



What's New in IDL 6.1

RSI
Research Systems Inc.

IDL Version 6.1
July, 2004 Edition
Copyright © Research Systems, Inc.
All Rights Reserved.

Restricted Rights Notice

The IDL[®], ION Script[™], and ION Java[™] 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. Research Systems, Inc., reserves the right to make changes to this document at any time and without notice.

Limitation of Warranty

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

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

Permission to Reproduce this Manual

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

Acknowledgments

IDL[®] is a registered trademark and ION[™], ION Script[™], ION Java[™], are trademarks of Research Systems Inc., registered in the United States Patent and Trademark Office, for the computer program described herein.

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

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.

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

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 are copyrighted by Merge Technologies Incorporated.

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

IDL Wavelet Toolkit Copyright © 2002 Christopher Torrence.

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



Contents

Chapter 1:	
Overview of New Features in IDL 6.1	9
New iTools and iTool Features	10
New File Format Import/Export Accessibility in iTools	11
New iMap Tool	12
Macros and Tool History	12
Styles	13
Enhancements to the iImage Tool	13
Operations on ROIs	16
Enhancements to the iTool Data Manager	17
New Drag Quality Feature	19
iTool Background Color	19
Changes to Legend Creation	19
New iVolume Properties	20
File and Edit Menu Keyboard Accelerators	20

Enhancements to Command Line Control of iTools	20
Enhanced Handling of Axes in Empty iTools	21
Expanded Support of Format Codes	21
Visualization Enhancements	23
Lighting and Color Enhancements to Objects	23
Alpha Channel Support for Object Graphics	23
Enhancements to Mapping Routines	24
CMYK Support in Direct and Object Graphics	24
Additional Support for Vector Graphics	25
Analysis Enhancements	26
New Unsharp-mask Filter	27
Hierarchical Cluster Tree Support	28
New Integer Arithmetic for PRODUCT and TOTAL	28
Enhancements to WATERSHED	29
Double-Precision Support for Spline Interpolation	29
Double-Precision Support for Median Smoothing	29
Absolute Values for MIN and MAX Functions	29
MISSING Keyword to BILINEAR	29
Complex Data Support for NORM and COND	30
BESEL Functions and Negative Input	30
Language Enhancements	31
Ability to Query and Selectively Restore SAVE File Contents	31
Access to Non-local Scope Variables	32
New DESCRIPTION Keyword to SAVE and RESTORE	32
Enhancements to SIZE	32
Default Thread Pool Configuration	33
Easy Restoration of !CPU System Variable Values	33
Enhancements to Formatted I/O	33
Enhancements to FILE_SEARCH	36
Enhancement to CREATE_STRUCT	36
Runtime / Virtual Machine Enhancements	37
Widget Event Blocking in Runtime and Virtual Machine Modes	37
Additional Virtual Machine Enhancements	37
File Access Enhancements	38
New IDL JPEG2000 File Format Support	38

New XML DOM Object Classes	39
Expanded DICOM Support	39
Revised Language Catalog System	40
CDF Library Upgrade	40
HDF5 Library Upgrade	40
New <i>Application User Directory</i> Access	40
Enhancements to READ_TIFF	41
Enhancements to WRITE_TIFF	41
New QUERY_TIFF <i>Info</i> Structure Fields	41
GEOTIFF Support for QUERY_TIFF	41
IDLDE Enhancements	42
Intelligent File Naming	42
Maintaining Cursor Position	42
Enabling Alt Key Accelerators on Macintosh	42
User Interface Toolkit Enhancements	44
Tabbing in Widget Applications	44
Keyboard Accelerators for Button Widgets	47
DIALOG_PICKFILE Routine Enhancements	48
Table Widget Enhancements	49
Property Sheet Widget Enhancements	50
WIDGET_CONTROL and WIDGET_INFO Routine Enhancements	52
Documentation Enhancements	53
New PDF Help System Index Utility	53
Note on Macintosh Online Help	53
Enhanced Acrobat Plug-in Control	54
New Working with Maps in iTools Chapter	54
New Working with Macros in iTools Chapter	54
New Working with Styles in iTools Chapter	54
Revised iTools Data Import/Export Chapter	55
Additional <i>iTool Developer's Guide</i> Chapters	55
New Using the XML DOM Object Classes Chapter	55
New Using Language Catalogs Chapter	56
New Library Authoring Chapter	56
New <i>Medical Imaging in IDL</i> Manual	56
New IDL Routines	57
IDL Routine Enhancements	59

