*->[IDLgrShader subclass::]Filter(*Image*)

**IDLgrShader::GetProperty** - Retrieves the value of a property or group of properties for the shader.

*Obj*->[IDLgrShader::]GetProperty  
[, *PROPERTY=variable*]

**IDLgrShader::GetUniformVariable** - Retrieves the value of the uniform variable specified in the *Name* argument (a case-sensitive string).

*Obj*->[IDLgrShader::]GetUniformVariable, *Name*, *Value*

**IDLgrShader::Init** - Initializes the shader object.

*Obj* = OBJ\_NEW('IDLgrShader' [, *PROPERTY=value*]) or  
*Result* = *Obj*->[IDLgrShader::]Init  
([, *PROPERTY=value*])

**IDLgrShader::SetProperty** - Sets the value of a property or group of properties for the shader.

*Obj*->[IDLgrShader::]SetProperty[, *PROPERTY=value*]

**IDLgrShader::SetUniformVariable** - Sets the value of the uniform variable specified in the *Name* argument (a case-sensitive string).

*Obj*->[IDLgrShader::]SetUniformVariable, *Name*, *Value*

**IDLgrShaderBytscl** - Performs byte-scale operations on image data as it is draw to the screen using a shader program if there is sufficient graphics hardware support or using a software fallback.

**Properties:** [, IN\_RANGE{Get, Init}=*array*]  
[, OUT\_RANGE{Get, Init, Set}=*array*]  
[, UNITS\_IN\_RANGE{Get, Init, Set}=*value*]

**IDLgrShaderBytscl::Cleanup** - Performs all cleanup on the object.

OBJ\_DESTROY, *Obj* or *Obj*-  
>[IDLgrShaderBytscl::]Cleanup

**IDLgrShaderBytscl::GetProperty** - Retrieves the value of a property or group of properties for the byte-scale shader.

*Obj*->[IDLgrShaderBytscl::]GetProperty  
[, *PROPERTY=variable*]

**IDLgrShaderBytscl::Init** - Initializes the byte-scale shader object.

*Obj* = OBJ\_NEW('IDLgrShaderBytscl'  
[, *PROPERTY=value*]) or  
*Result* = *Obj*->[IDLgrShaderBytscl::]Init  
([, *PROPERTY=value*])

**IDLgrShaderBytscl::SetProperty** - Sets the value of a property or group of properties for the byte-scale shader.

*Obj*->[IDLgrShaderBytscl::]SetProperty  
[, *PROPERTY=value*]

**IDLgrShaderConvolve3** - Performs a convolution operation (smooth, sharpen, edge detection, or custom) on image data as it is drawn to the screen using a shader (with sufficient graphics hardware support).

**Properties:** [, BASE\_BLEND\_FACTOR{Get, Init, Set}=value] [, CONVOLVE\_BLEND\_FACTOR{Get, Init, Set}=value] [, KERNEL{Get, Init, Set}=value or 9-element numerical array]

**IDLgrShaderConvolve3::Cleanup** - Performs all cleanup on the object.

OBJ\_DESTROY, *Obj* or *Obj*->[IDLgrShaderConvolve3::]Cleanup

**IDLgrShaderConvolve3::GetProperty** - Retrieves the value of a property or group of properties for the convolution shader.

*Obj*->[IDLgrConvolve3::]GetProperty  
[, PROPERTY=variable]

**IDLgrShaderConvolve3::Init** - Initializes the convolution shader object.

*Obj* = OBJ\_NEW('IDLgrShaderConvolve3'  
[, PROPERTY=value]) or  
*Result* = *Obj*->[IDLgrShaderConvolve3::]Init  
([, PROPERTY=value])

**IDLgrShaderConvolve3::SetProperty** - Sets the value of a property or group of properties for the convolution shader.

*Obj*->[IDLgrShaderConvolve3::]SetProperty  
[, PROPERTY=value]

**IDLgrSurface** - A shaded or vector representation of a mesh grid.

**Properties:** [, ALL{Get}=variable]  
[, ALPHA\_CHANNEL{Get, Init, Set}=value]  
[, AMBIENT{Get, Init, Set}=RGB vector]  
[, BOTTOM{Get, Set}=index or RGB vector]  
[, CLIP\_PLANES{Get, Init, Set}=array] [, COLOR{Get, Init, Set}=index or RGB vector] [, DATA{Init, Set}=vector or 2D array] [, DATAX{Init, Set}=vector or 2D array] [, DATAZ{Init, Set}=2D array]  
[, DEPTH\_OFFSET{Get, Init, Set}=value]  
[, DEPTH\_TEST\_DISABLE{Get, Init, Set}={0 | 1 | 2}]  
[, DEPTH\_TEST\_FUNCTION{Get, Init, Set}={0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8}] [, DEPTH\_WRITE\_DISABLE{Get, Init, Set}={0 | 1 | 2}] [, DIFFUSE{Get, Init, Set}=RGB vector] [, DOUBLE{Get, Init, Set}=value]  
[, EMISSION{Get, Init, Set}=RGB vector]  
[, /EXTENDED\_LEGO{Get, Init, Set}]  
[, /HIDDEN\_LINES{Get, Init, Set}] [, /HIDE{Get, Init, Set}] [, LINSTYLE{Get, Init, Set}=value]  
[, MAX\_VALUE{Get, Init, Set}=value]  
[, MIN\_VALUE{Get, Init, Set}=value] [, PALETTE{Get, Init, Set}=objref] [, PARENT{Get}=variable]  
[, /REGISTER\_PROPERTIES{Get, Init, Set}]  
[, /RESET\_DATA{Init, Set}] [, SHADE\_RANGE{Get, Init, Set}=index of darkest pixel, index of brightest pixel] [, SHADER{Get, Init, Set}=objref]  
[, SHADING{Get, Init, Set}={0 | 1}]

[, SHARE\_DATA{Init, Set}=objref] [, SHININESS{Get, Init, Set}=value] [, /SHOW\_SKIRT{Get, Init, Set}]  
[, SKIRT{Get, Init, Set}=Z value] [, SPECULAR{Get, Init, Set}=RGB vector] [, STYLE{Get, Init, Set}={0 | 1 | 2 | 3 | 4 | 5 | 6}] [, TEXTURE\_COORD{Get, Init, Set}=array] [, /TEXTURE\_HIGHRES{Get, Init, Set}]  
[, /TEXTURE\_INTERP{Get, Init, Set}]

**IDLgrSurface Properties - continued**

[, TEXTURE\_MAP{Get, Init, Set}=objref to IDLgrImage]  
[, THICK{Get, Init, Set}=points{1.0 to 10.0}]  
[, /USE\_TRIANGLES{Get, Init, Set}]  
[, VERT\_COLORS{Get, Init, Set}=vector]  
[, XCOORD\_CONV{Get, Init, Set}=vector]  
[, XRANGE{Get}=variable] [, YCOORD\_CONV{Get, Init, Set}=vector] [, YRANGE{Get}=variable]  
[, ZCOORD\_CONV{Get, Init, Set}=vector]  
[, ZERO\_OPACITY\_SKIP{Get, Init, Set}={0 | 1}]  
[, ZRANGE{Get}=variable]

**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::GetMultiTextureCoord** - Retrieves the texture coordinates associated with the specified *Unit* argument.

*Obj*->[IDLgrSurface::]SetMultiTextureCoord, *Unit*,  
*TexCoord*

**IDLgrSurface::GetProperty** - Retrieves the value of a property or group of properties for the surface.

*Obj*->[IDLgrSurface::]GetProperty  
[, PROPERTY=variable]

**IDLgrSurface::GetVertexAttributeData** - Retrieves the value of the attribute variable specified in the *Name* argument (a case-sensitive string).

*Obj*->[IDLgrSurface::]GetVertexAttributeData, *Name*,  
*Value*

**IDLgrSurface::Init** - Initializes the surface object.

*Obj* = OBJ\_NEW('IDLgrSurface' [, Z [, X, Y]]  
[, PROPERTY=value]) or  
*Result* = *Obj*->[IDLgrSurface::]Init([Z [, X, Y]]  
[, PROPERTY=value])

**IDLgrSurface::SetMultiTextureCoord** - Sets the texture coordinates associated with the specified *Unit* argument.

*Obj*->[IDLgrSurface::]SetMultiTextureCoord, *Unit*,  
*TexCoord*

**IDLgrSurface::SetProperty** - Sets the value of a property or group of properties for the surface.

*Obj*->[IDLgrSurface::]SetProperty[, PROPERTY=value]

**IDLgrSurface::SetVertexAttributeData** - Sets the value of the vertex attribute data variable specified in the *Name* argument (a case-sensitive string).

*Obj*->[IDLgrSurface::]SetVertexAttributeData, *Name*,  
*Value*

**IDLgrSymbol** - Represents a graphical element that is plotted relative to a particular position.

**Properties:** [, ALL{Get}=variable]  
[, ALPHA\_CHANNEL{Get, Init, Set}=value]  
[, COLOR{Get, Init, Set}=index or RGB vector]  
[, DATA{Get, Init, Set}=integer or objref] [, NAME{Get, Init, Set}=string] [, SIZE{Get, Init, Set}=vector]  
[, THICK{Get, Init, Set}=points{1.0 to 10.0}]  
[, UVALUE{Get, Init, Set}=value]

**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  
[, PROPERTY=variable]

**IDLgrSymbol::Init** - Initializes the plot symbol.

*Obj* = OBJ\_NEW('IDLgrSymbol'[ , *Data*]  
[, PROPERTY=value] or  
*Result* = *Obj*->[IDLgrSymbol::]Init([*Data*]  
[, PROPERTY=value])

**IDLgrSymbol::SetProperty** - Sets the value of a property or group of properties for the symbol.

*Obj*->[IDLgrSymbol::]SetProperty[, PROPERTY=value]

**IDLgrTessellator** - Converts a simple concave polygon (or a simple polygon with "holes") into a number of simple convex polygons (general triangles).

**IDLgrTessellator::AddPolygon** - Adds a polygon to the tessellator object.

*Obj*->[IDLgrTessellator::]AddPolygon, *X* [, *Y*, *Z*]  
[, AUXDATA=array of auxiliary data] [, /INTERIOR]  
[, POLYGON=array of polygon descriptions]

**IDLgrTessellator::Cleanup** - Performs all cleanup on the object.

OBJ\_DESTROY, *Obj* or  
*Obj*->[IDLgrTessellator::]Cleanup

**IDLgrTessellator::Init** - Initializes the tessellator object.

*Obj* = OBJ\_NEW('IDLgrTessellator') 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* [, AUXDATA=variable] [, /QUIET] )

**IDLgrText** - Represents one or more text strings that share common rendering attributes.

**Properties:** [, ALL{Get}=variable] [, ALIGNMENT{Get, Init, Set}=value{0.0 to 1.0}]

[, ALPHA\_CHANNEL{Get, Init, Set}=value{0.0 to 1.0}] [, BASELINE{Get, Init, Set}=vector]  
[, CHAR\_DIMENSIONS{Get, Init, Set}=[width, height]]  
[, CLIP\_PLANES{Get, Init, Set}=array]  
[, COLOR{Get, Init, Set}=index or RGB vector]  
[, DEPTH\_TEST\_DISABLE{Get, Init, Set}={0 | 1 | 2}]  
[, DEPTH\_TEST\_FUNCTION{Get, Init, Set}={0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8}] [, DEPTH\_WRITE\_DISABLE{Get, Init, Set}={0 | 1 | 2}] [, /ENABLE\_FORMATTING{Get, Init, Set}] [, /FILL\_BACKGROUND{Get, Init, Set}]  
[, FILL\_COLOR{Get, Init, Set}=index or RGB vector]  
[, FONT{Get, Init, Set}=objref] [, /HIDE{Get, Init, Set}] [, /KERNING{Get, Init, Set}] [, LOCATIONS{Get, Init, Set}=array] [, NAME{Get, Init, Set}=string]  
[, /ONGLASS{Get, Init, Set}] [, PALETTE{Get, Init, Set}=objref] [, PARENT{Get}=variable]  
[, RECOMPUTE\_DIMENSIONS{Get, Init, Set}={0 | 1 | 2}] [, /REGISTER\_PROPERTIES{Get, Init, Set}]  
[, RENDER\_METHOD{Get, Init, Set}={0 | 1}]  
[, SHADER{Get, Init, Set}=objref] [, STRINGS{Get, Init, Set}=string or vector of strings] [, UPDIR{Get, Init, Set}=vector] [, UVALUE{Get, Init, Set}=value]  
[, VERTICAL\_ALIGNMENT{Get, Init, Set}=value{0.0 to 1.0}] [, XCOORD\_CONV{Get, Init, Set}=vector]  
[, XRANGE{Get}=variable] [, YCOORD\_CONV{Get, Init, Set}=vector]

**IDLgrText Properties** - *continued*

[, YRANGE{Get}=variable] [, ZCOORD\_CONV{Get, Init, Set}=vector] [, ZRANGE{Get}=variable]

**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[, PROPERTY=variable]

**IDLgrText::Init** - Initializes the text object.

*Obj* = OBJ\_NEW('IDLgrText'[ , *String/string array*]  
[, PROPERTY=value] or  
*Result* = *Obj*->[IDLgrText::]Init( *String/string array* )  
[, PROPERTY=value])

**IDLgrText::SetProperty** - Sets the value of a property or group of properties for the text.

*Obj*->[IDLgrText::]SetProperty[, PROPERTY=value]

**IDLgrView** - Represents a rectangular area in which graphics objects are drawn. It is a container for objects of the IDLgrModel class.

**Properties:** [, ALL{Get}=variable]  
[, PARENT{Get}=variable] [, COLOR{Get, Init, Set}=index or RGB vector] [, DEPTH\_CUE{Get, Init,

Set)=[*zbright*, *zdim*] [, DIMENSIONS{Get, Init, Set}=*value*]  
 Set)=[*width*, *height*] [, DOUBLE {Get, Init, Set}=*value*]  
 [, EYE{Get, Init, Set}=*distance*] [, LOCATION{Get, Init, Set}=*x*, *y*] [, PROJECTION{Get, Init, Set}={1 | 2}]  
 [, /REGISTER\_PROPERTIES{Get, Init, Set}]  
 [, /TRANSPARENT{Get, Init, Set}] [, UNITS{Get, Init, Set}={0 | 1 | 2 | 3}] [, VIEWPLANE\_RECT{Get, Init, Set}=*x*, *y*, *width*, *height*] [, ZCLIP{Get, Init, Set}=*near*, *far*]

**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 [, *PROPERTY*=*variable*]

**IDLgrView::Init** - Initializes the view object.

*Obj* = OBJ\_NEW('IDLgrView'[, *PROPERTY*=*value*]) or  
*Result* = *Obj*->[IDLgrView::]Init(*PROPERTY*=*value*)

**IDLgrView::SetProperty** - Sets the value of the property or group of properties for the view.

*Obj*->[IDLgrView::]SetProperty[, *PROPERTY*=*value*]

**IDLgrViewgroup** - A simple container object that contains one or more IDLgrView objects. An IDLgrScene can contain one or more of these objects.

**Properties:** [, ALL{Get}=*variable*] [, /HIDE{Get, Init, Set}] [, PARENT{Get}=*variable*]  
 [, /REGISTER\_PROPERTIES{Get, Init, Set}]

**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  
 [, *PROPERTY*=*variable*]

**IDLgrViewgroup::Init** - Initializes the viewgroup object.

*Obj* = OBJ\_NEW('IDLgrViewgroup'[, *PROPERTY*=*value*])  
 or *Result* = *Obj*->[IDLgrViewgroup::]Init(*PROPERTY*=*value*)

**IDLgrViewgroup::SetProperty** - Sets the value of a property or group of properties for the viewgroup.

*Obj*->[IDLgrViewgroup::]SetProperty  
 [, *PROPERTY*=*value*]

**IDLgrVolume** - Represents mapping from a 3D array of data to a 3D array of voxel colors, which, when drawn, are projected to two dimensions.

**Properties:** [, ALL{Get}=*variable*]

[, ALPHA\_CHANNEL{Get, Init, Set}=*value*]  
 [, AMBIENT{Get, Init, Set}=*RGB vector*]  
 [, BOUNDS{Get, Init, Set}=*xmin*, *ymin*, *zmin*, *xmax*, *ymax*, *zmax*] [, CLIP\_PLANES{Get, Init, Set}=*array*]  
 [, COMPOSITE\_FUNCTION{Get, Init, Set}={0 | 1 | 2 | 3}]  
 [, DATA0{Get, Init, Set}=[*d<sub>x</sub>*, *d<sub>y</sub>*, *d<sub>z</sub>*] [, DATA1{Get, Init, Set}=[*d<sub>x</sub>*, *d<sub>y</sub>*, *d<sub>z</sub>*] [, DATA2{Get, Init, Set}=[*d<sub>x</sub>*, *d<sub>y</sub>*, *d<sub>z</sub>*] [, DATA3{Get, Init, Set}=[*d<sub>x</sub>*, *d<sub>y</sub>*, *d<sub>z</sub>*] [, DEPTH\_CUE{Get, Init, Set}=[*zbright*, *zdim*] [, /HIDE{Get, Init, Set}]  
 [, HINTS{Get, Init, Set}={0 | 1 | 2 | 3}]  
 [, /INTERPOLATE{Get, Init, Set}]  
 [, /LIGHTING\_MODEL{Get, Init, Set}]  
 [, OPACITY\_TABLE0{Get, Init, Set}=256-element byte array] [, OPACITY\_TABLE1{Get, Init, Set}=256-element byte array] [, PARENT{Get}=*variable*]  
 [, RENDER\_STEP{Get, Init, Set}=*x*, *y*, *z*] [, RGB\_TABLE0{Get, Init, Set}=256 x 3-element byte array] [, RGB\_TABLE1{Get, Init, Set}=256 x 3-element byte array] [, /REGISTER\_PROPERTIES{Get, Init, Set}]  
 [, SHADER{Get, Init, Set}=*objref*] [, /TWO\_SIDED{Get, Init, Set}] [, VALID\_DATA{Get}=*variable*]

**IDLgrVolume Properties** - *continued*

[, VOLUME\_SELECT{Get, Init, Set}={0 | 1 | 2}]  
 [, XCOORD\_CONV{Get, Init, Set}=*vector*]  
 [, XRANGE{Get}=*variable*] [, YCOORD\_CONV{Get, Init, Set}=*vector*] [, YRANGE{Get}=*variable*]  
 [, /ZBUFFER{Get, Init, Set}] [, ZCOORD\_CONV{Get, Init, Set}=*vector*] [, ZRANGE{Get}=*variable*]  
 [, ZERO\_OPACITY\_SKIP{Get, Init, Set}={0 | 1}]

**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  
[, *PROPERTY=variable*] [, /NO\_COPY]

**IDLgrVolume::Init** - Initializes the volume object.

*Obj* = OBJ\_NEW('IDLgrVolume' [, *vol0* [, *vol1* [, *vol2* [, *vol3*]]]) [, *PROPERTY=value*] [, /NO\_COPY]) or  
*Result* = *Obj*->[IDLgrVolume::]Init( [*vol0* [, *vol1* [, *vol2* [, *vol3*]]]) [, *PROPERTY=value*] [, /NO\_COPY])

**IDLgrVolume::PickVoxel** - Computes the coordinates of the voxel projected to a location specified by the 2D device coordinates point, [*x<sub>i</sub>*, *y<sub>i</sub>*], 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[, *PROPERTY=value*]  
[, /NO\_COPY]

**IDLgrVRML** - Saves the contents of an Object Graphics hierarchy into a VRML 2.0 format file.

**Properties:** [, ALL{Get}=*variable*]  
[, SCREEN\_DIMENSIONS{Get}=*variable*]  
[, COLOR\_MODEL{Get, Init}={0 | 1}]  
[, DIMENSIONS{Get, Init, Set}={width, height}]

**IDLgrVRML Properties** - *continued*

[, FILENAME{Get, Init, Set}=string]  
[, GRAPHICS\_TREE{Get, Init, Set}=objref]  
[, N\_COLORS{Get, Init}=integer{2 to 256}]  
[, PALETTE{Get, Init, Set}=objref] [, QUALITY{Get, Init, Set}={0 | 1 | 2}] [, /REGISTER\_PROPERTIES{Get, Init, Set}] [, RESOLUTION{Get, Init, Set}={xres, yres}]  
[, UNITS{Get, Init, Set}={0 | 1 | 2 | 3}]  
[, WORLDINFO{Init}=string array]  
[, WORLDTITLE{Init}=string]

**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.

*Obj*->[IDLgrVRML::]GetDeviceInfo [, ALL=*variable*]  
[, MAX\_NUM\_CLIP\_PLANES=*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  
[, *PROPERTY=variable*]

**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'[, *PROPERTY=value*]) or  
*Result* = *Obj*->[IDLgrVRML::]Init(*PROPERTY=value*)

**IDLgrVRML::SetProperty** - Sets the value of a property or group of properties for the VRML world.

*Obj*->[IDLgrVRML::]SetProperty[, *PROPERTY=value*]

**IDLgrWindow** - Represents an on-screen area on a display device that serves as a graphics destination.

**Properties:** [, ALL{Get}=*variable*]  
[, COLOR\_MODEL{Get, Init}={0 | 1}]  
[, DIMENSIONS{Get, Init, Set}={width, height}]  
[, DISPLAY\_NAME{Get}(X Windows Only)=*variable*]  
[, IMAGE\_DATA{Get}=*variable*]  
[, GRAPHICS\_TREE{Get, Init, Set}=objref of type  
*IDLgrScene*, *IDLgrViewgroup*, or *IDLgrView*]  
[, LOCATION{Get, Init, Set}={x, y}]  
[, MINIMUM\_VIRTUAL\_DIMENSIONS{Get, Init, Set}={width, height}] [, N\_COLORS{Get, Init}=integer{2 to 256}] [, PALETTE{Get, Init, Set}=objref] [, QUALITY{Get, Init, Set}={0 | 1 | 2}]  
[, /REGISTER\_PROPERTIES{Get, Init, Set}]  
[, RENDERER{Get, Init}={0 | 1}] [, RESOLUTION{Get}=*variable*] [, RETAIN{Get, Init}={0 | 1 | 2}]  
[, SCREEN\_DIMENSIONS{Get}=*variable*]  
[, TITLE{Get, Init, Set}=string] [, UNITS{Get, Init, Set}={0 | 1 | 2 | 3}]  
[, VIEWPORT\_DIMENSIONS{Get}={width, height}]  
[, VIRTUAL\_DIMENSIONS{Get, Init, Set}={width, height}] [, VISIBLE\_LOCATION{Get, Init, Set}={x, y}]  
[, ZBUFFER\_DATA{Get}=*variable*]  
[, ZOOM\_BASE{Get, Init, Set}=value]  
[, ZOOM\_NSTEP{Get}=*variable*]

**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={1 | 2}]

**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*] [ , FRAMEBUFFER\_OBJECT\_EXTENSION=*variable*] [ , MAX\_FRAGMENT\_UNIFORM\_COMPONENTS=*variable*] [ , MAX\_NUM\_CLIP\_PLANES=*variable*] [ , MAX\_TEXTURE\_DIMENSIONS=*variable*] [ , MAX\_TILE\_DIMENSIONS=*variable*] [ , MAX\_TEXTURE\_IMAGE\_UNITS=*variable*] [ , MAX\_VERTEX\_ATTRIBUTES=*variable*] [ , MAX\_VERTEX\_TEXTURE\_IMAGE\_UNITS=*variable*] [ , MAX\_VERTEX\_UNIFORM\_COMPONENTS=*variable*] [ , MAX\_VIEWPORT\_DIMENSIONS=*variable*] [ , NAME=*variable*] [ , NUM\_CPUS=*variable*] [ , SHADING\_LANGUAGE\_VERSION=*variable*] [ , VENDOR=*variable*] [ , VERSION=*variable*]

**IDLgrWindow::GetDimensions** - Returns a two-element vector, [width, height], representing the visible dimensions (in device units) of this window.

*Result = Obj ->* [IDLgrWindow::]GetDimensions( [ , MINIMUM\_VIRTUAL\_DIMENSIONS=*variable*] [ , ORIGINAL\_VIRTUAL\_DIMENSIONS=*variable*] [ , VIRTUAL\_DIMENSIONS=*variable*] [ , VISIBLE\_LOCATION=*variable*] )

**DLgrWindow::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  
[ , PROPERTY=*variable*]

**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'[ , PROPERTY=*value*])*  
or *Result = Obj->*[IDLgrWindow::]Init(  
[ , PROPERTY=*value*])

**IDLgrWindow::OnEnter** - Handles notification from the window that a mouse enter event has occurred.

*Obj->*[IDLgrWindow::]OnEnter

**IDLgrWindow::OnExit** - Handles notification from the window that a mouse exit event has occurred.

*Obj->*[IDLgrWindow::]OnExit

**IDLgrWindow::OnExpose** - Handles notification from the window that an expose event has occurred.

*Obj->*[IDLgrWindow::]OnExpose, *X, Y, Width, Height*

**IDLgrWindow::OnKeyboard** - Handles notification (from the native window device) that a keyboard event has occurred, and passes along that notification to all observers in the list of window event observers.

*Obj->*[IDLgrWindow::]OnKeyboard, *IsASCII, Character, KeySymbol, X, Y, Press, Release, Modifiers*

**IDLgrWindow::OnMouseDown** - Handles notification (from the native window device) that a mouse down event has occurred, and passes along that notification to all observers in the list of window event observers.

*Obj->*[IDLgrWindow::]OnMouseDown, *X, Y, ButtonMask, Modifiers, NumClicks*

**IDLgrWindow::OnMouseMove** - Handles notification (from the native window device) that a mouse motion event has occurred, and passes along that notification to all observers in the list of window event observers.

*Obj->*[IDLgrWindow::]OnMouseMove, *X, Y, Modifiers*

**IDLgrWindow::OnMouseUp** - Handles notification (from the native window device) that a mouse up event has occurred, and passes along that notification to all observers in the list of window event observers.

*Obj->*[IDLgrWindow::]OnMouseUp, *X, Y, ButtonMask*

**IDLgrWindow::OnResize** - Handles notification (from the native window device) that a resize event has occurred.

*Obj->*[IDLgrWindow::]OnResize, *Width, Height*

**IDLgrWindow::OnWheel** - Handles notification (from the native window device) that a mouse wheel event has occurred.

*Obj->*[IDLgrWindow::]OnWheel, *X, Y, Delta, Modifiers*

**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* [ , DIMENSIONS=*[width,height]*] [ , PATH=*objref(s)*] [ , PICK\_STATUS=*variable*])

**IDLgrWindow::QueryRequiredTiles** - Returns an array of named structures containing information regarding which tile data is needed for display. Used with a tiled IDLgrImage object.

*Result = Obj->*[IDLgrWindow::]QueryRequiredTiles  
(*View, Image* [ , COUNT=*variable*] [ , ALL\_VISIBLE=*value*] [ , 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]*] [ , /ORDER] [ , SUB\_SELECTION=*variable*] [ , 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::SetCurrentZoom** - Sets the current zoom factor for this window. The current zoom factor, the virtual canvas dimensions, and the visible dimensions of the window are updated to reflect the new zoom factor.

*Obj*->[IDLgrWindow::]SetCurrentZoom , *ZoomFactor* [, /RESET]

**IDLgrWindow::SetProperty** - Sets the value of a property or group of properties for the window.

*Obj*->[IDLgrWindow::]SetProperty [, *PROPERTY=value*]

**IDLgrWindow::Show** - Exposes or hides a window.

*Obj*->[IDLgrWindow::]Show, *Position*

**IDLgrWindow::ZoomIn** - causes the current zoom factor for this window to be increased (that is, multiplied by the factor given by the window's ZOOM\_BASE property).

*Obj*->[IDLgrWindow::]ZoomIn

**IDLgrWindow::ZoomOut** - causes the current zoom factor for this window to be decreased (that is, divided by the factor given by the window's ZOOM\_BASE property).

*Obj*->[IDLgrWindow::]ZoomOut

**IDLitCommand** - The base functionality for the iTools command buffer system.

**Properties:** [, SKIP\_REDO{Get, Init, Set}=0 | 1] [, SKIP\_UNDO{Get, Init, Set}=0 | 1] [, TARGET\_IDENTIFIER{Get, Init, Set}=string] [, OPERATION\_IDENTIFIER{Get, Init, Set}=string]

**IDLitCommand::AddItem** - Adds the specified data item to the data dictionary associated with this object.

*Result* = *Obj*->[IDLitCommand::]AddItem(*StrItem*, *Item* [, /OVERWRITE])

**IDLitCommand::Cleanup** - Performs all cleanup on the object.

OBJ\_DESTROY, *Obj* or  
*Obj*-> [IDLitCommand::]Cleanup()

**IDLitCommand::GetItem** - Retrieves the specified item from the data dictionary associated with this object.

*Result* = *Obj*->[IDLitCommand::]GetItem(*StrItem*, *Item*)

**IDLitCommand::GetProperty** - Retrieves the value of a property or group of properties of a command object.

*Obj*->[IDLitCommand::]GetProperty [, *PROPERTY=variable*]

**IDLitCommand::GetSize** - Returns an approximate value for the amount of memory being used by the items in the data dictionary associated with this object.

*Result* = *Obj*->[IDLitCommand::]GetSize( [, /KILOBYTES])

**IDLitCommand::Init** - Initializes the object and allows specification of items associated with it.

*Obj* = OBJ\_NEW('IDLitCommand' [, *PROPERTY=value*]) or  
*Result* = *Obj*->[IDLitCommand::]Init(  
[*PROPERTY=value*])

**IDLitCommand::SetProperty** - Sets the value of a property or group of properties for the command object.

*Obj*->[IDLitCommand::]SetProperty [, *PROPERTY=value*]

**IDLitCommandSet** - A container for IDLitCommand objects, which allows a group of commands to be managed as a single item.

**IDLitCommandSet::Cleanup** - Performs all cleanup on the object.

OBJ\_DESTROY, *Obj* or  
*Obj*-> [IDLitCommandSet::]Cleanup()

**IDLitCommandSet::GetSize** - Returns an approximate value for the amount of memory being used by the items contained by this command set.

*Result* = *Obj*->[IDLitCommandSet::]GetSize( [, /KILOBYTES])

**IDLitCommandSet::Init** - Initializes the object.

*Obj* = OBJ\_NEW('IDLitCommandSet') or  
*Result* = *Obj*->[IDLitCommandSet::]Init()

**IDLitComponent** - A core or base component, from which all other components subclass.

**Properties:** [, COMPONENT\_VERSION{Get}=value] [, DESCRIPTION{Get, Init, Set}=string] [, ICON{Get, Init, Set}=string] [, HELP{Get, Init, Set}=string] [, IDENTIFIER{Get, Init, Set}=string] [, NAME{Get, Init, Set}=string] [, /PRIVATE{Get, Init, Set}] [, UVALUE{Get, Init, Set}=value]

**IDLitComponent::Cleanup** - Performs all cleanup on the object.

OBJ\_DESTROY, *Obj* or  
*Obj*->[IDLitComponent::]Cleanup

**IDLitComponent::EditUserDefProperty** - Defines the interface that is displayed when a user selects the "Edit" button on a user-defined property in the property sheet interface.

*Result* = *Obj*->[IDLitComponent::]EditUserDefProperty(  
*iTool*, *PropertyIdentifier*)

**IDLitComponent::GetFullIdentifier** - Navigates the iTool object container hierarchy of the object on which it is called and retrieves the fully-qualified object identifier.

*Result* = *Obj*->[IDLitComponent::]GetFullIdentifier(  
[*Objref*])

**IDLitComponent::GetProperty** - Retrieves the value of an IDLitComponent property or properties.

*Obj*->[IDLitComponent::]GetProperty [, *PROPERTY=variable*] [, /NO\_COPY]

**IDLitComponent::GetPropertyAttribute** - Retrieves property attribute values for a registered property.

*Obj*->[IDLitComponent::]GetPropertyAttribute,  
*PropertyIdentifier* [, *TYPE=variable*]

**IDLitComponent::GetPropertyByIdentifier** - Retrieves the value of an IDLitComponent property.

*Result = Obj*-  
>[IDLitComponent::]GetPropertyByIdentifier(  
*PropertyIdentifier*, *Value*)

**IDLitComponent::Init** - Initializes the IDLitComponent object

*Obj = OBJ\_NEW*('IDLitComponent'  
[, *PROPERTY=value*])

or

*Result = Obj*->[IDLitComponent::]Init(  
[, *PROPERTY=value*])

**IDLitComponent::NotifyBridge** - Broadcasts a notification message to the Export Bridge object that is wrapping the IDLitComponent object.

*Result = Obj*->  
[IDLitComponent::]NotifyBridge(*IdMessage*, *Value*)

**IDLitComponent::QueryProperty** - Checks whether a property identifier is registered, or retrieves a list of all registered properties.

*Result = Obj*->[IDLitComponent::]QueryProperty(  
[*PropertyIdentifier*])

**IDLitComponent::RegisterProperty** - Registers a property as belonging to the component.

*Obj*->[IDLitComponent::]RegisterProperty,  
*PropertyIdentifier* [, *Type*] [, /BOOLEAN] [, /COLOR]  
[, /DESCRIPTION=*string*] [, /ENUMLIST=*stringvector*]  
[, /FLOAT] [, /HIDE] [, /INTEGER] [, /LINESTYLE]  
[, /NAME=*string*] [, /SENSITIVE] [, /STRING]  
[, /SYMBOL] [, /THICKNESS] [, /UNDEFINED]  
[, /USERDEF=*string*] [, /VALID\_RANGE=*vector*]

**IDLitComponent::Restore** - Performs any transitional work required after an object has been restored from a SAVE file.

*Obj*->[IDLitComponent::]Restore

**IDLitComponent::SetProperty** - Sets the value of an IDLitComponent property or properties.

*Obj*->[IDLitComponent::]SetProperty  
[, *PROPERTY=value*] [, /NO\_COPY]

**IDLitComponent::SetPropertyAttribute** - Sets one or more property attributes for a registered property.

*Obj*->[IDLitComponent::]SetPropertyAttribute,  
*PropertyIdentifier*

**IDLitComponent::SetPropertyByIdentifier** - Sets the value of an IDLitComponent property.

*Obj*->[IDLitComponent::]SetPropertyByIdentifier,  
*PropertyIdentifier*, *Value*

**IDLitComponent::UpdateComponentVersion** - Updates the value of the COMPONENT\_VERSION property for the specified object to the current release of IDL.

*Obj*->[IDLitComponent::]UpdateComponentVersion

**IDLitContainer** - A specialization of the IDL\_Container class that manages a collection of IDLitComponents and provides methods for working with the Identifier system of the iTools framework.

**IDLitContainer::Add** - Adds items to the container object.

*Obj*->[IDLitContainer::]Add, *Components*  
[, /NO\_NOTIFY]

**IDLitContainer::AddByIdentifier** - Adds an object to the container hierarchy in the position specified by the *Identifier* argument.

*Obj*->[IDLitContainer::]AddByIdentifier, *Identifier*, *Item*  
[, /FOLDER\_CLASSNAME=*string*]

**IDLitContainer::Cleanup** - Performs all cleanup on the object.

OBJ\_DESTROY, *Obj* or *Obj*->[IDLitContainer::]Cleanup

**IDLitContainer::FindIdentifiers** - Retrieves the full identifiers for items within an iTool container.

*Result = Obj*->[IDLitContainer::]FindIdentifiers([*Pattern*]  
[, /COUNT=*variable*] [, /LEAF\_NODES])

**IDLitContainer::Get** - Retrieves items from the container.

*Result = Obj*->[IDLitContainer::]Get([, /ALL]  
[, /COUNT=*variable*] [, /ISA=*string* or *array of strings*]  
[, /POSITION=*index* or *array of indices*]  
[, /SKIP\_PRIVATE])

**IDLitContainer::GetByIdentifier** - Retrieves an object from a container hierarchy using the specified identifier to locate the object.

*Result = Obj*->[IDLitContainer::]GetByIdentifier(  
*Identifier*)

**IDLitContainer::Init** - Initializes the object.

*Obj = OBJ\_NEW*('IDLitContainer') or  
*Result = Obj*->[IDLitContainer::]Init()

**IDLitContainer::Remove** - Removes items from the container.

*Obj*->[IDLitContainer::]Remove, *Components*  
[, /NO\_NOTIFY]

**IDLitContainer::RemoveByIdentifier** - Removes an object from a container hierarchy using the specified identifier to locate the object.

*Result = Obj*->[IDLitContainer::]RemoveByIdentifier(  
*Identifier*)

**IDLitData** - A generic data storage object that can hold any IDL data type available. It provides typing, metadata, and data change notification functionality. When coupled with IDLitDataContainer, it forms the element for the construction of composite data types.

**Properties:** [, /HIDE{Get, Init, Set}]  
[, /READ\_ONLY{Init}] [, /TYPE{Get, Init}=*string*]

**IDLitData::AddDataObserver** - Specifies an object (the *Observer*) that will be notified when the contents of the data object are changed.

*Obj*->[IDLitData::]AddDataObserver, *Observer*

**IDLitData::Cleanup** - Performs all cleanup operations on the object

OBJ\_DESTROY, *Obj* or *Obj*->[IDLitData::]Cleanup

**IDLitData::Copy** - Returns an exact copy of the data object and its contents, including registered property values.

*Result = Obj*->[IDLitData::]Copy()

**IDLitData::GetByType** - Returns all contained objects of the specified iTool data type.

*Result = Obj->[IDLitData::]GetByType(Type  
[, COUNT=variable])*

**IDLitData::GetData** - Retrieves the data stored in the object.

*Result = Obj->[IDLitData::]GetData(Data[, Identifier]  
[, NAN=variable] [, /NO\_COPY])*

**IDLitData::GetProperty** - Retrieves the value of an IDLitData property.

*Obj->[IDLitData::]GetProperty[, PROPERTY=variable]*

**IDLitData::GetSize** - Returns an approximate value for the amount of memory being used by the data object.

*Result = Obj->[IDLitData::]GetSize()*

**IDLitData::Init** - Initializes the IDLitData object.

*Obj = OBJ\_NEW('IDLitData'[, Data]  
[, PROPERTY=value]) or  
Result = Obj->[IDLitData::]Init([Data]  
[, PROPERTY=value])*

**IDLitData::NotifyDataChange** - Is called when a data object has been changed; it is part of the notification process that allows data updates to be reflected by visualizations that use the data. It works in conjunction with the NotifyDataComplete method to provide a two-pass change notification system that minimizes the number of operations performed when a data object changes.

*Obj->[IDLitData::]NotifyDataChange*

**IDLitData::NotifyDataComplete** - Is called after a data object has been changed; it is part of the notification process that allows data updates to be reflected by visualizations that use the data. It works in conjunction with the NotifyDataChange method to provide a two-pass change notification system that minimizes the number of operations performed when a data object changes.

*Obj->[IDLitData::]NotifyDataComplete*

**IDLitData::RemoveDataObserver** - Unregisters an object that has been registered as an *observer* of this data object.

*Obj->[IDLitData::]RemoveDataObserver, Observer*

**IDLitData::SetData** - Copies the data from an IDL variable or expression into the data object, and notifies all its observers that the data has changed.