New IDL Object Classes	70
New IDL Object Properties	71
New IDL Object Methods	73
IDL Object Property Enhancements	76
IDL Object Method Enhancements	77
ION 6.1 Enhancements	80
Support for Secure HTTP (HTTPS)	80
Features Obsolete in IDL 6.1	81
Obsolete Routines	81
Obsolete Arguments or Keywords	81
Avoiding Backward Compatibility Issues	82
Requirements for this Release	83
IDL 6.1 Requirements	83
ION 6.1 Requirements	85
IDL-Java Bridge	87
Chapter 2:	
Library Authoring	89
Overview of Library Authoring	90
Recognizing Potential Naming Conflicts	91
Choosing Routine Names to Avoid Conflicts	91
Advice for Library Authors	93
Prefixing Routine Names	93
Converting Existing Libraries	94
Chapter 3:	
Using Language Catalogs	97
What Is a Language Catalog?	98
Creating a Language Catalog File	99
Storing and Loading Language Catalog Files	100
Using the IDLffLangCat Class	101
Creating a Language Catalog Object	101
Adding Application Keys	101
Getting and Setting Languages	102
Performing Queries	102
Destroying a Language Catalog Object	103
Widget Example	104

Chapter 4:	
Using the XML DOM Object Classes	107
About the Document Object Model	108
When to Use the DOM	108
About the DOM Structure	108
How IDL Uses the DOM Structure	110
About the XML DOM Object Classes	111
IDLffXMLDOMNode Class Hierarchy	111
IDLffXMLDOM Object Helper Classes	113
IDL Node Ownership	114
Saving and Restoring IDLffXMLDOM Objects	116
Using the XML DOM Object Classes	117
Loading an XML Document	117
Reading XML Data	118
Modifying Existing Data	119
Creating New Data	119
Destroying IDLffXMLDOM Objects	120
Working with Whitespace	121
Orphan Nodes	122
Tree-Walking Example	123
Index	125



Chapter 1: Overview of New Features in IDL 6.1

This chapter contains the following topics:

New iTools and iTool Features	10	IDL Routine Enhancements	59
Visualization Enhancements	23	New IDL Object Classes	70
Analysis Enhancements	26	New IDL Object Properties	71
Language Enhancements	31	IDL Object Property Enhancements	76
Runtime / Virtual Machine Enhancements	37	IDL Object Method Enhancements	77
File Access Enhancements	38	ION 6.1 Enhancements	80
IDLDE Enhancements	42	Features Obsoleted in IDL 6.1	81
User Interface Toolkit Enhancements	44	Avoiding Backward Compatibility Issues	82
Documentation Enhancements	53	Requirements for this Release	83
New IDL Routines	57		

New iTools and iTool Features

The Intelligent Tools (iTools) are a set of interactive utilities that combine data analysis and visualization with the task of producing presentation quality graphics. Introduced in IDL 6.0, the iTools are designed to help you get the most out of your data with minimal effort. They allow you to continue to benefit from the control of a programming language, while accelerating your data analysis through the use of interactive utilities. In IDL 6.1, we continue building on this foundation by:

- Formalizing additional framework components and documentation of these components
- Providing additional tools, rounding out the toolset
- Improving support for publication quality graphics
- Enabling batch processing

For details on these additions and other enhancements that have been made to the IDL iTools system for the 6.1 release, see the following topics:

- [“New File Format Import/Export Accessibility in iTools”](#) on page 11
- [“New iMap Tool”](#) on page 12
- [“Macros and Tool History”](#) on page 12
- [“Styles”](#) on page 13
- [“Enhancements to the iImage Tool”](#) on page 13
- [“Operations on ROIs”](#) on page 16
- [“Enhancements to the iTool Data Manager”](#) on page 17
- [“New Drag Quality Feature”](#) on page 19
- [“iTool Background Color”](#) on page 19
- [“Changes to Legend Creation”](#) on page 19
- [“New iVolume Properties”](#) on page 20
- [“File and Edit Menu Keyboard Accelerators”](#) on page 20
- [“Enhancements to Command Line Control of iTools”](#) on page 20
- [“Enhanced Handling of Axes in Empty iTools”](#) on page 21
- [“Expanded Support of Format Codes”](#) on page 21

New File Format Import/Export Accessibility in iTools

The IDL iTools can now export visualizations in Encapsulated Postscript and Windows Enhanced Metafile formats. This lets you output visualizations such as figures and plots in scalable, publication-quality formats.

<ul style="list-style-type: none"> • Encapsulated Postscript (.eps) 	<p>Exporting capability. From any iTool, select File → Export... and select Encapsulated Postscript (eps). Output will be in vector format (if you wish to have raster output, an image format such as TIFF should be used).</p>
<ul style="list-style-type: none"> • Windows Enhanced Metafile (.emf) 	<p>Exporting capability. From any iTool, select File → Export... and select Windows Enhanced Metafile (emf). Output will be in vector format (if you wish to have raster output, an image format such as TIFF should be used).</p>

Table 1-1: New Export Formats in iTools

Vector graphic support has also been enhanced in this release. Vector graphics are described by simple graphic primitives. In a vector file, IDLgrText objects are now rendered as text primitives. These text primitives as well as the other graphic primitive making up the figure or plot can be edited in vector graphic files.

The main advantages of vector graphics are excellent scalability, and the ability to easily edit text and graphic features of the objects in the display. The graphic quality is maintained regardless of whether the graphic size is increased or decreased. The capabilities of the graphic editor determines what can be successfully edited. Simple lines and horizontal text can be easily edited in an EMF file inserted into a Microsoft Word document. In addition to the iTools export format enhancements, changes to IDLgrClipboard and IDLgrPrinter support vector graphic enhancements. See “IDLgrClipboard::Draw” and “IDLgrPrinter::Draw” in the *IDL Reference Guide* for more information.

Note

How to choose the appropriate output format based on scene contents is documented in “[Bitmap and Vector Graphic Output](#)” in Chapter 34 of the *Using IDL* manual.

The IDL iTools can now import the following file types:

<ul style="list-style-type: none"> • ESRI Shapefiles (.shp) 	<p>Importing capabilities. From any iTool, select either File → Open or File → Import for ESRI Shapefiles, (these are assumed to have the .shp file extension).</p>
<ul style="list-style-type: none"> • JPEG2000 (.jpx or .jp2) 	<p>Importing capabilities. From any iTool, select either File → Open or File → Import for JPEG2000 files having the .jp2 or .jpx file extensions, or raw codestreams of any file extension.</p>

Table 1-2: New Import Formats in iTools

New iMap Tool

The new iMap tool allows you to easily display georeferenced image and contour data along with polyline, polygon and point data imported from ESRI Shapefiles. Several predefined shapefiles are provided, including continents, countries, rivers, lakes, US states and Canadian provinces. The iMap tool allows you to quickly display visualizations by defining the data to be warped to the desired map projection, and manipulate visualizations by customizing map projection parameters. See [Chapter 15, “Working with Maps”](#) in the *iTool User’s Guide* manual.