*Result = Obj->[IDLitData::]SetData(Data[, Identifier]  
[, /NO\_COPY] [, /NULL])*

**IDLitData::SetProperty** - Sets the value of an IDLitData property

*Obj->[IDLitData::]SetProperty[, PROPERTY=value]*

**IDLitDataContainer** - A container for IDLitData and IDLitDataContainer objects. This container is used to form hierarchical data structures. Data and DataContainer objects can be added and removed to/from a DataContainer during program execution, allowing for dynamic creation of composite data types.

**IDLitDataContainer::Add** - Adds items to the data container object.

*Obj->[IDLitDataContainer::]Add, Data [, /NO\_NOTIFY]*

**IDLitDataContainer::Cleanup** - Performs all cleanup operations on the object.

*OBJ\_DESTROY, Obj or  
Obj->[IDLitDataContainer::]Cleanup*

**IDLitDataContainer::GetData** - Retrieves the data value contained in the data object specified by the *Identifier* argument.

*Result = Obj->[IDLitDataContainer::]GetData(Data  
[, Identifier] [, /NO\_COPY])*

**IDLitDataContainer::GetIdentifiers** - Retrieves the object identifiers for all data and data container objects contained in the data container object.

*Result = Obj->[IDLitDataContainer::]GetDataIdentifiers(  
[Pattern] [, /LEAF])*

**IDLitDataContainer::GetProperty** - Retrieves the value of an IDLitDataContainer property

*Obj->[IDLitDataContainer::]GetProperty  
[, PROPERTY=variable]*

**IDLitDataContainer::Init** - Initializes the IDLitDataContainer object.

*Obj = OBJ\_NEW('IDLitDataContainer'[, Data]  
[, PROPERTY=value]) or  
Result = Obj->[IDLitDataContainer::]Init([Data]  
[, PROPERTY=value])*

**IDLitDataContainer::SetData** - Stores data in the IDLitData object specified by *Identifier*.

*Result = Obj->[IDLitDataContainer::]SetData(Data,  
Identifier[, /NO\_COPY] [, /NULL])*

**IDLitDataContainer::SetProperty** - Sets the value of an IDLitDataContainer property.

*Obj->[IDLitDataContainer::]SetProperty  
[, PROPERTY=value]*

**IDLitDataOperation** - A subclass to IDLitOperation that automates data access and automatically records information for the undo-redo system.

**Properties:** [, WITHIN\_UI{GET}=1 | 0]

**IDLitDataOperation::Cleanup** - Performs all cleanup on the object.

*OBJ\_DESTROY, Obj or  
Obj->[IDLitOperation::]Cleanup*

**IDLitDataOperation::DoExecuteUI** - Provides a way for the iTool developer to request user input before performing an operation.

*Result = Obj->[IDLitDataOperation::]DoExecuteUI()*

**IDLitDataOperation::Execute** - Contains the execution logic for the operation.

*Result = Obj->[IDLitDataOperation::]Execute(Data)*

**IDLitDataOperation::GetProperty** - Retrieves the value of a property or group of properties of an operation object.

*Obj->[IDLitDataOperation::]GetProperty  
[, PROPERTY=variable]*

**IDLitDataOperation::Init** - Initializes the IDLitDataOperation object and sets properties that define the behavior the operation provides.

*Obj = OBJ\_NEW('IDLitDataOperation'  
[, PROPERTY=variable]) or  
Result = Obj->[IDLitDataOperation::]Init(  
[, PROPERTY=variable])*

**IDLitDataOperation::SetProperty** - Sets the value of a property or group of properties for the operation.

*Obj*->[IDLitDataOperation::]SetProperty  
[, *PROPERTY=value*]

**IDLitDataOperation::UndoExecute** - IS called when a user selects the Undo operation after executing an IDLitDataOperation that sets the value of the REVERSIBLE\_OPERATION property to 1.

*Result* = *Obj*->[IDLitDataOperation::]UndoExecute( *Data*)

**IDLitDirectWindow** - A representation of an on-screen area on a display device that serves as a graphics destination. In contrast to IDLgrWindow and IDLitWindow, the IDLitDirectWindow uses Direct Graphics. This object is only for use with the Export Bridges.

**Properties:** [, WINDOW\_INDEX=*integer*]

**IDLitDirectWindow::Cleanup** - Performs all cleanup on the object.

OBJ\_DESTROY, *Obj* or  
*Obj*->[IDLitDirectWindow::]Cleanup

**IDLitDirectWindow::GetProperty** - Retrieves the value of an IDLitDirectWindow property

*Obj*->[IDLitDirectWindow::]GetProperty  
[, *PROPERTY=variable*]

**IDLitDirectWindow::Init** - Initializes the window object.

*Obj* = OBJ\_NEW('IDLitDirectWindow'  
[, *PROPERTY=value*]) or  
*Result* = *Obj*->[IDLitDirectWindow::]Init(  
[, *PROPERTY=value*])

**IDLitDirectWindow::MakeCurrent** - Sets the current graphics device to the window device and sets the current output window to the IDLitDirectWindow's window.

*Obj*->[IDLitDirectWindow::]MakeCurrent

**IDLitDirectWindow::OnEnter** - Handles notification from the window that a mouse enter event has occurred.

*Obj*->[IDLitDirectWindow::]OnEnter

**IDLitDirectWindow::OnExit** - Handles notification from the window that a mouse exit event has occurred.

*Obj*->[IDLitDirectWindow::]OnExit

**IDLitDirectWindow::OnExpose** - Handles notification from the window that an expose event has occurred.

*Obj*->[IDLitDirectWindow::]OnExpose, *X*, *Y*, *Width*, *Height*

**IDLitDirectWindow::OnKeyboard** - Handles notification (from the native window device) that a keyboard event has occurred, and passes along that notification to all observers in the list of window event observers.

*Obj*->[IDLitDirectWindow::]OnKeyboard, *IsASCII*,  
*Character*, *KeySymbol*, *X*, *Y*, *Press*, *Release*, *Modifiers*

**IDLitDirectWindow::OnMouseDown** - Handles notification (from the native window device) that a mouse down event has occurred, and passes along that notification to all observers in the list of window event observers.

*Obj*->[IDLitDirectWindow::]OnMouseDown, *X*, *Y*,  
*ButtonMask*, *Modifiers*, *NumClicks*

**IDLitDirectWindow::OnMouseMotion** - Handles notification (from the native window device) that a mouse motion event has occurred, and passes along that notification to all observers in the list of window event observers.

*Obj*->[IDLitDirectWindow::]OnMouseMotion, *X*, *Y*,  
*Modifiers*

**IDLitDirectWindow::OnMouseUp** - Handles notification (from the native window device) that a mouse up event has occurred, and passes along that notification to all observers in the list of window event observers.

*Obj*->[IDLitDirectWindow::]OnMouseUp, *X*, *Y*, *ButtonMask*

**IDLitDirectWindow::OnResize** - Handles notification (from the native window device) that a resize event has occurred.

*Obj*->[IDLitDirectWindow::]OnResize, *Width*, *Height*

**IDLitDirectWindow::OnWheel** - Handles notification (from the native window device) that a mouse wheel event has occurred.

*Obj*->[IDLitDirectWindow::]OnWheel, *X*, *Y*, *Delta*,  
*Modifiers*

**IDLitDirectWindow::SetProperty** - Sets the value of an IDLitDirectWindow property.

*Obj*->[IDLitDirectWindow::]SetProperty[,  
*PROPERTY=value*]

**IDLitMessaging** - An interface providing common methods to send or trigger messaging and error actions, which may occur during execution.

**IDLitMessaging::AddOnNotifyObserver** - Is used to register a specified iTool component object to receive messages generated by the DoOnNotify method of another specified iTool component object.

*Obj*->[IDLitMessaging::]AddOnNotifyObserver,  
*IdObserver*, *IdSubject*

**IDLitMessaging::DoOnNotify** - Is used to broadcast a notification message to iTool component objects that are observing the source of the message.

*Obj*->[IDLitMessaging::]DoOnNotify, *IdOriginator*,  
*IdMessage*, *Value*

**IDLitMessaging::ErrorMessage** - Is used to display an error message to the user.

*Obj*->[IDLitMessaging::]ErrorMessage, *StrMessage*  
[, SEVERITY=*integer*] [, TITLE=*string*]  
[, /USE\_LAST\_ERROR]

**IDLitMessaging::GetTool** - Returns an object reference to the iTool object associated with the object on which it is called.

*Result* = *Obj*->[IDLitMessaging::]GetTool()

**IDLitMessaging::ProbeStatusMessage** - Is used to display a status message to the user, which is displayed in a data-specific region of the user interface.

*Obj*->[IDLitMessaging::]ProbeStatusMessage, *StrMessage*

**IDLitMessaging::ProgressBar** - Is used to display a progress bar to the user and update the displayed values.

*Result* = *Obj*->[IDLitMessaging::]ProgressBar( *StrMessage*  
[, CANCEL=*string*] [, PERCENT=*value*]  
[, /SHUTDOWN])

**IDLitMessaging::PromptUserText** - Is used to prompt the iTool user with a question and retrieve a text answer.

*Result* = *Obj*->[IDLitMessaging::PromptUserText(  
*StrPrompt*, *Answer* [, *TITLE=string*])

**IDLitMessaging::PromptUserYesNo** - Is used to prompt the user with a yes or no question and return an answer.

*Result* = *Obj*->[IDLitMessaging::PromptUserYesNo(  
*StrPrompt*, *Answer* [, *TITLE=string*])

**IDLitMessaging::RemoveOnNotifyObserver** - Is used to un-register a specified iTool component object as wishing to receive messages generated by the DoOnNotify method of another specified iTool component object.

*Obj*->[IDLitMessaging::]RemoveOnNotifyObserver,  
*IdObserver*, *IdSubject*

**IDLitMessaging::SignalError** - Is used to signal an error in the system.

*Obj*->[IDLitMessaging::]SignalError, *StrMessage*  
[, *CODE=string*] [, *SEVERITY=integer*]

**IDLitMessaging::StatusMessage** - Is used to display a status message to the user.

*Obj*->[IDLitMessaging::]StatusMessage, *StrMessage*  
[, *SEGMENT\_IDENTIFIER=string*]

**IDLitManipulator** - The base functionality of the iTools manipulator system.

**Properties:** [, /BUTTON\_EVENTS{Get, Init, Set}]  
[, DEFAULT\_CURSOR{Init}=string]  
[, DESCRIPTION{Get, Init, Set}=string]  
[, /DISABLE{Get, Init, Set}] [, DRAG\_QUALITY{Get,  
Init, Set}={0 | 1 | 2}] [, /KEYBOARD\_EVENTS{Get,  
Init, Set}] [, /MOTION\_EVENTS{Get, Init, Set}]  
[, OPERATION\_IDENTIFIER{Get, Init, Set}=string]  
[, PARAMETER\_IDENTIFIER{Get, Init, Set}=string}]  
[, TRANSIENT\_DEFAULT{Get, Init, Set}={0 | 1 | 2 |  
3}] [, /TRANSIENT\_MOTION{Get, Init, Set}]  
[, TYPES{Get, Init}=string vector]  
[, /VIEWS\_ONLY{Init}] [VISUAL\_TYPE{Get, Init,  
Set}=string]

**IDLitManipulator::Cleanup** - Performs all cleanup on the object.

*OBJ\_DESTROY*, *Obj* or  
*Obj*->[IDLitManipulator::]Cleanup()

**IDLitManipulator::CommitUndoValues** - Is used to complete a transaction that is occurring as a result of the manipulator interaction.

*Result* = *Obj*->[IDLitManipulator::]CommitUndoValues(  
[UNCOMMENT])

**IDLitManipulator::GetCursorType** - Retrieves the name of the cursor to display for the manipulator.

*Result* = *Obj*->[IDLitManipulator::]GetCursorType(  
*TypeIn*, *KeyMods*)

**IDLitManipulator::GetProperty** - Retrieves the value of an IDLitManipulator property

*Obj*->[IDLitManipulator::]GetProperty  
[, *PROPERTY=variable*]

**IDLitManipulator::Init** - Initializes the IDLitManipulator object

*Obj* = *OBJ\_NEW*('IDLitManipulator'  
[, *PROPERTY=value*]) or  
*Result* = *Obj*->[IDLitManipulator::]Init(  
[, *PROPERTY=value*])

**IDLitManipulator::OnKeyboard** - Is used when a keyboard event occurs on the target window (*Win*).

*Obj*->[IDLitManipulator::]OnKeyboard, *Win*, *IsASCII*,  
*Character*, *Key Value*, *X*, *Y*, *Press*, *Release*, *KeyMods*

**IDLitManipulator::OnLoseCurrentManipulator** - Is used when this manipulator is no longer the current manipulator in the system.

*Obj*->[IDLitManipulator::]OnLoseCurrentManipulator

**IDLitManipulator::OnMouseDown** - Is used when a mouse down event occurs on the target window.

*Obj*->[IDLitManipulator::]OnMouseDown, *Win*, *X*, *Y*,  
*IButton*, *KeyMods*, *NClicks*

**IDLitManipulator::OnMouseMove** - Manages the setting of the cursor on the window if no mouse button is down.

*Obj*->[IDLitManipulator::]OnMouseMove, *Win*, *X*, *Y*,  
*KeyMods*

**IDLitManipulator::OnMouseUp** - Is used when a mouse up event occurs on the target window.

*Obj*->[IDLitManipulator::]OnMouseUp, *Win*, *X*, *Y*, *IButton*

**IDLitManipulator::OnWheel** - Handles notification (from the native window device) that a mouse wheel event has occurred.

*Obj*->[IDLitManipulator::]OnWheel, *X*, *Y*, *Delta*,  
*Modifiers*

**IDLitManipulator::RecordUndoValues** - Is used to begin recording the transaction that is occurring as a result of the manipulator interaction. This method works in conjunction with the CommitUndoValues method.

*Result* = *Obj*->[IDLitManipulator::]RecordUndoValues()

**IDLitManipulator::RegisterCursor** - Defines the appearance and name of the cursor associated with the manipulator.

*Obj*->[IDLitManipulator::]RegisterCursor, *ArrCursor*,  
*Name* [, /DEFAULT]

**IDLitManipulator::SetCurrentManipulator** - Sets the manipulator as the current manipulator in the system.

*Obj*->[IDLitManipulator::]SetCurrentManipulator

**IDLitManipulator::SetProperty** - Sets the value of an IDLitManipulator property

*Obj*->[IDLitManipulator::]SetProperty  
[, *PROPERTY=value*]

**IDLitManipulatorContainer** - A container for IDLitManipulator objects, which allows for the construction of manipulator hierarchies. This container implements the concept of a current manipulator for the items it contains.

**Property:** [, /AUTO\_SWITCH{Get, Init, Set}]

**IDLitManipulatorContainer::Add** - Is used to add a new manipulator to the container.

*Obj*->[IDLitManipulatorContainer::]Add, *Manipulator*

**IDLitManipulatorContainer::GetCurrent** - Is used to get the object reference of the current manipulator of the container.

*Result = Obj->[IDLitManipulatorContainer:] GetCurrent()*

**IDLitManipulatorContainer::GetCurrentManipulator** - Is used to get the current manipulator of the system.

*Result = Obj->[IDLitManipulatorContainer:]  
GetCurrentManipulator[, /IDENTIFIER])*

**IDLitManipulatorContainer::GetProperty** - Retrieves the value of an IDLitManipulatorContainer property

*Obj->[IDLitManipulatorContainer:]GetProperty  
[, PROPERTY=variable]*

**IDLitManipulatorContainer::Init** - Initializes the IDLitManipulatorContainer object.

*Obj = OBJ\_NEW('IDLitManipulatorContainer'  
[, PROPERTY=value] or  
Result = Obj->[IDLitManipulatorContainer:]Init(  
[PROPERTY=value])*

**IDLitManipulatorContainer::OnKeyboard** - Is used when a keyboard event occurs on the target window (*Win*).

*Obj->[IDLitManipulator:]OnKeyboard, Win, IsASCII,  
Character, KeyVal, X, Y, Press, Release, KeyMods*

**IDLitManipulatorContainer::OnMouseDown** - Is used when a mouse down event occurs on the target window.

*Obj->[IDLitManipulatorContainer:]OnMouseDown, Win,  
X, Y, IButton, KeyMods, NClicks*

**IDLitManipulatorContainer::OnMouseMotion** - Manages the setting of the cursor on the window if no mouse button is down.

*Obj->[IDLitManipulatorContainer:]OnMouseMotion,  
Win, X, Y, KeyMods*

**IDLitManipulatorContainer::OnMouseUp** - Is used when a mouse up event occurs on the target window.

*Obj->[IDLitManipulatorContainer:]OnMouseUp, Win, X,  
Y, IButton*

**IDLitManipulatorContainer::OnWheel** - Handles notification (from the native window device) that a mouse wheel event has occurred.

*Obj->[IDLitManipulatorContainer:]OnWheel, X, Y, Delta,  
Modifiers*

**IDLitManipulatorContainer::SetCurrent** - Is used to set a manipulator within the container to be the current manipulator.

*Obj->[IDLitManipulatorContainer:]SetCurrent,  
Manipulator*

**IDLitManipulatorContainer::SetCurrentManipulator** - Is used to set the current child manipulator in a manipulator hierarchy.

*Obj->[IDLitManipulatorContainer:]  
SetCurrentManipulator, Identifier*

**IDLitManipulatorContainer::SetProperty** - Sets the value of an IDLitManipulatorContainer property

*Obj->[IDLitManipulator:]SetProperty  
[, PROPERTY=value]*

**IDLitManipulatorManager** - A specialization of the manipulator container (IDLitManipulatorContainer), which acts as the root of the manipulator hierarchy.

**IDLitManipulatorManager::Add** - Is used to add manipulators to the manipulator manager.

*Obj->[IDLitManipulatorManager:]Add, Manipulator  
[, /DEFAULT]*

**IDLitManipulatorManager::AddManipulatorObserver** - Is used to add an observer to the manipulator system.

*Obj->[IDLitManipulatorManager:]  
AddManipulatorObserver, Observer*

**IDLitManipulatorManager::GetDefaultManipulator** - Returns a reference to the manipulator that was most recently added as the default manipulator.

*Result = Obj->[IDLitManipulatorManager:]->  
GetDefaultManipulator()*

**IDLitManipulatorManager::Init** - Initializes the IDLitManipulatorManager object

*Obj = OBJ\_NEW('IDLitManipulatorManager'  
[, PROPERTY=value] or  
Result = Obj->[IDLitManipulatorManager:]Init(  
[PROPERTY=value])*

**IDLitManipulatorManager::RemoveManipulatorObserver** - Is used to remove a manipulator observer from the manipulator object.

*Obj->[IDLitManipulatorManager:]  
RemoveManipulatorObserver, Observer*

**IDLitManipulatorVisual** - The basis of all visual elements associated with an interactive manipulator.

**Properties:** [, /UNIFORM\_SCALE]  
[, VISUAL\_TYPE=string]

**IDLitManipulatorVisual::Cleanup** - Performs all cleanup on the object.

*OBJ\_DESTROY, Obj or  
Obj->[IDLitManipulatorVisual:]Cleanup*

**IDLitManipulatorVisual::GetProperty** - Retrieves the value of an IDLitManipulatorVisual property.

*Obj->[IDLitManipulatorVisual:]GetProperty*

**IDLitManipulatorVisual::Init** - Initializes the IDLitManipulatorVisual object.

*Obj = OBJ\_NEW('IDLitManipulatorVisual'  
[, PROPERTY=value] or  
Result = Obj->[IDLitManipulatorVisual:]Init(  
[, PROPERTY=value])*

**IDLitManipulatorVisual::SetProperty** - Sets the value of an IDLitManipulatorVisual property.

*Obj->[IDLitManipulatorVisual:]SetProperty  
[, PROPERTY=value]*

**IDLitOperation** - The basis for all iTool operations. It defines how an operation is executed and how information about the operation is recorded for the command transaction (undo-redo) system.

Properties: [, /EXPENSIVE\_COMPUTATION{Get, Init}]  
[, /REVERSIBLE\_OPERATION{Get, Init}]  
[, /SHOW\_EXECUTION\_UI{Get, Init}] [, TYPES{Get, Init}=string or string array]

**IDLitOperation::Cleanup** - Performs all cleanup on the object.

OBJ\_DESTROY, *Obj* or  
*Obj*->[IDLitOperation::]Cleanup

**IDLitOperation::DoAction** - Is called when an operation is requested by the iTools system, either as the result of a user action such as selection of a menu item or toolbar button, or by another operation.

*Result* = *Obj*->[IDLitOperation::]DoAction(*Tool*)

**IDLitOperation::GetProperty** - Retrieves the value of a property or group of properties of an operation object.

*Obj*->[IDLitOperation::]GetProperty  
[, *PROPERTY=variable*]

**IDLitOperation::Init** - Initializes the IDLitOperation object and sets properties that define the behavior the operation provides.

*Obj* = OBJ\_NEW('IDLitOperation'  
[, *PROPERTY=variable*]) or  
*Result* = *Obj*->[IDLitOperation::]Init(  
[, *PROPERTY=variable*])

**IDLitOperation::QueryAvailability** - Determines whether an operation is applicable for the selected data and/or visualization.

*Result* = *Obj*->[IDLitOperation::]QueryAvailability(*Tools*,  
*SelTypes*)

**IDLitOperation::RecordFinalValues** - Records the information needed to redo an operation.

*Result* =  
*Obj*->[IDLitOperation::]RecordFinalValues(*CommandSet*,  
*Targets* [, *IdProperty*])

**IDLitOperation::RecordInitialValues** - Records the information needed to undo an operation.

*Result* = *Obj*->[IDLitOperation::]RecordInitialValues(  
*CommandSet*, *Targets* [, *IdProperty*])

**IDLitOperation::RedoOperation** - Is called by the iTool system when the user requests the re-execution of an operation (usually by selecting Redo from the iTool Edit menu or toolbar).

*Result* = *Obj*->[IDLitOperation::]RedoOperation(  
*CommandSet*)

**IDLitOperation::SetProperty** - Sets the value of a property or group of properties for the operation.

*Obj*->[IDLitOperation::]SetProperty  
[, *PROPERTY=value*]

**IDLitOperation::UndoOperation** - Is called by the iTool system when the user requests the un-execution of an operation (usually by selecting Undo from the iTool Edit menu or toolbar).

*Result* = *Obj*->[IDLitOperation::]UndoOperation(  
*CommandSet*)

**IDLitParameter** - An interface providing parameter management methods to associate parameter names with IDLitData objects.

**IDLitParameter::Cleanup** - Removes the *data observer* from each data object in the visualization object's parameter set, and cleans up the objects and pointers defined to hold parameter data when the visualization object was created.

OBJ\_DESTROY, *Obj* or  
*Obj*->[IDLitParameter::]Cleanup()

**IDLitParameter::GetParameter** - Retrieves the IDLitData object associated with a registered parameter.

*Result* = *Obj*->[IDLitParameter::]GetParameter(*Name*  
[, /ALL] [, COUNT=*variable*])

**IDLitParameter::GetParameterAttribute** - Retrieves the value of an attribute associated with a registered parameter.

*Obj*->[IDLitParameter::]GetParameterAttribute,  
*ParamName* [, ATTRIBUTE=*variable*]

**IDLitParameter::GetParameterSet** - Returns a reference to the IDLitParameterSet object associated with the visualization object.

*Result* = *Obj*->[IDLitParameter::]GetParameterSet(  
[, /DEEP\_COPY])

**IDLitParameter::Init** - Initializes object instance fields that contain parameter data.

*Result* = *Obj*->[IDLitParameter::]Init()

**IDLitParameter::OnDataChangeUpdate** - Is called when a data value has been updated or a new data object has been associated with the visualization object.

*Obj*->[IDLitParameter::]OnDataChangeUpdate, *Data*,  
*ParameterName*

**IDLitParameter::OnDataDisconnect** - Is called when a data value has been disconnected from a parameter.

*Obj*->[IDLitParameter::]OnDataDisconnect,  
*ParameterName*

**IDLitParameter::QueryParameter** - Checks whether a parameter is registered, or retrieves a list of the of all registered parameters.

*Result* =  
*Obj*->[IDLitParameter::]QueryParameter(*ParamName*  
[, COUNT=*variable*])

**IDLitParameter::RegisterParameter** - Registers a parameter with the visualization object.

*Obj*->[IDLitParameter::]RegisterParameter, *Name*  
[, /BY\_VALUE] [, DESCRIPTION=*string*] [, /INPUT]  
[, /OPTARGET] [, /OPTIONAL] [, /OUTPUT]  
[, TYPES=*string*]

**IDLitParameter::SetData** - Is used to set data in this interface, associating a data object with a given parameter.

*Result* = *Obj*->[IDLitParameter::]SetData(*Data*  
[, /BY\_VALUE] [, PARAMETER\_NAME=*string*]  
[, /NO\_UPDATE])

**IDLitParameter::SetParameterAttribute** - Sets one or more parameter attributes for a registered parameter.

*Obj*->[IDLitParameter::]SetParameterAttribute,  
*ParamName* [, ATTRIBUTE=*variable*]

**IDLitParameterSet::SetParameterSet** - Is used to associate an IDLitParameterSet object with the visualization object's parameter interface.

*Result =*  
*Obj->*[IDLitParameterSet::]SetParameterSet(*ParamSet*)

**IDLitParameterSet** - A specialized subclass of the IDLitDataContainer class. This class provides the ability to associate names with contained IDLitData objects.

**IDLitParameterSet::Add** - Is used to add data to the parameter set.

*Obj->*[IDLitParameterSet::]Add, *Data*  
[, *PARAMETER\_NAME=string*]  
[, /PRESERVE\_LOCATION]

**IDLitParameterSet::Cleanup** - Performs all cleanup on the parameter set object.

OBJ\_DESTROY, *Obj* or  
*Obj->*[IDLitParameterSet::]Cleanup()

**IDLitParameterSet::Copy** - Returns a copy of the parameter set and its contents.

*Result = Obj->*[IDLitParameterSet::]Copy()

**IDLitParameterSet::Get** - Is used to retrieve one or more IDLitData objects from the parameter set.

*Result = Obj->*[IDLitParameterSet::]Get([, /ALL]  
[, *COUNT=variable*] [, *NAME=variable*]  
[, *POSITION=integer*])

**IDLitParameterSet::GetByName** - Returns the IDLitData object associated with a specified named parameter.

*Result = Obj->*[IDLitParameterSet::]GetByName(*Names*  
[, *COUNT=variable*] [, *NAME=variable*])

**IDLitParameterSet::GetParameterName** - Retrieves the name of a specified parameter using a provided data object.

*Result = Obj->*[IDLitParameterSet::]  
GetParameterName(*Data*, *Name*)

**IDLitParameterSet::Init** - Initializes the IDLitParameter object.

*Obj =* OBJ\_NEW('IDLitParameterSet') or  
*Result = Obj->*[IDLitParameterSet::]Init()

**IDLitParameterSet::Remove** - Is used to remove a data item from the parameter set.

*Obj->*[IDLitParameterSet::]Remove [, *Items*] [, /ALL]  
[, *POSITION=integer*]

**IDLitReader** - The definition of the interface and the process used to construct file readers for the iTools framework. When a new file reader is constructed for the iTools system, a new class is subclassed from this IDLitReader class.

**IDLitReader::Cleanup** - Performs all cleanup on the object, and should be called by the subclass' Cleanup method.

OBJ\_DESTROY, *Obj* or *Obj->*[IDLitReader::]Cleanup

**IDLitReader::GetData** - Is called by the system to retrieve the data from the current file.

*Result = Obj->*[IDLitReader::]GetData(*Data*)

**IDLitReader::GetFileExtensions** - Is called by the system to retrieve the file extensions supported by this particular reader.

*Result = Obj->*[IDLitReader::]GetFileExtensions(  
[*COUNT=variable*])

**IDLitReader::GetFilename** - Is called by the system to retrieve the current filename associated with this reader.

*Result = Obj->*[IDLitReader::]GetFilename()

**IDLitReader::GetProperty** - Retrieves the value of an IDLitReader property.

*Obj->*[IDLitReader::]GetProperty  
[, *PROPERTY=variable*]

**IDLitReader::Init** - Initializes the IDLitReader object.

*Obj =* OBJ\_NEW('IDLitReader', [, *Extensions*]  
[, *PROPERTY=value*]) or  
*Result = Obj->*[IDLitReader::]Init([, *Extensions*]  
[, *PROPERTY=value*])

**IDLitReader::IsA** - Is called by the system to determine if the given file is of the type supported by this file reader.

*Result = Obj->*[IDLitReader::]IsA(*Filename*)

**IDLitReader::SetFilename** - Is called by the system to set the current filename associated with this reader.

*Obj->*[IDLitReader::]SetFilename, *Filename*

**IDLitReader::SetProperty** - Sets the value of an IDLitReader property.

*Obj->*[IDLitReader::]SetProperty[, *PROPERTY=value*]

**IDLitTool** - All the functionality provided by a particular instance of an IDL Intelligent Tool (iTool). This object provides the management systems for the underlying tool functionality.

**Properties:** [, *DESCRIPTION*{Get, Init, Set}=string]  
[, *ICON*{Get, Init, Set}=string] [, *NAME*{Get, Init,  
Set}=string] [*TYPE*{Init, Set}=string or string array]  
[, /UPDATE\_BYTYPE{Init, Set}] [, /VERBOSE{Get,  
Init, Set}] [, *VERSION*{Get, Init}=variable]

**IDLitTool::ActivateManipulator** - Activates a manipulator that has been registered with this tool.

*Obj->*ActivateManipulator, *Identifier* [, /DEFAULT]

**IDLitTool::Add** - Adds any item to the tool.

*Obj->*[IDLitTool::]Add, *Item*

**IDLitTool::AddService** - Adds a service to the tool.

*Obj->*[IDLitTool::]AddService, *Service*

**IDLitTool::Cleanup** - Performs all cleanup on the object

OBJ\_DESTROY, *Obj* or *Obj->*[IDLitTool::]Cleanup

**IDLitTool::CommitActions** - Commits all pending transactions to the undo-redo buffer and causes a refresh of the current window.

*Obj->*[IDLitTool::]CommitActions

**IDLitTool::DisableUpdates** - Disables all drawing updates to the current window and UI updates (menu sensitivity) being passed to the user interface.

*Obj->*[IDLitTool::]DisableUpdates (  
[, *PREVIOUSLY\_DISABLED=variable*])

**IDLitTool::DoAction** - Initiates an operation or action in the tool object.

*Result = Obj->*[IDLitTool::]DoAction(*Identifier*)



**IDLitTool::Do SetProperty** - Sets a property on a target component object, and places the change in the undo/redo transaction buffer.

```
Result = Obj->[IDLitTool::]DoSetProperty(
    TargetIdentifier, PropertyIdentifier, Value)
```

**IDLitTool::DoUIService** - Initiates a request for a UI service to execute.

```
Result = Obj->[IDLitTool::]DoUIService(
    ServiceIdentifier, Requestor)
```

**IDLitTool::EnableUpdates** - Re-enables all drawing updates to the current window and UI updates (menu sensitivity) being passed to the user interface.

```
Obj->[IDLitTool::]EnableUpdates
```

**IDLitTool::FindIdentifiers** - Retrieve the full identifiers for items within the tool container

```
Result = Obj->IDLitTool::FindIdentifiers([Pattern]
    [, /ANNOTATIONS] [, /COUNT=variable]
    [, /DATA_MANAGER] [, /FILE_READERS]
    [, /FILE_WRITERS] [, /LEAF_NODES]
    [, /MANIPULATORS] [, /OPERATIONS]
    [, /VISUALIZATIONS])
```

**IDLitTool::GetCurrentManipulator** - Returns the current manipulator in the system.

```
Result = Obj->[IDLitTool::]GetCurrentManipulator()
```

**IDLitTool::GetFileReader** - Retrieves a file reader registered with the tool object.

```
Result = Obj->[IDLitTool::]GetFileReaders(Identifier
    [, /ALL] [, COUNT=variable])
```

**IDLitTool::GetFileWriter** - Retrieves a file writer registered with the tool object.

```
Result = Obj->[IDLitTool::]GetFileWriters(Identifier
    [, /ALL] [, COUNT=variable])
```

**IDLitTool::GetManipulators** - Retrieves the manipulators registered with the tool object.

```
Result = Obj->[IDLitTool::]GetManipulators(
    [COUNT=variable])
```

**IDLitTool::GetOperations** - Retrieves the operations registered with the tool object.

```
Result = Obj->[IDLitTool::]GetOperations(
    [, IDENTIFIER=string] [, COUNT=variable])
```

**IDLitTool::GetProperty** - Retrieves the value of an IDLitTool property.

```
Obj->[IDLitTool::]GetProperty, [PROPERTY=variable]
```

**IDLitTool::GetSelectedItems** - Returns a vector of references to the objects currently selected within the current window in the tool.

```
Result = Obj->[IDLitTool::]GetSelectedItems(
    [COUNT=variable])
```

**IDLitTool::GetService** - Retrieves a service that has been registered with the tool.

```
Result = Obj->[IDLitTool::]GetService(IdService)
```

**IDLitTool::GetVisualization** - Retrieves a visualization registered with the tool object.

```
Result = Obj->[IDLitTool::]GetVisualizations(Identifier
    [, /ALL] [, COUNT=variable])
```

**IDLitTool::Init** - Initializes the IDLitTool object

```
Obj = OBJ_NEW('IDLitTool' [, PROPERTY=value]) or
Result = Obj->[IDLitTool::]Init([PROPERTY=value])
```

**IDLitTool::RefreshCurrentWindow** - Redraws the current window of the tool.

```
Obj->[IDLitTool::]RefreshCurrentWindow
```

**IDLitTool::Register** - Registers a generic component with the tool.

```
Obj->[IDLitTool::]Register, Name, ClassName
    [, /DEFAULT] [, DESCRIPTION=string]
    [, ICON=string] [, IDENTIFIER=string]
    [, PROXY=string]
```

**IDLitTool::RegisterCustomization** - Registers an operation class that represents the graphics customization operation to be associated with this tool.

```
Obj->[IDLitTool::]RegisterCustomization, Name,
    ClassName
```

**IDLitTool::RegisterFileReader** - Registers a file reader component with the tool.

```
Obj->[IDLitTool::]RegisterFileReader, Name, ClassName
    [, /DEFAULT][, DESCRIPTION=string] [, ICON=string]
    [, IDENTIFIER=string] [, PROXY=string]
```

**IDLitTool::RegisterFileWriter** - Registers a file writer component with the tool.

```
Obj->[IDLitTool::]RegisterFileWriter, Name, ClassName
    [, /DEFAULT][, DESCRIPTION=string]
    [, ICON=string] [, IDENTIFIER=string]
    [, PROXY=string]
```

**IDLitTool::RegisterManipulator** - Registers a manipulator component with the tool.

```
Obj->[IDLitTool::]Manipulator, Name, ClassName
    [, /DEFAULT] [, DESCRIPTION=string]
    [, ICON=string] [, IDENTIFIER=string]
```

**IDLitTool::RegisterOperation** - Registers an operation component with the tool.

```
Obj->[IDLitTool::]RegisterOperation, Name, ClassName
    [, ACCELERATOR=string] [, /CHECKED]
    [, DESCRIPTION=string] [, /DROPLIST_EDIT]
    [, DROPLIST_INDEX=value]
    [, DROPLIST_ITEMS=string array] [, ICON=string]
    [, IDENTIFIER=string] [, PROXY=string]
    [, /SEPARATOR]
```

**IDLitTool::RegisterStatusBarSegment** - Registers a status message bar segment with the tool.

```
Obj->[IDLitTool::]RegisterStatusBarSegment, Name,
    [, IDENTIFIER=string] [, NORMALIZED_WIDTH=scalar]
```

**IDLitTool::RegisterVisualization** - Registers a visualization component with the tool.

*Obj*->[IDLitTool::]RegisterVisualization, *Name*,  
*ClassName* [, /DEFAULT] [, DESCRIPTION=*string*]  
 [, ICON=*string*] [, IDENTIFIER=*string*]  
 [, PROXY=*string*]

**IDLitTool::SetProperty** - Sets the value of an IDLitTool property

*Obj*->[IDLitTool::]SetProperty[, *PROPERTY=**value*]

**IDLitTool::UnRegister** - Unregisters a component with the tool.

*Obj*->[IDLitTool::]UnRegister, *Identifier*

**IDLitTool::UnRegisterCustomization** - Unregisters an operation class (that was previously registered as the graphics customization operation to be associated with this tool).

*Obj*->[IDLitTool::]UnRegisterCustomization

**IDLitTool::UnRegisterFileReader** - Unregisters a file reader component with the tool.

*Obj*->[IDLitTool::]UnRegisterFileReader, *Identifier*

**IDLitTool::UnRegisterFileWriter** - Unregisters a component with the tool.

*Obj*->[IDLitTool::]UnRegisterFileWriter, *Identifier*

**IDLitTool::UnRegisterManipulator** - Unregisters a manipulator component with the tool.

*Obj*->[IDLitTool::]UnRegisterManipulator, *Identifier*

**IDLitTool::UnRegisterOperation** - Unregisters an operation component with the tool.

*Obj*->[IDLitTool::]UnRegisterOperation, *Identifier*

**IDLitTool::UnRegisterStatusBarSegment** - Unregisters a status message bar segment.

*Obj*->[IDLitTool::]UnRegisterStatusBarSegment,  
*Identifier*

**IDLitTool::UnRegisterVisualization** - Unregisters a visualization component with the tool.

*Obj*->[IDLitTool::]UnRegisterVisualization, *Identifier*

**IDLitUI** - A link between the underlying functionality of an iTool and the IDL widget interface.

**Property:** [, GROUP\_LEADER=*WidgetID*]

**IDLitUI::AddOnNotifyObserver** - Is used to register a specified iTool component object as wishing to receive messages generated by the DoOnNotify method of another specified iTool component object.

*Obj*->[IDLitUI::]AddOnNotifyObserver, *IdObserver*,  
*IdSubject*

**IDLitUI::Cleanup** - Performs all cleanup on the object

OBJ\_DESTROY, *Obj* or *Obj*->[IDLitUI::]Cleanup

**IDLitUI::DoAction** - Initiates an operation or action in the tool object associated with this user interface.

*Result* = *Obj*->[IDLitUI::]doAction(*Identifier*)

**IDLitUI::GetProperty** - Retrieves the value of an IDLitUI property.

*Obj*->[IDLitUI::]GetProperty, [*PROPERTY=**variable*]

**IDLitUI::GetTool** - Returns an object reference to the iTool with which the user interface is associated.

*Result* = *Obj*->[IDLitUI::]GetTool()

**IDLitUI::GetWidgetByName** - returns the IDL Widget ID of a widget that has been registered with the user interface object via a call to the IDLitUI::RegisterWidget method.

*Result* = *Obj*->[IDLitUI::]GetWidgetByName(*Name*)

**IDLitUI::Init** - Initializes the IDLitUI object

*Obj* = OBJ\_NEW('IDLitUI', *oTool*) or  
*Result* = *Obj*->[IDLitUI::]Init(*oTool*)

**IDLitUI::RegisterUIService** - Registers a user interface service with the user interface.

*Result* = *Obj*->[IDLitUI::]RegisterUIService(*Name*,  
*Callback*)

**IDLitUI::RegisterWidget** - Registers an IDL widget hierarchy with the user interface object.

*Result* = *Obj*->[IDLitUI::]RegisterWidget(*wID*, *Name*,  
*Callback*[, /FLOATING])

**IDLitUI::RemoveOnNotifyObserver** - Is used to un-register a specified iTool component object as wishing to receive messages generated by the DoOnNotify method of another specified iTool component object.