Macros and Tool History

The iTools now provide a *macro* mechanism that lets you record and playback a sequence of interactive operations. You can record a series of actions in one or more iTools, save the series as a macro, and apply it to a new set of data to save you from having to repeat the actions manually. IDL keeps a history of the iTool actions that you perform in a session. You can use the new Macro Editor to create macros from these history items, as well as from other iTool and macro operations, and you can edit previously recorded macros. For more information on macros, see [“Working with Macros”](#) in Chapter 8 of the *iTool User’s Guide* manual.

A MACRO_NAMES keyword has also been added to each iTool routine. For more information, see [“IDL Routine Enhancements”](#) on page 59.

Styles

The iTools now provide a *style* mechanism, which gives you a convenient way to store and apply a set of properties to selected items in an iTool. A new chapter in the *iTool User's Guide* describes the style capabilities and helps you get started using styles. For more information on styles, see [Chapter 9, “Working with Styles”](#).

A `STYLE_NAME` keyword has also been added to each iTool routine. For more information, see [“IDL Routine Enhancements”](#) on page 59.

Enhancements to the iImage Tool

The iImage tool has been enhanced for greater usability. The enhancements include:

- **Default pixel scale** — By default, when an image is first displayed in iImage, it appears at 100% pixel scale (one image pixel maps to one display pixel).
- **Pixel scale labels** — The current pixel scale is now reported in the iImage panel (on the right side of the iTool). The pixel scale displays, as a percentage, the number of screen pixels used to display each pixel in the selected image.
- **Axes** — Axes are no longer created automatically when an image is displayed, but can be inserted by the user.
- **Edit Palette button** — An **Edit Palette** button has been added to the iImage panel. This button provides quick access to the palette editor (also available via the property sheet for an image). The palette editor now allows you to immediately see the results of a palette change. The new **Palette Editor** is fully documented in the *iTool User's Guide* manual.
- **Image cropping functionality** — A new operation and interactive manipulator allow you to crop an image to a specified size. The new functionality is described below, and is fully documented in the *iTool User's Guide* manual.

New Cropping Functionality

New features in the iImage tool allow you to crop an image to a specified size. There are two ways to specify the region to be cropped:

1. By numerically specifying the position and size using the *crop operation*
2. By interactively clicking and dragging a bounding box using the new *crop manipulator*

When you crop an image, the original image data is replaced by the new, cropped image data. You can retrieve your original data by selecting **Edit** → **Undo Crop**, or by reloading the original image data as a separate visualization.

Note

If your original image contains regions of interest (ROIs) that do not lie completely within the crop box, they will be removed from the cropped image.

Using the Crop Operation

The Crop operation allows you to crop the selected image to a specified size at a specified location. To activate the crop operation, select **Operations** → **Crop**. The Crop manipulator is automatically activated and the Crop dialog appears.

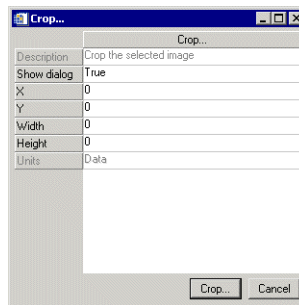


Figure 1-1: The Crop Operation Dialog

The **X** and **Y** properties of the crop operation represent the location (in the units specified by the Units property) of the lower-left corner of the crop box, relative to the lower-left corner of the image.

The **Width** and **Height** properties of the crop operation represent the dimensions (in the units specified by the Units property) of the crop box.

If the X, Y, Width, and Height values have not been set previously, either via the Crop dialog or via the crop manipulator, they are initialized to match the full size of the selected image. If the crop operation has been applied to an image in the current iTool, the values are saved as the operation's new default properties. This allows you to apply the same crop to multiple images within the same tool simply by selecting each image in turn and reselecting the crop operation.

The **Units** property specifies the units of measure to be used for reporting crop box coordinates. The default is data units. This property will automatically reset to data units (and become de-sensitized) if multiple images are selected and their pixel origins or pixel sizes differ.

Note

You can change the size and location of the crop box either by modifying the values in the Crop dialog or using the crop manipulator and your mouse.