*Obj*->[IDLitUI::]RemoveOnNotifyObserver, *IdObserver*,  
*IdSubject*

**IDLitUI::SetProperty** - Sets the value of an IDLitUI property.

*Obj*->[IDLitUI::]SetProperty[, *PROPERTY=**value*]

**IDLitUI::UnRegisterUIService** - Unregisters a user interface service with the user interface object.

*Obj*->[IDLitUI::]UnRegisterUIService, *Name*

**IDLitUI::UnRegisterWidget** - Unregisters a widget with the user interface object.

*Obj*->[IDLitUI::]UnRegisterWidget, *Name*

**IDLitVisualization** - The basis for all iTool visualizations. All visualization components subclass from this class.

**Properties:** [, CENTER\_OF\_ROTATION{Get, Init,  
 Set}=[, *x*, *y*] or [, *x*, *y*, *z*]] [, /IMPACTS\_RANGE{Get, Init,  
 Set}] [, ISOTROPIC{Get, Init, Set}]  
 [, /MANIPULATOR\_TARGET{Get, Init, Set}]  
 [, /PROPERTY\_INTERSECTION{Get, Init}]  
 [, TYPE{Init, Set}=*string* or *string* array]

**IDLitVisualization::Add** - Adds objects to the visualization container.

*Obj*->[IDLitVisualization::]Add, *Objects*  
 [, /AGGREGATE] [, /NO\_UPDATE]  
 [, POSITION=*index*]

**IDLitVisualization::Aggregate** - Adds the given object(s) to this visualization's property aggregate.

*Obj*->[IDLitVisualization::]Aggregate, *Objects*

**IDLitVisualization::BeginManipulation** - Handles notifications that an IDLitManipulator object is about to manipulate this visualization.

*Obj*->[IDLitVisualization::]BeginManipulation,  
*Manipulator*

**IDLitVisualization::Cleanup** - Performs all cleanup on the object OBJ\_DESTROY, *Obj* or

*Obj*->[IDLitVisualization::]Cleanup

**IDLitVisualization::EndManipulation** - Handles notifications that an IDLitManipulator object has finished manipulating this visualization.

*Obj*->[IDLitVisualization::]EndManipulation,  
*Manipulator*

**IDLitVisualization::Get** - Retrieves object(s) from the visualization.

*Result* = *Obj*->[IDLitVisualization::]Get([, /ALL]  
[, COUNT=*variable*] [, ISA=*string* or *array of strings*]  
[, POSITION=*index* or *array of indices*]  
[, /SKIP\_PRIVATE])

**IDLitVisualization::GetCenterRotation** - Returns the center of rotation for this visualization.

*Result* = *Obj*->[IDLitVisualization::]GetCenterRotation  
([, /NO\_TRANSFORM] [, XRANGE=[*xmin*, *xmax*]]  
[, YRANGE=[*ymin*, *ymax*]] [, ZRANGE=[*zmin*, *zmax*]])

**IDLitVisualization::GetCurrentSelectionVisual** - Returns the currently active selection visual object for this visualization.

*Result* = *Obj*->[IDLitVisualization::]  
GetCurrentSelectionVisual()

**IDLitVisualization::GetDataSpace** - Returns a reference to the parent data space object within the graphics hierarchy that contains the visualization.

*Result* = *Obj*->[IDLitVisualization::]GetDataSpace(  
[, /UNNORMALIZED])

**IDLitVisualization::GetDataString** - Retrieves a description of this visualization's data at the given x, y, and z location.

*Result* = *Obj*->[IDLitVisualization::]GetDataString(  
*XYZLocation*)

**IDLitVisualization::GetDefaultSelectionVisual** - Returns an object that serves as the default selection visual for this visualization.

*Result* = *Obj*->[IDLitVisualization::]  
GetDefaultSelectionVisual()

**IDLitVisualization::GetManipulatorTarget** - Retrieves the manipulator target associated with this visualization.

*Result* = *Obj*->[IDLitVisualization::]  
GetManipulatorTarget()

**IDLitVisualization::GetProperty** - Retrieves the value of a property or group of properties for the object.

*Obj*->[IDLitVisualization::]GetProperty  
[, *PROPERTY=variable*]

**IDLitVisualization::GetRequestedAxesStyle** - Returns the axes style requested by this visualization.

*Result* = *Obj*->[IDLitVisualization::]  
GetRequestedAxesStyle()

**IDLitVisualization::GetSelectionVisual** - Retrieves the selection visual for this visualization that corresponds to the given manipulator.

*Result* = *Obj*->[IDLitVisualization::]GetSelectionVisual(  
*Manipulator*)

**IDLitVisualization::GetTypes** - Identifies the types that this visualization represents, including base types and any specializations.

*Result* = *Obj*->[IDLitVisualization::]GetTypes()

**IDLitVisualization::GetXYZRange** - Computes the x, y, and z ranges of the overall contents of the visualization, taking into account the IMPACTS\_RANGE property setting for itself and its contents.

*Result* = *Obj*->[IDLitVisualization::]GetXYZRange(  
*XRange*, *YRange*, *ZRange* [, /DATA]  
[, /NO\_TRANSFORM])

**IDLitVisualization::Init** - Initializes the visualization object.

*Obj* = OBJ\_NEW('IDLitVisualization'  
[, *PROPERTY=value*] ) or  
*Result* = *Obj*->[IDLitVisualization::]Init(  
[*PROPERTY=value*])

**IDLitVisualization::Is3D** - Determines whether or not this visualization (or any of its contents) is marked as being three-dimensional.

*Result* = *Obj*->[IDLitVisualization::]Is3D()

**IDLitVisualization::IsIsotropic** - Indicates whether or not this visualization (or any of its contents) is marked as being isotropic.

*Result* = *Obj*->[IDLitVisualization::]IsIsotropic()

**IDLitVisualization::IsManipulatorTarget** - Determines whether or not this visualization is a manipulator target.

*Result* = *Obj*->[IDLitVisualization::]  
IsManipulatorTarget()

**IDLitVisualization::IsSelected** - Determines if this visualization is currently selected or not.

*Result* = *Obj*->[IDLitVisualization::]IsSelected()

**IDLitVisualization::Move** - Changes the location of an object within the visualization container.

*Obj*->[IDLitVisualization::]Move, *Source*, *Destination*

**IDLitVisualization::On2DRotate** - Ensures that objects that use the visualization's data are notified when the rotation of the parent dataspace changes.

*Obj*->[IDLitVisualization::]On2DRotate, *Notifier*,  
*IsRotated*

**IDLitVisualization::OnAxesRequestChange** - Ensures that objects that use the visualization's data are notified when the axes of a contained object are changed.

*Obj*->[IDLitVisualization::]OnAxesRequestChange,  
*Notifier*, *AxesRequest*

**IDLitVisualization::OnAxesStyleRequestChange -**

Ensures that objects that use the visualization's data are notified when the axes style of a contained object is changed.

*Obj->*[IDLitVisualization::]OnAxesStyleRequestChange, *Notifier, StyleRequest*

**IDLitVisualization::OnDataChange -** Ensures that objects that use the visualization's data are notified when the visualization's data changes.

*Obj->*[IDLitVisualization::]OnDataChange, *Notifier*

**IDLitVisualization::OnDataComplete -** Ensures that objects that use the visualization's data are notified when changes to the visualization's data are complete.

*Obj->*[IDLitVisualization::]OnDataComplete, *Notifier*

**IDLitVisualization::OnDataRangeChange -** Ensures that objects that use the visualization's data are notified when the range of the visualization's data changes.

*Obj->*[IDLitVisualization::]OnDataRangeChange, *Notifier, XRange, YRange, Zrange*

**IDLitVisualization::OnDimensionChange -** Ensures that objects that use the visualization's data are notified when the visualization's dimensionality changes.

*Obj->*[IDLitVisualization::]OnDimensionChange, *Notifier, Is3D*

**IDLitVisualization::OnWorldDimensionChange -** Ensures that objects that use the visualization's data are notified when the visualization's parent dataspace's dimensionality changes.

*Obj->*[IDLitVisualization::]OnWorldDimensionChange, *Notifier, Is3D*

**IDLitVisualization::Remove -** Removes the given object(s) from the visualization.

*Obj->*[IDLitVisualization::]Remove, *Object* [, /NO\_UPDATE]

**IDLitVisualization::RequestsAxes -** Indicates whether or not the visualization requests axes. Returns a 1 if the visualization does request axes, or 0 if does not request axes.

*Result = Obj->*[IDLitVisualization::]RequestsAxes( )

**IDLitVisualization::Restore -** Performs any transitional work required after an object has been restored from a SAVE file.

*Obj->*[IDLitVisualization::]Restore

**IDLitVisualization::Rotate -** Rotates this visualization about the given axis by the given angle.

*Obj->*[IDLitVisualization::]Rotate, *Axis, Angle* [, CENTER\_OF\_ROTATION=[x, y, z]] [, /PREMULTIPLY]

**IDLitVisualization::Scale -** Scales the visualization by the given scale factors.

*Obj->*[IDLitVisualization::]Scale, *SX, SY, SZ* [, CENTER\_OF\_ROTATION=[x, y] | [x, y, z]] [, /PREMULTIPLY]

**IDLitVisualization::Select -** Handles notification of mechanisms that key off the current selection (such as the visualization browser) when this visualization has been selected.

*Obj->*[IDLitVisualization::]Select[, *Mode*] [, /ADDITIVE | /SELECT | /TOGGLE | /UNSELECT] [, /NO\_NOTIFY]

**DLitVisualization::Set3D -** Sets a flag indicating this visualization is three-dimensional.

*Obj->*[IDLitVisualization::]Set3D, *Is3D* [, /ALWAYS] [, /AUTO\_COMPUTE]

**IDLitVisualization::SetAxesRequest -** Sets the current axes request for this visualization.

*Obj->*[IDLitVisualization::]SetAxesRequest, *AxesRequest* [, /ALWAYS | /AUTO\_COMPUTE] [, /NO\_NOTIFY]

**IDLitVisualization::SetAxesStyleRequest -** Sets the current axes style request for this visualization.

*Obj->*[IDLitVisualization::]SetAxesStyleRequest, *StyleRequest* [, /NO\_NOTIFY]

**IDLitVisualization::SetCurrentSelectionVisual -** Sets the current selection visual for the given manipulator.

*Obj->*[IDLitVisualization::]SetCurrentSelectionVisual, *Manipulator*

**IDLitVisualization::SetData -** Sets the data parameter of the visualization.

*Result = Obj->*[IDLitVisualization::]SetData(*Data*)

**IDLitVisualization::SetDefaultSelectionVisual -** Sets the default selection visual to be associated with this visualization.

*Obj->*[IDLitVisualization::]SetDefaultSelectionVisual, *SelectionVisual* [, POSITION=*value*]

**IDLitVisualization::SetParameterSet -** Associates a parameter set with this visualization.

*Result = Obj->*[IDLitVisualization::]SetParameterSet(*ParameterSet*)

**IDLitVisualization::SetProperty -** Sets the value of a property or group of properties for the object.

*Obj->*[IDLitVisualization::]SetProperty [, *PROPERTY=value*]

**IDLitVisualization::UpdateSelectionVisual -** Transforms this visualization's selection visual to match the visualization's geometry.

*Obj->*[IDLitVisualization::]UpdateSelectionVisual

**IDLitVisualization::VisToWindow -** Transforms given points from visualization data space to window device coordinates.

*Obj->*[IDLitVisualization::]VisToWindow, *InX, InY, InZ, OutX, OutY, OutZ* [, /NO\_TRANSFORM] or *Obj->*[IDLitVisualization::] VisToWindow, *InX, InY, OutX, OutY* [, /NO\_TRANSFORM] or *Obj->*[IDLitVisualization::] VisToWindow, *InVerts, OutVerts* [, /NO\_TRANSFORM]

**IDLitVisualization::WindowToVis** - Transforms given points from window device coordinates to visualization data space.

*Obj*->[IDLitVisualization:]WindowToVis, *InX*, *InY*, *InZ*, *OutX*, *OutY*, *OutZ* or *Obj*->[IDLitVisualization:]WindowToVis, *InX*, *InY*, *OutX*, *OutY* or *Obj*->[IDLitVisualization:]WindowToVis, *InVerts*, *OutVerts*

**IDLitWindow** - The basis for all iTool visualization windows. All iTool visualization windows subclass from this class.

**Properties:** IDLitWindow inherits all properties from the IDLgrWindow superclass. See “IDLgrWindow” on page 105 for details.

**IDLitWindow::Add** - Adds the given object(s) to the window.

*Obj*->[IDLitWindow:]Add, *Objects* [, POSITION=*value*]

**IDLitWindow::AddWindowEventObserver** - Adds the given object(s) to the list of observers that are to be notified of events that occur within this window.

*Obj*->[IDLitWindow:]AddWindowEventObserver, *Objects*

**IDLitWindow::Cleanup** - Performs all cleanup on the object.

OBJ\_DESTROY, *Obj* or *Obj*->[IDLitWindow:]Cleanup

**IDLitWindow::ClearSelections** - Clears the window’s list of currently selected items (within its current view).

*Obj*->[IDLitWindow:]ClearSelections

**IDLitWindow::DoHitTest** - Performs a hit test to determine which visualizations within the destination are displayed at a given pixel location.

*Result* = *Obj*->DoHitTest(*X*, *Y* [, DIMENSIONS=[*width*, *height*]] [, /ORDER] [, SUB\_HIT=*variable*] [, UNITS={0 | 1 | 2 | 3})

**IDLitWindow::GetEventMask** - Returns a bitwise mask representing the events that are enabled for this window.

*Result* = *Obj*->[IDLitWindow:]GetEventMask([, BUTTON\_EVENTS=*variable*] [, KEYBOARD\_EVENTS=*variable*] [, /MOTION\_EVENTS=*variable*] [, TIMER\_EVENTS=*variable*] [, /TRACKING\_EVENTS=*variable*] [, /WHEEL\_EVENTS=*variable*])

**IDLitWindow::GetProperty** - Retrieves the value of an IDLitWindow property

*Obj*->[IDLitWindow:]GetProperty[, *PROPERTY*=*variable*]

**IDLitWindow::GetSelectedItems** - Returns the currently selected objects within this window’s scene, which represents a container for all of the views (and their corresponding visualization hierarchies) that appear within a window.

*Result* = *Obj*->[IDLitWindow:]GetSelectedItems([, /ALL] [, COUNT=*named variable*])

**IDLitWindow::Init** - Initializes the window object.

*Obj* = OBJ\_NEW('IDLitWindow'[, *PROPERTY*=*value*]) or *Result* = *Obj*->[IDLitWindow:]Init([, *PROPERTY*=*value*])

**IDLitWindow::OnEnter** - Handles notification from the window that a mouse enter event has occurred.

*Obj*->[IDLitWindow:]OnEnter

**IDLitWindow::OnExit** - Handles notification from the window that a mouse exit event has occurred.

*Obj*->[IDLitWindow:]OnExit

**IDLitWindow::OnExpose** - Handles notification from the window that an expose event has occurred.

*Obj*->[IDLitWindow:]OnExpose, *X*, *Y*, *Width*, *Height*

**IDLitWindow::OnKeyboard** - Handles notification (from the native window device) that a keyboard event has occurred, and passes along that notification to all observers in the list of window event observers.

*Obj*->[IDLitWindow:]OnKeyboard, *IsASCII*, *Character*, *KeySymbol*, *X*, *Y*, *Press*, *Release*, *Modifiers*

**IDLitWindow::OnMouseDown** - Handles notification (from the native window device) that a mouse down event has occurred, and passes along that notification to all observers in the list of window event observers.

*Obj*->[IDLitWindow:]OnMouseDown, *X*, *Y*, *ButtonMask*, *Modifiers*, *NumClicks*

**IDLitWindow::OnMouseMotion** - Handles notification (from the native window device) that a mouse motion event has occurred, and passes along that notification to all observers in the list of window event observers.

*Obj*->[IDLitWindow:]OnMouseMotion, *X*, *Y*, *Modifiers*

**IDLitWindow::OnMouseUp** - Handles notification (from the native window device) that a mouse up event has occurred, and passes along that notification to all observers in the list of window event observers.

*Obj*->[IDLitWindow:]OnMouseUp, *X*, *Y*, *ButtonMask*

**IDLitWindow::OnResize** - Handles notification (from the native window device) that a resize event has occurred.

*Obj*->[IDLitWindow:]OnResize, *Width*, *Height*

**IDLitWindow::OnScroll** - Handles notification (from the native window device) that a scrolling event has occurred.

*Obj*->[IDLitWindow:]OnScroll, *X*, *Y*

**IDLitWindow::OnTimer** - handles notification (from the native window device) that a timer event has occurred, and passes that notification to all observers in the list of window event observers by calling each observer’s OnTimer method.

*Obj*->[IDLitWindow:]OnTimer

**IDLitWindow::OnWheel** - Handles notification (from the native window device) that a mouse wheel event has occurred.

*Obj*->[IDLitWindow:]OnWheel, *X*, *Y*, *Delta*, *Modifiers*

**IDLitWindow::Remove** - Removes the given object(s) from the window.

*Obj*->[IDLitWindow:]Remove, *Object* [, /ALL] [, POSITION=*index*]

**IDLitWindow::RemoveWindowEventObserver** - Removes the given object(s) from the list of observers that are notified of events that occur within this window.

*Obj*->[IDLitWindow:]RemoveWindowEventObserver, *Objects*

**IDLitWindow::SetCurrentZoom** - Sets the current zoom factor for this window by changing its virtual dimensions.

*Obj*->[IDLitWindow:]SetCurrentZoom, *ZoomFactor*  
[, /RESET]

**IDLitWindow::SetEventMask** - Enables the given events within this window.

*Obj*->[IDLitWindow:]SetEventMask([*EventMask*]  
[, /BUTTON\_EVENTS] [, /KEYBOARD\_EVENTS]  
[, /MOTION\_EVENTS] [, /TIMER\_EVENTS]  
[, /TRACKING\_EVENTS] [, /WHEEL\_EVENTS])

**IDLitWindow::SetManipulatorManager** - Sets the given IDLManipulatorManager object as the current manager of this window's manipulators.

*Obj*->[IDLitWindow:]SetManipulatorManager, *Manager*

**IDLitWindow::SetProperty** - Sets the value of an IDLitWindow property.

*Obj*->[IDLitWindow:]SetProperty[, *PROPERTY=value*]

**IDLitWindow::SetTimerInterval** - Specifies the floating-point number of seconds between timer events.

*Obj*->[IDLitWindow:]SetProperty[, *PROPERTY=value*]

**IDLitWindow::ZoomIn** - Causes the current zoom factor for this window to be increased (that is, multiplied by the factor given by the window's ZOOM\_BASE property).

*Obj*->[IDLitWindow:]ZoomIn

**IDLitWindow::ZoomOut** - Causes the current zoom factor for this window to be decreased (that is, divided by the factor given by the window's ZOOM\_BASE property).

*Obj*->[IDLitWindow:]ZoomOut

**IDLitWriter** - The definition of the interface and the process used to construct file writers for the iTools framework. When a new file writer is constructed for the iTools system, a new class is subclassed from this IDLitWriter class.

**IDLitWriter::Cleanup** - Performs all cleanup on the object

OBJ\_DESTROY, *Obj* or *Obj*->[IDLitWriter:]Cleanup

**IDLitWriter::GetFileExtensions** - Is called by the system to retrieve the file extensions supported by this particular writer.

*Result* = *Obj*->[IDLitWriter:]GetFileExtensions(  
[COUNT=*count*])

**IDLitWriter::GetFilename** - Is called by the system to retrieve the current filename associated with this writer.

*Result* = *Obj*->[IDLitWriter:]GetFilename()

**IDLitWriter::GetProperty** - Retrieves the value of an IDLitWriter property.

*Obj*->[IDLitWriter:]GetProperty [, *PROPERTY=variable*]

**IDLitWriter::Init** - Initializes the IDLitWriter object

*Obj* = OBJ\_NEW('IDLitWriter', [, *Extensions*]  
[, *PROPERTY=value*] or  
*Result* = *Obj*->[IDLitWriter:]Init([, *Extensions*]  
[, *PROPERTY=value*])

**IDLitWriter::IsA** - Is called by the system to determine if the given file is of the type supported by this file writer.

*Result* = *Obj*->[IDLitWriter:]IsA(*Filename*)

**IDLitWriter::SetData** - Is called by the system to set the data for the current file.

*Result* = *Obj*->[IDLitWriter:]SetData(*Data*)

**IDLitWriter::SetFilename** - Is called by the system to set the current filename associated with this writer.

*Obj*->[IDLitWriter:]SetFilename, *Filename*

**IDLitWriter::SetProperty** - Sets the value of an IDLitWriter property

*Obj*->[IDLitWriter:]SetProperty[, *PROPERTY=value*]

**IDLjavaObject** - An IDL object encapsulating a Java object. IDL provides data type and other translation services, allowing IDL programs to access the Java object's methods and properties using standard IDL syntax.

**IDLjavaObject::GetProperty** - Retrieves properties (known as data members in Java) from the Java object that underlies the IDLjavaObject.

*Obj*->[IDLjavaObject:]GetProperty  
[, *PROPERTY=variable*]

**IDLjavaObject::Init** - Instantiates the given Java object and establishes a link between the resulting IDL object with the underlying Java object.

*Obj* = OBJ\_NEW('IDLjavaObject\$JAVACLASSNAME',  
*JavaClassName* [, *Arg1*, ...]) or  
*Result* = *Obj*->[IDLjavaObject\$JAVACLASSNAME:]Init(  
*JavaClassName* [, *Arg1*, ...])

**IDLjavaObject::SetProperty** - Sets properties (known as data members in Java) for the Java object that underlies an instance of IDLjavaObject.

*Obj*->[IDLjavaObject:]SetProperty [, *PROPERTY=value*]

**IDLnetOGCWCS** - Provides the ability to connect to a remote Open Geospatial Consortium Web Coverage Service server and return coverage offering briefs and unrendered data containing geographic data.

**Properties:** [, AUTHENTICATION {Get, Init, Set}=*value*] [, CALLBACK\_DATA {Get, Init, Set}=*variable*] [, CALLBACK\_FUNCTION {Get, Init, Set}=*string*] [, CAPABILITIES\_FILENAME {Get, Init, Set}=*string*] [, CONNECT\_TIMEOUT {Get, Init, Set}=*value*] [, CONTENT\_TYPE {Get}=*string*] [, COVERAGE\_FILENAME {Get, Init, Set}=*string*] [, DESCRIBE\_COVERAGE\_FILENAME {Get, Init, Set}=*string*] [, ENCODE {Get, Init, Set}=*value*] [, HEADERS {Get, Init, Set}=*string*] [, LAST\_FILE {Get, Init, Set}=*string*] [, PASSWORD {Init, Set}=*string*] [, PROXY\_AUTHENTICATION {Get, Init, Set}=*value*] [, PROXY\_HOSTNAME {Get, Init, Set}=*string*] [, PROXY\_PASSWORD {Init, Set}=*string*] [, PROXY\_PORT {Get, Init, Set}=*string*] [, PROXY\_USERNAME {Get, Init, Set}=*string*] [, RESPONSE\_HEADER {Get}=*string*] [, SSL\_CERTIFICATE\_FILE {Get, Init, Set}=*string*] [, SSL\_VERIFY\_HOST {Get, Init, Set}=*Boolean*] [, SSL\_VERIFY\_PEER {Get, Init, Set}=*Boolean*] [, TIMEOUT {Get, Init, Set}=*value*] [, URL\_HOSTNAME {Get, Init, Set}=*string*]

[, URL\_PATH {Get, Init, Set}=string] [, URL\_PORT {Get, Init, Set}=string] [, URL\_QUERY\_PREFIX {Get, Init, Set}=string] [, URL\_QUERY\_SUFFIX {Get, Init, Set}=string] [, URL\_SCHEME {Get, Init, Set}=string] [, USERNAME {Get, Init, Set}=string] [, VERBOSE {Get, Init, Set}=value] [, WCS\_VERSION {Get, Init, Set}=string]

**IDLnetOGCWCS::Cleanup** - Performs all cleanup on the object.  
OBJ\_DESTROY, *Obj*  
or  
*Obj*->[IDLnetOGCWCS::]Cleanup (Only in subclass' Cleanup method)

**IDLnetOGCWCS::DescribeCoverage** - Retrieves an XML file containing one or more descriptive coverage offerings from a remote OGC WCS server. Return value contains names of coverage names.

*Result* = *Obj*->[IDLnetOGCWCS::]DescribeCoverage  
([, COUNT=variable] [, FROM\_FILE=value] ,  
NAMES=value [, SCHEMA\_CHECK=value])

**IDLnetOGCWCS::GetCapabilities** - Retrieves an XML file containing coverage offering briefs from a remote OGC WCS server.

*Result* = *Obj*->[IDLnetOGCWCS::]GetCapabilities(  
[, FROM\_FILE=value] [, SCHEMA\_CHECK=value])

**IDLnetOGCWCS::GetCoverage** - Retrieves a data file containing unrendered data from a remote OGC WCS server.

*Result* = *Obj*->[IDLnetOGCWCS::]GetCoverage  
(*Coverage\_Request*)

**IDLnetOGCWCS::GetCoverageOffering** - Returns a complex structure containing details about a single coverage.

*Result* = *Obj*->[IDLnetOGCWCS::]GetCoverageOffering  
([ INDEX=value])

**IDLnetOGCWCS::GetCoverageOfferingBriefs** - Provides access to coverage offering briefs returned by GetCapabilities.

*Result* = *Obj*->[IDLnetOGCWCS::]GetCoverageOffering  
([ COUNT=variable] [, NUMBER=value]  
[, INDEX=value])

**IDLnetOGCWCS::GetProperty** - Retrieves the value of a property or group of properties.

*Obj*->[IDLnetOGCWCS::]GetProperty  
[, PROPERTY=variable]]

**IDLnetOGCWCS::GetServiceSection** - Provides a structure that accesses Service section values contained in the XML file returned by GetCapabilities.

*Result* = *Obj*->[IDLnetOGCWCS::]GetServiceSection()

**IDLnetOGCWCS::Init** - Initializes an OGC WCS object.

*Obj* =  
OBJ\_NEW(IDLnetOGCWCS [PROPERTY=value])

or  
*Result* = *Obj*->  
[ IDLnetOGCWCS::]Init([PROPERTY=value]) (Only in a subclass' Init method)

**IDLnetOGCWCS::ParseUrl** - Given a valid URL, populated required properties with portions of the URL string.

*Obj*->[IDLnetOGCWCS::]ParseUrl, *URL*

**IDLnetOGCWCS::SetProperty** - Sets the value of a property or group of properties for the object.

*Obj*-> [ IDLnetOGCWCS::]SetProperty  
[, PROPERTY=value]

**IDLnetOGCWMS** - Provides the ability to connect to a remote Open Geospatial Consortium Web Map Service server and return one or more layers as a rendered image.

**Properties:** [, AUTHENTICATION {Get, Init, Set}=value] [, CALLBACK\_DATA {Get, Init, Set}=variable] [, CALLBACK\_FUNCTION {Get, Init, Set}=string] [, CAPABILITIES\_FILENAME {Get, Init, Set}=string] [, CONNECT\_TIMEOUT {Get, Init, Set}=value] [, CONTENT\_TYPE {Get}=string] [, ENCODE {Get, Init, Set}=value] [, FEATURE\_INFO\_FILENAME {Get, Init, Set}=string] [, HEADERS {Get, Init, Set}=string] [, LAST\_FILE {Get, Init, Set}=string] [, MAP\_FILENAME {Get, Init, Set}=string] [, PASSWORD {Init, Set}=string] [, PROXY\_AUTHENTICATION {Get, Init, Set}=value] [, PROXY\_HOSTNAME {Get, Init, Set}=string] [, PROXY\_PASSWORD {Init, Set}=string] [, PROXY\_PORT {Get, Init, Set}=string] [, PROXY\_USERNAME {Get, Init, Set}=string] [, RESPONSE\_HEADER {Get}=string] [, SSL\_CERTIFICATE\_FILE {Get, Init, Set}=string] [, SSL\_VERIFY\_HOST {Get, Init, Set}=Boolean] [, SSL\_VERIFY\_PEER {Get, Init, Set}=Boolean] [, TIMEOUT {Get, Init, Set}=value] [, URL\_HOSTNAME {Get, Init, Set}=string] [, URL\_PATH {Get, Init, Set}=string] [, URL\_PORT {Get, Init, Set}=string] [, URL\_QUERY\_PREFIX {Get, Init, Set}=string] [, URL\_QUERY\_SUFFIX {Get, Init, Set}=string] [, URL\_SCHEME {Get, Init, Set}=string] [, USERNAME {Get, Init, Set}=string] [, VERBOSE {Get, Init, Set}=value] [, WMS\_VERSION {Get, Init, Set}=string]

**IDLnetOGCWMS::Cleanup** - Performs all cleanup on the object.

OBJ\_DESTROY, *Obj*

or

*Obj*->[IDLnetOGCWMS::]Cleanup (Only in subclass' Cleanup method)

**IDLnetOGCWMS::GetCapabilities** - Retrieves an XML file containing layer information from a remote OGC WCS server.

*Result* = *Obj*->[IDLnetOGCWMS::]GetCapabilities  
( [ /DISABLE\_INHERITANCE] [, FROM\_FILE=value]  
[, SCHEMA\_CHECK=value])

**IDLnetOGCWMS::GetFeatureInfo** - Retrieves a information about a feature on a map if supported by the WMS server and if the information exists.

*Result = Obj->[IDLnetOGCWMS::]GetFeatureInfo  
(Feature\_Info\_Request)*

**IDLnetOGCWCS::GetLayers** - Provides access to layer details returned by GetCapabilities.

*Result = Obj->[IDLnetOGCWCS::]GetLayers  
( [ COUNT=variable] [, NUMBER=value]  
[, INDEX=value])*

**IDLnetOGCWCS::GetMap** - Retrieves one or more layers as a rendered map image.

*Result = Obj->[IDLnetOGCWCS::]GetMap  
(Map\_Request)*

**IDLnetOGCWMS::GetProperty** - Retrieves the value of a property or group of properties.

*Obj->[IDLnetOGCWMS::]GetProperty  
[, PROPERTY=variable]]*

**IDLnetOGCWMS::GetServiceSection** - Provides a structure that accesses Service section values contained in the XML file returned by GetCapabilities.

*Result = Obj->[IDLnetOGCWMS::]GetServiceSection()*

**IDLnetOGCWMS::Init** - Initializes an OGC WMS object.

*Obj =  
OBJ\_NEW(IDLnetOGCWMS [PROPERTY=value])  
or  
Result = Obj->  
[ IDLnetOGCWMS::]Init([PROPERTY=value]) (Only in  
a subclass' Init method)*

**IDLnetOGCWMS::ParseUrl** - Given a valid URL, populated required properties with portions of the URL string.

*Obj->[IDLnetOGCWMS::]ParseUrl, URL*

**IDLnetOGCWMS::SetProperty** - Sets the value of a property or group of properties for the object.

*Obj-> [ IDLnetOGCWMS::]SetProperty  
[, PROPERTY=value]*

**IDLnetURL** - Allows IDL to act as a client to an HTTP or FTP server. Use this object to open a connection, create and send HTTP requests, and receive responses from a remote HTTP or FTP server. You can also use an IDLnetURL object to receive and write large data files to disk at a user-specified location.

**Properties:** [, AUTHENTICATION {Get, Init, Set}=value] [, CALLBACK\_DATA {Get, Init, Set}=variable] [, CALLBACK\_FUNCTION {Get, Init, Set}=string] [, CONNECT\_TIMEOUT {Get, Init, Set}=value] [, CONTENT\_TYPE {Get}=string] [, ENCODE {Get, Init, Set}=value] [, FTP\_CONNECTION\_MODE {Get, Init, Set}=value] [, HEADERS {Get, Init, Set}=string] [, PROXY\_AUTHENTICATION {Get, Init, Set}=value] [, PROXY\_HOSTNAME {Get, Init, Set}=string] [, PROXY\_PASSWORD { Init, Set}=string] [, PROXY\_PORT {Get, Init, Set}=string]

[, PROXY\_USERNAME {Get, Init, Set}=string]  
[, RESPONSE\_CODE {Get}=value]  
[, RESPONSE\_FILENAME {Get}=string]  
[, RESPONSE\_HEADER {Get}=string]  
[, SSL\_CERTIFICATE\_FILE {Get, Init, Set}=string]  
[, SSL\_VERIFY\_HOST {Get, Init, Set}=value]  
[, SSL\_VERIFY\_PEER {Get, Init, Set}=value]  
[, SSL\_VERSION {Get, Init, Set}=value] [, TIMEOUT {Get, Init, Set}=value] [, URL\_HOSTNAME {Get, Init, Set}=string] [, URL\_PASSWORD {Init, Set}=string] [, URL\_PATH {Get, Init, Set}=string] [, URL\_PORT {Get, Init, Set}=string] [, URL\_QUERY {Get, Init, Set}=string] [, URL\_SCHEME {Get, Init, Set}=string] [, URL\_USERNAME {Get, Init, Set}=string] [, VERBOSE {Get, Init, Set}=boolean]

**IDLnetURL::Cleanup** - Performs all cleanup operations on the object, and closes all open connections when the object is destroyed.

*OBJ\_DESTROY, Obj*

or

*Obj->[IDLnetURL::]Cleanup (In a lifecycle method only.)*

**IDLnetURL::CloseConnections** - Closes all the existing connections to remote HTTP and FTP servers.

*Obj->[IDLnetURL::]CloseConnections*

**IDLnetURL::FtpCommand** - Executes FTP commands, as well as deletes files, makes directories, deletes directories, changes directories, and so on, on the remote FTP server.

*Result = Obj->[IDLnetURL::]FtpCommand( command,  
[, URL=string] [, /FTP\_ASCII]  
[, /FTP\_EXPLICIT\_SSL])*

**IDLnetURL::Get** - Retrieves a resource from a remote HTTP or FTP server and writes it to disk, a memory buffer, or an array of strings.

*Result = Obj->[IDLnetURL::]Get( [, URL=string]  
[, FILENAME=string] [, /BUFFER]  
[, /STRING\_ARRAY] [, /FTP\_ASCII]  
[, /FTP\_EXPLICIT\_SSL])*

**IDLnetURL::GetFtpDirList** - Retrieves a list of files (returned as an array of strings) available from a remote FTP server on a particular path.

*Result = Obj->[IDLnetURL::]GetFtpDirList(  
[, URL=string] [, /VERBOSE]  
[, /FTP\_EXPLICIT\_SSL])*

**IDLnetURL::GetProperty** - Returns information describing the properties of an IDLnetURL object in the supplied variable/key-word pairs.

*Obj->[IDLnetURL::]GetProperty [, PROPERTYNAME =  
value]*

**IDLnetURL::Init** - Initializes an IDLnetURL object.

*obj =OBJ\_NEW('IDLnetURL')*

or

*Obj->[IDLnetURL::]Init (In a lifecycle method only.)*



**IDLnetURL::Put** - Sends a file or a buffer of data to a remote HTTP or FTP server.

*Result = Obj->*[IDLnetURL::] Put ( *data*, [, URL=*string*]  
[, RESPONSE\_FILENAME=*filename*] [, /FTP\_ASCII]  
[, /FTP\_EXPLICIT\_SSL] [, /BUFFER] [, /POST])

**IDLnetURL::SetProperty** - Sets the properties of an object that is open for writing.

*Obj->*[IDLnetURL::] SetProperty [, PROPERTYNAME =  
value]

**IDLsysMonitorInfo** - Contains information about the display monitor or monitors attached to your system. Use this object to obtain information about the monitor(s) when building applications that utilize multiple monitors.

**Properties:** [, DISPLAY\_NAME {Get, Init, Set}=*string*]

**IDLsysMonitorInfo::Cleanup** - Performs all cleanup on the object.

OBJ\_DESTROY, *Obj*  
or  
*Obj->*[IDLsysMonitorInfo::]Cleanup (Only in subclass'  
Cleanup method)

**IDLsysMonitorInfo::GetMonitorNames** - Retrieves into an array the X Window System display name strings associated with each monitor on the system. There is one name string in the array for each monitor.

*Result =*  
*Obj->*[IDLsysMonitorInfo::]GetMonitorNames()

**IDLsysMonitorInfo::GetNumberOfMonitors** - Retrieves the number of monitors available on the system as an IDL integer.

*Result =*  
*Obj->*[IDLsysMonitorInfo::]GetNumberOfMonitors()

**IDLsysMonitorInfo::GetPrimaryMonitorIndex** - Retrieves the index of the monitor that is considered the system's primary monitor as an IDL integer.

*Result =*  
*Obj->*[IDLsysMonitorInfo::] GetPrimaryMonitorIndex()

**IDLsysMonitorInfo::GetProperty** - Retrieves the value of a property or group of properties.

*Obj->*[IDLsysMonitorInfo::]GetProperty  
[, PROPERTY=*variable*]

**IDLsysMonitorInfo::GetRectangles** - Retrieves monitor pixel coordinates and dimensions as a [4, n] IDL long array. There is one row in the array for each monitor.

*Result = Obj->*[IDLsysMonitorInfo::]GetRectangles(  
[, /EXCLUDE\_TASKBAR])

**IDLsysMonitorInfo::GetResolutions** - Retrieves monitor resolutions (pixel sizes) as a [2, n] IDL float array of x and y cm/pixel.

*Result = Obj->*[IDLsysMonitorInfo::]GetResolutions()

**IDLsysMonitorInfo::Init** - Initializes a monitor information object.

*Obj =*  
OBJ\_NEW(IDLsysMonitorInfo [PROPERTY=*value*])  
or  
*Result = Obj->*  
[ IDLsysMonitorInfo::]Init([PROPERTY=*value*]) (Only  
in a subclass' Init method)

**IDLsysMonitorInfo::IsExtendedDesktop** - Returns a Boolean value (0 or 1 as an IDL integer) that indicates if the display system is configured as an extended desktop across two or more monitors.

*Result =*  
*Obj->*[IDLsysMonitorInfo::]IsExtendedDesktop()

**IDLsysMonitorInfo::SetProperty** - Sets the value of a property or group of properties for the object.

*Obj->*  
[ IDLsysMonitorInfo::]SetProperty[, PROPERTY=*value*]

**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).

**Properties:** [, AXIS{Init}={0 | 1 | 2}]  
[, /CONSTRAIN{Init}] [, MOUSE{Init}={1 | 2 | 4}]

**TrackBall::Init** - Initializes the TrackBall object.

*Obj = OBJ\_NEW*('TrackBall', *Center*, *Radius*  
[, PROPERTY=*value*]) or  
*Result = Obj->*[TrackBall::]Init(*Center*, *Radius*  
[, PROPERTY=*value*])

**TrackBall::Reset** - Resets the state of the TrackBall object.

*Obj->*[TrackBall::]Reset, *Center*, *Radius* [, AXIS={0 | 1 |  
2}] [, /CONSTRAIN] [, MOUSE={1 | 2 | 4}]

**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={1  
| 2 | 4}] [, TRANSFORM=*variable*] [, /TRANSLATE] )

# Statements

## Assignment

---

**variable = expression** - Assigns a value to a variable.

**variable[subscripts] = expression** - Assigns a value to the elements of an array specified by the array subscripts.

**variable[subscript\_range] = expression** - Assigns a value to the elements of an array specified by the array subscript range.