When the crop box is sized and positioned as you desire, crop the image either by clicking the **Crop** button on the Crop dialog, or by double-clicking anywhere within the crop box itself.

Using the Crop Manipulator

The crop manipulator allows you to interactively specify the location and size of the crop box by clicking and dragging a bounding box on the selected image with your mouse. Using the manipulator to create and position a crop box automatically sets the X, Y, Width, and Height properties of the crop operation. Activate the crop manipulator by clicking on the crop button on the manipulator toolbar.



Figure 1-2: The Crop Manipulator Button on the Toolbar (Second from Right)

Note

The crop manipulator is activated automatically if you select **Crop** from the **Operations** menu.

When the crop manipulator is activated, if the crop box size and position properties of the crop operation apply for the first currently selected image, then the crop box visual will automatically appear in the primary visualization window. If the crop box information has never been set for the crop operation, or does not fit within the image, then no crop box automatically appears.

If no crop box visual is present, you can click using the mouse anywhere on an image and drag to create a crop box.

Note

All portions of the image that fall outside of the crop box are grayed out.

Once a crop box is present, you can reposition it by clicking and dragging anywhere within the crop box, or along its edges. You can also use the keyboard arrow keys to reposition the crop box.

You can change the size of the crop box by clicking and dragging on one of the scale handles. You can remove the current crop box and create a new one by clicking anywhere within the gray area outside of the current crop box.

When the crop box is sized and positioned as you desire, crop the image by:

- Double-clicking anywhere within the defined crop box
- Right-clicking to invoke the context menu, and then selecting Crop
- Selecting **Operations** → **Crop** from the iImage tool

Operations on ROIs

Several operations have been expanded to modify a selected ROI drawn on an image. Previously these operations could only be applied to the entire image. These operations can now modify the data within the ROI, or the vertices of the ROI.

Operations Modifying ROI Data

The following operations act on the data within the ROI:

- Filter operations including Convolution, Median, Smooth, Roberts, and Sobel
- Morphological operations
- Transform operations including scaling and inverting the data

Operations Modifying ROI Vertices

The following operations act on the ROI vertices, but not data within the ROI:

- Rotate right, left, or by a specified angle
- Flip horizontal or vertical
- Region grow

See [“Operations on Regions of Interest”](#) in Chapter 7 of the *iTool User’s Guide* manual for additional information.

Enhancements to the iTool Data Manager

The iTool Data Manager (Figure 1-3) has been significantly redesigned for greater ease of use. Information about the selected data item, including the iTool visualization in which it is used (if any), is now displayed in a property sheet to the right of the Data Manager tree view.

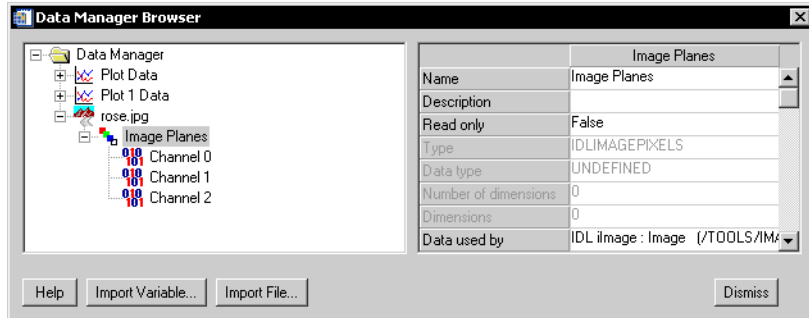


Figure 1-3: The Redesigned iTool Data Manager

The iTool Parameter Editor and the Insert Visualization dialog (Figure 1-4), both of which incorporate the Data Manager, have also been redesigned. Parameters for the selected visualization are now shown in a property sheet directly beneath the Data Manager tree view, and the data types allowed for the selected visualization parameter are now displayed to the right of the parameter list.