## Program Control

---

### Compound Statements

---

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

```
BEGIN
  statements
END | ENDIF | ENDELSE | ENDFOR | ENDREP |
ENDWHILE
```

### Conditional Statements

---

**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 ]
```

**CASE** - Selects one statement for execution from multiple choices, depending on the value of an expression.

```
CASE expression OF
  expression: statement
...
  expression: statement
[ ELSE: statement ]
ENDCASE
```

**SWITCH** - Selects one statement for execution from multiple choices, depending upon the value of an expression.

```
SWITCH expression OF
  expression: statement
...
  expression: statement
[ELSE: statement ]
ENDSWITCH
```

## Loop Statements

---

**FOR...DO** - 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
```

**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
```

## Jump Statements

---

**BREAK** - Immediately exits from a loop (FOR, WHILE, REPEAT), CASE, or SWITCH statement without resorting to GOTO statements.

```
BREAK
```

**CONTINUE** - Immediately starts the next iteration of the enclosing FOR, WHILE, or REPEAT loop.

```
CONTINUE
```

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

```
GOTO, label
```

## Functions and Procedures

---

**COMPILE\_OPT** - Gives IDL compiler information that changes the default rules for compiling functions or procedures.

COMPILE\_OPT *opt*<sub>1</sub> [, *opt*<sub>2</sub>, ..., *opt*<sub>*n*</sub>]

**Note:** *opt*<sub>*n*</sub> can be IDL2, DEFINT32, HIDDEN, LOGICAL\_PREDICATE, OBSOLETE, STRICTARR, or STRICTARRSUBS

**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*<sub>1</sub>, *Name*<sub>2</sub>, ..., *Name*<sub>*n*</sub>

**FUNCTION** - Defines a function.

FUNCTION *Function\_Name*, *parameter*<sub>1</sub>, ..., *parameter*<sub>*n*</sub>

**PRO** - Defines a procedure.

PRO *Procedure\_Name*, *argument*<sub>1</sub>, ..., *argument*<sub>*n*</sub>

**Procedure\_Name** - Calls a procedure.

*Procedure\_Name*, *argument*<sub>1</sub>, ..., *argument*<sub>*n*</sub>

**Result = FUNCTION( *arg*<sub>1</sub>, ..., *arg*<sub>*n*</sub> )** - Calls a function.

## Variable Scope

---

**COMMON** - Creates a common block.

COMMON *Block\_Name*, *Variable*<sub>1</sub>, ..., *Variable*<sub>*n*</sub>

# Executive Commands

Executive commands must be entered at the IDL command prompt. They cannot be used in programs.

**.COMPILE** - Compiles programs without running.

`.COMPILE [File1,..., Filen]`

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 IDL Workbench (Windows and Motif only). Note that filenames are separated by spaces, not commas.

`.EDIT File1 [File2 Filen]`

**.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 [File1,...,Filen]`

To save listing in a file: `.RNEW -L ListFile.lis File1  
[, File2,..., Filen]`

To display listing on screen: `.RNEW -T File1 [, File2,...,  
Filen]`

**.RUN** - Compiles and executes IDL commands from files or keyboard.

`.RUN [File1,..., Filen]`

To save listing in a file: `.RUN -L ListFile.lis File1`

`[, File2,..., Filen]`

To display listing on screen: `.RUN -T File1 [, File2,...,  
Filen]`

**.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

The following table lists the characters that have a special meaning in IDL:

Character	Description
<b>Ampersand (&amp;)</b>	<ul style="list-style-type: none"> <li>•Separates multiple commands on a single line</li> </ul>
<b>Apostrophe (')</b>	<ul style="list-style-type: none"> <li>•Delimits string constants</li> <li>•Indicates part of octal or hex constant</li> </ul>
<b>Asterisk (*)</b>	<ul style="list-style-type: none"> <li>•Multiplication operator</li> <li>•Array subscript range</li> <li>•Pointer dereference (if in front of a valid pointer)</li> </ul>
<b>At Sign (@)</b>	<ul style="list-style-type: none"> <li>•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.</li> <li>•In interactive mode, @ is used to execute a batch file.</li> </ul>
<b>Colon (:)</b>	<ul style="list-style-type: none"> <li>•Ends label identifiers</li> <li>•Separates start and end subscript ranges</li> </ul>
<b>Dollar Sign (\$)</b>	<ul style="list-style-type: none"> <li>•Continue current command on the next line</li> <li>•Issue operating system command if entered on a line by itself</li> </ul>
<b>Exclamation Point (!)</b>	<ul style="list-style-type: none"> <li>•First character of system variable names and font-positioning commands</li> </ul>
<b>Period (.)</b>	<ul style="list-style-type: none"> <li>•First character of executive commands</li> <li>•Indicates floating-point numbers</li> <li>•Indicates fields in a structure, such as in mystructure.field1</li> </ul>
<b>Question Mark (?)</b>	<ul style="list-style-type: none"> <li>•Invokes online help when entered at the IDL command line</li> <li>•Part of the ?: ternary operator used in conditional expressions</li> </ul>
<b>Semicolon (;)</b>	<ul style="list-style-type: none"> <li>•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</li> </ul>

# 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.

Example	Description
<b>Vector</b> [i]	Element $i + 1$ of a vector. <code>Vector[12]</code> denotes the value of the 13th element of <code>Vector</code> .
<b>Array</b> [i, j]	The element stored at column $i$ , row $j$ of an array.
<b>Vector</b> [i:j]	Elements $i$ through $j$ of a vector.
<b>Vector</b> [i:*]	Elements from $i$ through the end of a vector.
<b>Array</b> [i, *]	Column $i$ of a two-dimensional array.
<b>Array</b> [*, j]	The $j$ th row of a two-dimensional array.
<b>Array</b> [i:j, m:n]	Subarray of columns $i$ through $j$ , rows $m$ through $n$ .
<b>Array</b> [i:j:k]	A range of subscripts, written $[k_0:k_1:k_2]$ , denoting every $k_2$ th element within the range of subscripts $k_0$ through $k_1$ ( $k_0$ must not be greater than $k_1$ ). $k_2$ is referred to as the subscript <i>stride</i> .
<b>Array</b> [i:*:k]	Every $k_2$ th element from a given element to the last element of the dimension, written as $[k_0:*:k_2]$ . $k_2$ is referred to as the subscript <i>stride</i> .
<b>Array</b> [Array2]	The elements of <code>Array</code> whose subscripts are the values of <code>Array2</code> .
<b>(Array_Expression)</b> [i]	Element $i$ of an array-valued expression.

# Operators

## Mathematical Operators

---

**+** Addition, String Concatenation  
**-** Subtraction and Negation  
**\*** Multiplication, Pointer dereference  
**/** Division  
**^** Exponentiation  
**++** Increment  
**--** Decrement  
**MOD** Modulo

## Minimum/Maximum Operators

---

**<** The Minimum Operator  
**>** The Maximum Operator

## Matrix Operators

---

**# and ##** Matrix Multiplication

## Logical Operators

---

**&&** - Logical AND  
**||** Logical OR  
**~** Logical NOT  
**AND** Bitwise AND  
**NOT** Bitwise complement  
**OR** Bitwise OR  
**XOR** Bitwise 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  
**op=** Compound assignment  
**?:** Conditional expression  
**->** Object method invocation

## 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)
	[] (brackets, to concatenate arrays)
Second	. (structure field dereference)
	[] (brackets, to subscript an array)
	() (parentheses, used in a function call)
Third	* (pointer dereference)
	^ (exponentiation)
	++ (increment)
	-- (decrement)
Fourth	* (multiplication)
	# and ## (matrix multiplication)
	/ (division)
	MOD (modulus)
Fifth	+ (addition)
	- (subtraction and negation)
	< (minimum)
	> (maximum)
	NOT (bitwise negation)
	~ (logical negation)
Sixth	EQ (equality)
	NE (not equal)
	LE (less than or equal)
	LT (less than)
	GE (greater than or equal)
	GT (greater than)
Seventh	AND (bitwise AND)
	OR (bitwise OR)
	XOR (bitwise exclusive OR)
Eighth	&& (logical AND)
	(logical OR)
Ninth	? : (conditional 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,  $\pi/180 \approx 0.01745$ .

**!MAP** - Read-only system variable used by MAP\_SET.

**!PI** - Single-precision pi (p).

**!RADEG** - Radians to degrees,  $180/\pi \approx 57.2958$ .

**!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 134)

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 ( $x_0, y_0, x_1, y_1$ )

PSYM - plotting symbol index (see “[Plotting Symbols](#)” on page 134)

REGION - normalized coords of plot region ( $x_0, y_0, x_1, y_1$ )

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 134)

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

**!X, !Y, !Z** - *continued*

**TICKINTERVAL** - indicates the interval between major tick marks for the first axis level

**TICKLAYOUT** - indicates the tick layout style to be used to draw each level of the axis

**TICKLEN** - tick mark length, in normal coords

**TICKNAME** - annotation for each tick (string array)

**TICKS** - number of major tick intervals

**TICKUNITS** - indicates the units to be used for axis tick labeling

**TICKV** - data values for each tick mark (array)

**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)

**!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

## 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.

**SYS\_CODE\_TYPE** - A string describing the type of system code contained in **SYS\_CODE**.

**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).

## IDL Environment System Variables

---

**!CPU** - Read-only variable that supplies information about the state of the system processor, and of IDL's use of it.

**Fields:** **HW\_VECTOR** - True (1) if the system supports a vector unit (e.g. Macintosh Altivec/Velocity Engine) or False (0) otherwise.

**VECTOR\_ENABLE** - True (1) if IDL will use a vector unit, if such a unit is available on the current system, and False (0) otherwise.

**HW\_NCPU** - The number of CPUs contained in the system on which IDL is currently running.

**TPOOL\_NTHREADS** - The number of threads that IDL will use in thread pool computations.

**TPOOL\_MIN\_ELTS** - The number of elements in a computation that are necessary before IDL will use the thread pool to perform the work

**TPOOL\_MAX\_ELTS** - The maximum number of elements in a computation for which IDL will use the thread pool.

**!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 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.

**!MAKE\_DLL** - Used to configure how IDL uses the **CALL\_EXTERNAL**, **DLMs**, and **LINKIMAGE** for the current platform.

**!PATH** - Search path for IDL routines.

UNIX: colon-separated list of directories.

Windows: semicolon-separated list of directories.

**!PROMPT** - String to be used for IDL prompt.

**!QUIET** - Suppresses informational messages if set to nonzero.

**!VERSION** - Type, architecture, and version of IDL.

**Fields:** **ARCH** - CPU hardware architecture of the system.

**OS** - The name of the underlying operating system kernel (e.g. sunos, Win32).

**OS\_FAMILY** - The generic name of the operating system (e.g. UNIX, Windows).

**OS\_NAME** - The vendor's name for the operating environment (e.g. Solaris, Microsoft Windows).

**RELEASE** - The IDL version number.

**BUILD\_DATE** - Date the IDL executable was compiled.

# Graphics Information

## Direct Graphics Devices

---

**CGM** - The CGM Device  
**HP** - The HP-GL 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.  
**{XYZ}TICKINTERVAL** - Set to indicate the interval between major tick marks for the first axis level.  
**{XYZ}TICKLAYOUT** - Set to indicate the tick layout style to be used to draw each level of the axes.  
**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}TICKUNITS** - Set to indicate the units to be used for axis tick labeling.  
**{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 `O PLOT`, `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 `O PLOT`, `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 <code>USERSYM</code> procedure.
9	Undefined
10	Histogram mode.





# Scientific Data Formats

This quick reference guide contains an alphabetical listing of all scientific data format routines including CDF\*, EOS\* and HDF\* routines. The alphabetical listing contains all functions, procedures, and statements including the syntax of each.

- [“CDF Routines”](#) on page 136
- [“EOS Routines”](#) on page 137
- [“HDF Routines”](#) on page 142
- [“HDF5 Routines”](#) on page 147
- [“NetCDF Routines”](#) on page 151

## 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\_ATT EXISTS** - Determines whether specified attribute exists.

*Result* = CDF\_ATT EXISTS( *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\_NEGTOPOSP0\_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\_NEGTOPOSP0\_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] [, /BMRS\_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]  
[, /BMRS\_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\_ENCODE\_EPOCH16** - Encodes CDF\_EPOCH16 variable into a string.  
*Result* = CDF\_ENCODE\_EPOCH16(*Epoch16*  
[, 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\_EPOCH\_COMPARE** - Compares two epoch (date and time) values and returns an integer value of 1, 0, or -1.  
*Result* = CDF\_EPOCH\_COMPARE(*epoch1, epoch2*)
- CDF\_EPOCH\_DIFF** - Compares two epoch (date and time) values and returns the difference in milliseconds (default) or microseconds.  
*Result* = CDF\_EPOCH\_DIFF(*epoch1, epoch2*  
[, /MILLI\_SECONDS] [, /MICRO\_SECONDS])
- CDF\_EPOCH16** - Computes/breaks down CDF\_EPOCH16 values.  
CDF\_EPOCH16, *Epoch, Year* [, *Month, Day, Hour, Minute, Second, Milli, Micro, Nano, Pico*]  
[, /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\_PARSE\_EPOCH16** - Parses input string into a double precision value properly formatted for use as CDF\_EPOCH16 variable.  
*Result* = CDF\_PARSE\_EPOCH16(*Epoch\_string*)
- CDF\_SET\_CDF27\_BACKWARD\_COMPATIBLE** - Allows users of IDL version 6.3 and later to create a CDF file that can be read by IDL 6.2 or earlier  
CDF\_SET\_CDF27\_BACKWARD\_COMPATIBLE  
[, /YES | /NO ]
- 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* [, /ZVar] )
- 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\_DEFPXREG** - Defines whether the pixel center or pixel corner is used when requesting the location of a given pixel.

*Result = EOS\_GD\_DEFPXREG(gridID, pxreg)*

**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*, *zonocode*, *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 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 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( filename [, /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, filename, 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\_DEVRTREGION** - Subsets along any dimension.

*Result* = EOS\_SW\_DEVRTREGION(*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 given 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*, *n**type*, *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*, *n**type*, *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 ID of annotation with given 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 44.

**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, 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 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* )
- HDF\_HDF2IDLTYPE** - Converts HDF data type code into IDL variable type code.  
*Result* = HDF\_HDF2IDLTYPE( *hdfstypecode* )
- HDF\_IDL2HDFTYPE** - Converts IDL variable type code into HDF data type code.  
*Result* = HDF\_IDL2HDFTYPE( *idlstypecode* )
- 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 44.
- 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\_CHAR8] [, /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\_ATTRFIND** - Returns an attribute's index number given the name of an attribute.

*Result* = HDF\_VD\_ATTRFIND(*VData*, *FieldID*, *Name*)

**HDF\_VD\_ATTRINFO** - Retrieves information about a VData attribute.

HDF\_VD\_ATTRINFO, *VData*, *FieldID*, *AttrID*, *Values*  
[, *COUNT=variable*] [, *DATA=variable*]  
[, *HDF\_TYPE=variable*] [, *NAME=variable* ]  
[, *TYPE=variable*]

**HDF\_VD\_ATTRSET** - Writes a vdata attribute or a vdata field attribute to the currently attached HDF VData structure.

HDF\_VD\_ATTRSET, *VData*, *FieldID*, *Attr\_Name*, *Values*  
[, *Count*] [, */BYTE*] [, */DFNT\_CHAR8*]  
[, */DFNT\_FLOAT32*] [, */DFNT\_FLOAT64*]  
[, */DFNT\_INT8*] [, */DFNT\_INT16*] [, */DFNT\_INT32*]  
[, */DFNT\_UCHAR8*] [, */DFNT\_UINT8*]  
[, */DFNT\_UINT16*] [, */DFNT\_UINT32*] [, */DOUBLE*]  
[, */FLOAT*] [, */INT*] [, */LONG*] [, */SHORT*] [, */STRING*]  
[, */UINT*] [, */ULONG* ]

**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\_INSERT** - Adds VData or VGroup to contents of VGroup.

HDF\_VD\_INSERT, *VGroup*, *VData(or*  
*Vgroup)* [, *POSITION=variable*]

**HDF\_VD\_ISATTR** - Returns True (1) if the VData is storing an attribute, False (0) otherwise.

*Result* = HDF\_VD\_ISATTR(*VData*)

**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 VData's that are not contained in another VData.

*Result* = HDF\_VD\_LONE( *FileHandle*  
[, *MAXSIZE=value*] )

**HDF\_VD\_NATTRS** - Returns the number of attributes associated with the specified VData.

*Result* = HDF\_VD\_NATTRS( *VData*, *FieldID* )

**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]
  [, REF=variable] [, TAG=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 [, CLASSNAME=string]
  [, NAME=string]
```

## HDF5 Routines

**H5\_BROWSER** - Presents a graphical user interface for viewing and reading HDF5 files.

```
Result = H5_BROWSER([Files] [, /DIALOG_READ] )
```

**H5\_CLOSE** - Flushes all data to disk, closes file identifiers, and cleans up memory.

```
H5_CLOSE
```

**H5\_CREATE** - Creates and closes a new HDF5 file.

```
H5_CREATE, Filename, Structure
```

**H5\_GET\_LIBVERSION** - Returns the current version of the HDF5 library used by IDL.

```
Result = H5_GET_LIBVERSION( )
```

**H5\_OPEN** - Initializes IDL's HDF5 library.

```
H5_OPEN
```

**H5\_PARSE** - Recursively descends through an HDF5 file or group and creates an IDL structure containing object information and data.

```
Result = H5_PARSE (File [, /READ_DATA])
```

or

```
Result = H5_PARSE (Loc_id, Name [, FILE=string]
  [, PATH=string] [, /READ_DATA])
```

**H5A\_CLOSE** - Closes the specified attribute and releases resources used by it.

```
H5A_CLOSE, Attribute_id
```

**H5A\_CREATE** - Creates a dataset as an attribute of another group or dataset.

```
Result = H5A_CREATE(Loc_id, Name, Datatype_id,
  Dataspace_id)
```

**H5A\_DELETE** - Removes the attribute specified by its name from a dataset, group, or named datatype.

```
H5A_DELETE, Loc_id, Name
```

**H5A\_GET\_NAME** - Retrieves an attribute name given the attribute identifier number.

```
Result = H5A_GET_NAME(Attribute_id)
```

**H5A\_GET\_NUM\_ATTRS** - Returns the number of attributes attached to a group, dataset, or a named datatype.

```
Result = H5A_GET_NUM_ATTRS(Loc_id)
```

**H5A\_GET\_SPACE** - Returns the identifier number of a copy of the dataspace for an attribute.

*Result* = H5A\_GET\_SPACE(*Attribute\_id*)

**H5A\_GET\_TYPE** - Returns the identifier number of a copy of the datatype for an attribute.

*Result* = H5A\_GET\_TYPE(*Attribute\_id*)

**H5A\_OPEN\_IDX** - Opens an existing attribute by the index of that attribute.

*Result* = H5A\_OPEN\_IDX(*Loc\_id*, *Index*)

**H5A\_OPEN\_NAME** - Opens an existing attribute by the name of that attribute.

*Result* = H5A\_OPEN\_NAME(*Loc\_id*, *Name*)

**H5A\_READ** - Reads the data within an attribute, converting from the HDF5 file datatype into the HDF5 memory datatype, and finally into the corresponding IDL datatype.

*Result* = H5A\_READ(*Attribute\_id* [, *Datatype\_id*])

**H5A\_WRITE** - Writes data to an attribute.

H5A\_WRITE, *Attribute\_id*, *Data*

**H5D\_CLOSE** - Closes the specified dataset and releases its used resources.

H5D\_CLOSE, *Dataset\_id*

**H5D\_CREATE** - Creates a dataset at the specified location.

*Result* = H5D\_CREATE(*Loc\_id*, *Name*, *Datatype\_id*,  
*Dataspace\_id* [, CHUNK\_DIMENSIONS=*vector*  
[, GZIP=*value* [, /SHUFFLE]])

**H5D\_EXTEND** - Changes the current dimensions of the Dataset, within the limits of the Dataspace.

H5D\_EXTEND, *Dataset\_id*, *Size*

**H5D\_GET\_SPACE** - Returns an identifier number for a copy of the dataspace for a dataset.

*Result* = H5D\_GET\_SPACE(*Dataset\_id*)

**H5D\_GET\_STORAGE\_SIZE** - Returns the amount of storage in bytes required for a dataset.

*Result* = H5D\_GET\_STORAGE\_SIZE(*Dataset\_id*)

**H5D\_GET\_TYPE** - Returns an identifier number for a copy of the datatype for a dataset.

*Result* = H5D\_GET\_TYPE(*Dataset\_id*)

**H5D\_OPEN** - Opens an existing dataset within an HDF5 file.

*Result* = H5D\_OPEN(*Loc\_id*, *Name*)

**H5D\_READ** - Reads the data within a dataset, converting from the HDF5 file datatype into the HDF5 memory datatype, and finally into the corresponding IDL datatype.

*Result* = H5D\_READ(*Dataset\_id* [, *Datatype\_id*]  
[, FILE\_SPACE=*id*] [, MEMORY\_SPACE=*id*])

**H5D\_WRITE** - Writes data to a dataset.

H5D\_WRITE, *Dataset\_id*, *Data*  
[, MEMORY\_SPACE\_ID=*value*]  
[, FILE\_SPACE\_ID=*value*]

**H5F\_CLOSE** - Closes the specified file and releases resources used by it.

H5F\_CLOSE, *File\_id*

**H5F\_CREATE** - The primary function for creating HDF5 files.

*Result* = H5F\_CREATE(*Filename*)

**H5F\_IS\_HDF5** - Determines if a file is in the HDF5 format.

*Result* = H5F\_IS\_HDF5(*Filename*)

**H5F\_OPEN** - Opens an existing HDF5 file.

*Result* = H5F\_OPEN(*Filename*)([, /WRITE])

**H5G\_CLOSE** - Closes the specified group and releases resources used by it.

H5G\_CLOSE, *Group\_id*

**H5G\_CREATE** - Creates a new empty group and gives it a name.

*Result* = H5G\_CREATE(*Loc\_id*, *Name*)

**H5G\_GET\_COMMENT** - Retrieves a comment string from a specified object.

*Result* = H5G\_GET\_COMMENT(*Loc\_id*, *Name*)

**H5G\_GET\_LINKVAL** - Returns the name of the object pointed to by a symbolic link.

*Result* = H5G\_GET\_LINKVAL(*Loc\_id*, *Name*)

**H5G\_GET\_MEMBER\_NAME** - Retrieves the name of an object within a group, by its zero-based index.

*Result* = H5G\_GET\_MEMBER\_NAME(*Loc\_id*, *Name*,  
*Index*)

**H5G\_GET\_NMEMBERS** - Returns the number of objects within a group.

*Result* = H5G\_GET\_NMEMBERS(*Loc\_id*, *Name*)

**H5G\_GET\_NUM\_OBJS** - Returns number of objects in the group specified by its identifier.

*Result* = H5G\_GET\_NUM\_OBJS(*Loc\_id*)

**H5G\_GET\_OBJ\_NAME\_BY\_IDX** - Returns a name of an object specified by an index.

*Result* = H5G\_GET\_OBJ\_NAME\_BY\_IDX(*Loc\_id*,  
*Index*)

**H5G\_GET\_OBJINFO** - Retrieves information from a specified object.

*Result* = H5G\_GET\_OBJINFO(*Loc\_id*, *Name*  
[, /FOLLOW\_LINK])

**H5G\_LINK** - Creates a link of the specified type. A link can only point to one of the three classes of named objects: group, dataset, and named datatype.

H5G\_LINK, *Loc\_id*, *Current\_Name*, *New\_Name*  
[, /SOFTLINK] [, NEW\_LOC\_ID=*value*]

**H5G\_MOVE** - Renames/moves an object within an HDF5 group or file.

H5G\_MOVE, *Loc\_id*, *Src\_Name*, *Dst\_Name*  
[, NEW\_LOC\_ID=*value*]

**H5G\_OPEN** - Opens an existing group within an HDF5 file.

*Result* = H5G\_OPEN(*Loc\_id*, *Name*)

**H5G\_SET\_COMMENT** - Sets a comment for a specified object.

H5G\_SET\_COMMENT, *Loc\_id*, *Name*, *Comment*

**H5G\_UNLINK** - Removes the link to an object from a group.

H5G\_UNLINK, *Loc\_id*, *Name*

**H5I\_GET\_FILE\_ID** - Retrieves an identifier for the file containing the specified object.

*Result* = H5I\_GET\_FILE\_ID(*Loc\_id*)

**H5I\_GET\_TYPE** - Returns the object's type.

*Result* = H5I\_GET\_TYPE(*Obj\_id*)

**H5R\_CREATE** - Creates a reference to either an object or a dataspace region of a dataset.

*Result* = H5R\_CREATE(*Loc\_id*, *Name*  
[, DATASPACE\_ID=*value*])

**H5R\_DEREFERENCE** - Opens a reference and returns the object identifier.

*Result* = H5R\_DEREFERENCE(*Loc\_id*, *Reference*)

**H5R\_GET\_OBJECT\_TYPE** - Returns the type of object that an object reference points to.

*Result* = H5R\_GET\_OBJECT\_TYPE(*Loc\_id*, *Reference*)

**H5R\_GET\_REGION** - Retrieves a dataspace associated with a region reference.

*Result* = H5R\_GET\_REGION(*Dataset\_id*, *Reference*)

**H5S\_CLOSE** - Releases and terminates access to a dataspace.

H5S\_CLOSE, *Dataspace\_id*

**H5S\_COPY** - Copies an existing dataspace.

*Result* = H5S\_COPY(*Dataspace\_id*)

**H5S\_CREATE\_SCALAR** - Creates a scalar dataspace.

*Result* = H5S\_CREATE\_SCALAR()

**H5S\_CREATE\_SIMPLE** - Creates a simple dataspace.

*Result* = H5S\_CREATE\_SIMPLE(*Dimensions*  
[, MAX\_DIMENSIONS=*vector*])

**H5S\_GET\_SELECT\_BOUNDS** - Retrieves the coordinates of the bounding box containing the current dataspace selection.

*Result* = H5S\_GET\_SELECT\_BOUNDS(*Dataspace\_id*)

**H5S\_GET\_SELECT\_ELEM\_NPOINTS** - Determines the number of element points in the current dataspace selection.

*Result* = H5S\_GET\_SELECT\_ELEM\_NPOINTS  
(*Dataspace\_id*)

**H5S\_GET\_SELECT\_ELEM\_POINTLIST** - Returns a list of the element points in the current dataspace selection.

*Result* = H5S\_GET\_SELECT\_ELEM\_POINTLIST  
(*Dataspace\_id* [, START=*value*] [, NUMBER=*value*])

**H5S\_GET\_SELECT\_HYPER\_BLOCKLIST** - Returns a list of the hyperslab blocks in the current dataspace selection.

*Result* = H5S\_GET\_SELECT\_HYPER\_BLOCKLIST  
(*Dataspace\_id* [, START=*value*] [, NUMBER=*value*])

**H5S\_GET\_SELECT\_HYPER\_NBLOCKS** - Determines the number of hyperslab blocks in the current dataspace selection.

*Result* = H5S\_GET\_SELECT\_HYPER\_NBLOCKS  
(*Dataspace\_id*)

**H5S\_GET\_SELECT\_NPOINTS** - Determines the number of elements in a dataspace selection.

*Result* = H5S\_GET\_SELECT\_NPOINTS  
(*Dataspace\_id*)

**H5S\_GET\_SIMPLE\_EXTENT\_DIMS** - Returns the dimension sizes for a dataspace.

*Result* =  
H5S\_GET\_SIMPLE\_EXTENT\_DIMS(*Dataspace\_id*  
[, MAX\_DIMENSIONS=*variable*])

**H5S\_GET\_SIMPLE\_EXTENT\_NDIMS** - Determines the number of dimensions (or rank) of a dataspace.

*Result* = H5S\_GET\_SIMPLE\_EXTENT\_NDIMS  
(*Dataspace\_id*)

**H5S\_GET\_SIMPLE\_EXTENT\_NPOINTS** - Determines the number of elements in a dataspace.

*Result* = H5S\_GET\_SIMPLE\_EXTENT\_NPOINTS  
(*Dataspace\_id*)

**H5S\_GET\_SIMPLE\_EXTENT\_TYPE** - Returns the current class of a dataspace.

*Result* = H5S\_GET\_SIMPLE\_EXTENT\_TYPE  
(*Dataspace\_id*)

**H5S\_IS\_SIMPLE** - Determines whether a dataspace is a simple dataspace.

*Result* = H5S\_IS\_SIMPLE(*Dataspace\_id*)

**H5S\_OFFSET\_SIMPLE** - Sets the selection offset for a simple dataspace.

H5S\_OFFSET\_SIMPLE, *Dataspace\_id*, *Offset*

**H5S\_SELECT\_ALL** - Selects the entire extent of a dataspace.

H5S\_SELECT\_ALL, *Dataspace\_id*

**H5S\_SELECT\_ELEMENTS** - Selects array elements to be included in the selection for a dataspace.

H5S\_SELECT\_ELEMENTS, *Dataspace\_id*, *Coordinates*,  
/RESET

**H5S\_SELECT\_HYPERSLAB** - Selects a hyperslab region to be included in the selection for a dataspace.

H5S\_SELECT\_HYPERSLAB, *Dataspace\_id*, *Start*,  
*Count* [, BLOCK=*vector*] [, /RESET] [, STRIDE=*vector*]

**H5S\_SELECT\_NONE** - Resets the dataspace selection region to include no elements.

H5S\_SELECT\_NONE, *Dataspace\_id*

**H5S\_SELECT\_VALID** - Verifies that the selection is within the extent of a dataspace.

*Result* = H5S\_SELECT\_VALID(*Dataspace\_id*)

**H5S\_SET\_EXTENT\_NONE** - Removes the extent of a dataspace and sets the type to H5S\_NO\_CLASS. As such the dataspace cannot be resized or used in the creation of datasets or attributes.

H5S\_SET\_EXTENT\_NONE, *Dataspace\_id*

**H5S\_SET\_EXTENT\_SIMPLE** - Sets or resets the extent of a dataspace.

*H5S\_SET\_EXTENT\_SIMPLE, Dataspace\_id, Dimensions [,MAX\_DIMENSIONS=vector]*

**H5T\_ARRAY\_CREATE** - Creates an array datatype object.

*Result = H5T\_ARRAY\_CREATE(Datatype\_id, Dimensions)*

**H5T\_CLOSE** - Releases the specified datatype's identifier and releases resources used by it.

*H5T\_CLOSE, Datatype\_id*

**H5T\_COMMIT** - Commits a transient datatype to a file, creating a new named datatype.

*H5T\_COMMIT, Loc\_id, Name, Datatype\_id*

**H5T\_COMMITTED** - Determines whether a datatype is a named datatype or a transient type.

*Result = H5T\_COMMITTED(Datatype\_id)*

**H5T\_COMPOUND\_CREATE** - Creates a compound datatype object.

*Result = H5T\_COMPOUND\_CREATE(Datatype\_id, Name)*

**H5T\_COPY** - Copies an existing datatype.

*Result = H5T\_COPY(Datatype\_id)*

**H5T\_ENUM\_CREATE** - Creates an enumeration datatype object.

*Result = H5T\_ENUM\_CREATE()*

**H5T\_ENUM\_GET\_DATA** - Retrieves the name/value pairs from an enumeration datatype object.

*Result = H5T\_ENUM\_GET\_DATA(Datatype\_id)*

**H5T\_ENUM\_INSERT** - Inserts a new member into an existing enumeration datatype.

*H5T\_ENUM\_INSERT, Datatype\_id, Name, Value*

**H5T\_ENUM\_NAMEOF** - Retrieves the name of a member of an enumeration datatype corresponding to the specified value.

*Result = H5T\_ENUM\_NAMEOF(Datatype\_id, Value)*

**H5T\_ENUM\_SET\_DATA** - Sets all the name/value pairs on an enumeration datatype object.

*H5T\_ENUM\_SET\_DATA, Datatype\_id, Data, Values*

**H5T\_ENUM\_VALUEOF** - Retrieves the value of a member of an enumeration datatype corresponding to the specified name.

*Result = H5T\_ENUM\_VALUEOF(Datatype\_id, Name)*

**H5T\_ENUM\_VALUES\_TO\_NAMES** - Converts values to the corresponding names of an enumeration datatype.

*Result = H5T\_ENUM\_VALUES\_TO\_NAMES(Datatype\_id, Values)*

**H5T\_EQUAL** - Determines whether two datatype identifiers refer to the same datatype.

*Result = H5T\_EQUAL(Datatype\_id1, Datatype\_id2)*

**H5T\_GET\_ARRAY\_DIMS** - Returns the dimension sizes for an array datatype object.

*Result = H5T\_GET\_ARRAY\_DIMS(Datatype\_id [, PERMUTATIONS=variable])*

**H5T\_GET\_ARRAY\_NDIMS** - Determines the number of dimensions (or rank) of an array datatype object.

*Result = H5T\_GET\_ARRAY\_NDIMS(Datatype\_id)*

**H5T\_GET\_CLASS** - Returns the datatype's class.

*Result = H5T\_GET\_CLASS(Datatype\_id)*

**H5T\_GET\_CSET** - Returns the character set type of a string datatype.

*Result = H5T\_GET\_CSET(Datatype\_id)*

**H5T\_GET\_EBIAS** - Returns the exponent bias of a floating-point type.

*Result = H5T\_GET\_EBIAS(Datatype\_id)*

**H5T\_GET\_FIELDS** - Retrieves information about the positions and sizes of bit fields within a floating-point datatype.

*Result = H5T\_GET\_FIELDS(Datatype\_id)*

**H5T\_GET\_INPAD** - Returns the padding method for unused internal bits within a floating-point datatype.

*Result = H5T\_GET\_INPAD(Datatype\_id)*

**H5T\_GET\_MEMBER\_CLASS** - Returns the datatype class of a compound datatype member.

*Result = H5T\_GET\_MEMBER\_CLASS(Datatype\_id, Member)*

**H5T\_GET\_MEMBER\_INDEX** - Retrieves the index of a specified member of a compound or enumeration datatype.

*Result = H5T\_GET\_MEMBER\_INDEX(Datatype\_id, Field\_name)*

**H5T\_GET\_MEMBER\_NAME** - Returns the datatype name of a compound datatype member.

*Result = H5T\_GET\_MEMBER\_NAME(Datatype\_id, Member)*

**H5T\_GET\_MEMBER\_OFFSET** - Returns the byte offset of a field within a compound datatype.

*Result = H5T\_GET\_MEMBER\_OFFSET(Datatype\_id, Member)*

**H5T\_GET\_MEMBER\_TYPE** - Returns the datatype identifier for a specified member within a compound datatype.

*Result = H5T\_GET\_MEMBER\_TYPE(Datatype\_id, Member)*

**H5T\_GET\_MEMBER\_VALUE** - Retrieves the value of an enumeration datatype member.

*Result = H5T\_GET\_MEMBER\_VALUE(Datatype\_id, Member)*

**H5T\_GET\_NMEMBERS** - Returns the number of fields in a compound datatype.

*Result = H5T\_GET\_NMEMBERS(Datatype\_id)*

**H5T\_GET\_NORM** - Returns the mantissa normalization of a floating-point datatype.

*Result = H5T\_GET\_NORM(Datatype\_id)*

**H5T\_GET\_OFFSET** - Returns the bit offset of the first significant bit in an atomic datatype.

*Result* = H5T\_GET\_OFFSET(*Datatype\_id*)

**H5T\_GET\_ORDER** - Returns the byte order of an atomic datatype.

*Result* = H5T\_GET\_ORDER(*Datatype\_id*)

**H5T\_GET\_PAD** - Returns the padding method of the least significant bit (*lsb*) and most significant bit (*msb*) of an atomic datatype.

*Result* = H5T\_GET\_PAD(*Datatype\_id*)

**H5T\_GET\_PRECISION** - Returns the precision in bits of an atomic datatype.

*Result* = H5T\_GET\_PRECISION(*Datatype\_id*)

**H5T\_GET\_SIGN** - Returns the sign type for an integer datatype.

*Result* = H5T\_GET\_SIGN(*Datatype\_id*)

**H5T\_GET\_SIZE** - Returns the size of a datatype in bytes.

*Result* = H5T\_GET\_SIZE(*Datatype\_id*)

**H5T\_GET\_STRPAD** - Returns the padding method for a string datatype.

*Result* = H5T\_GET\_STRPAD(*Datatype\_id*)

**H5T\_GET\_SUPER** - Returns the base datatype from which a datatype is derived.

*Result* = H5T\_GET\_SUPER(*Datatype\_id*)

**H5T\_GET\_TAG** - Retrieves a tag string from an opaque data type.

*Result* = H5T\_GET\_TAG(*Datatype\_id*)

**H5T\_IDL\_CREATE** - Creates a datatype object based on the IDL type of the supplied data.

*Result* = H5T\_IDL\_CREATE(*Data*  
[,MEMBER\_NAMES=*vector*] [, /OPAQUE])

**H5T\_IDLTYPE** - Returns the IDL type code corresponding to a datatype.

*Result* = H5T\_IDLTYPE(*Datatype\_id*  
[, ARRAY\_DIMENSIONS=*variable*]  
[, STRUCTURE=*variable*])

**H5T\_INSERT** - Adds a new member to the end of a compound datatype.

H5T\_INSERT, *Datatype\_id*, *Name*, *Field\_id*

**H5T\_MEMTYPE** - Returns the native memory datatype corresponding to a file datatype.

*Result* = H5T\_MEMTYPE(*Datatype\_id*)

**H5T\_OPEN** - Opens a named datatype.

*Result* = H5T\_OPEN(*Loc\_id*, *Name*)

**H5T\_REFERENCE\_CREATE** - Creates a reference datatype object.

*Result* = H5T\_REFERENCE\_CREATE(/REGION)

**H5T\_SET\_TAG** - Sets a tag string on an opaque data type.

H5T\_SET\_TAG(*Datatype\_id*, *Tag*)

**H5T\_STR\_TO\_VLEN** - Converts an IDL string array to an IDL\_H5\_VLEN array of strings.

*Result* = H5T\_STR\_TO\_VLEN(*Array* [, /NO\_COPY])

**H5T\_VLEN\_CREATE** - Creates a variable length array datatype object.

*Result* = H5T\_VLEN\_CREATE(*Datatype\_id*)

**H5T\_VLEN\_TO\_STR** - Converts an IDL\_H5\_VLEN array of strings to an IDL string array.

*Result* = H5T\_VLEN\_TO\_STR(*Array* [, /PTR\_FREE])

## 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]  
[, *Attrnum*])

**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 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*