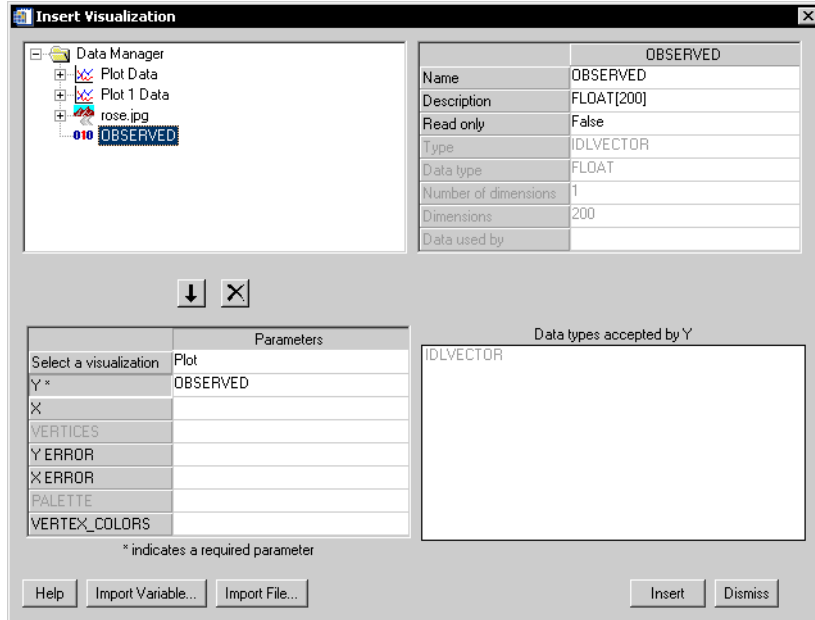


Figure 1-4: The Redesigned iTool Insert Visualization Dialog

The features of the redesigned iTool Data Manager are described in detail in [Chapter 2, “Importing and Exporting Data”](#). The features of the redesigned Parameter Editor and Insert Visualization dialog are described in [Chapter 3, “Visualizations”](#).

New Drag Quality Feature

Drag quality defines the level of detail shown in your tool window when a visualization is translated, scaled, zoomed, or otherwise moved via the mouse. There are three levels of drag quality: Low, Medium, and High. The higher the quality the better the visualizations appear during movement, but at a potential speed decrease (depending on the size of the visualization involved). The default setting is High quality.

You can alter the drag quality in two ways:

- **Drag Quality General Preference** — The General Settings category in the Preferences dialog contains the **Default drag quality** property. Changing this property means that all *new* iTools use the value specified as the default drag quality, but the behavior of the current iTool will not change.
- **Drag Quality Property** — Each iTool Window object has a **Drag quality** property, which can be changed via the Visualization Browser. Changing the value of the drag quality for the Window immediately effects visualizations in the current iTool, but it does not apply to other iTools or to future iTools.

iTool Background Color

A new `BACKGROUND_COLOR` keyword has been added to the iTool launch routines (`ICONTOUR`, `IIMAGE`, *etc.*) to allow you to set an initial background color for a view from the command line.

Changes to Legend Creation

The **Insert** → **Legend** menu item has been renamed **Insert** → **New Legend**. The **Add to Legend** toolbar manipulator has been removed and replaced by the **Insert** → **Legend Item** menu item.

Note

The **Insert** → **Legend Item** operation requires that a visualization be selected. There is no longer any need to manually select the legend itself.

New iVolume Properties

The iVolume tool has the following new properties available in the visualization property sheet:

- **Auto render** — Whether to automatically render the volume each time the window is redrawn
- **Quality** — Quality of the volume (low or high)
- **Boundary** — Boundary around the volume (off, wire frame, or solid walls)
- **Boundary transparency** — Percent transparency of the boundary around the volume
- **Render step X** — Stepping factor through the voxel matrix in the *x* direction
- **Render step Y** — Stepping factor through the voxel matrix in the *y* direction
- **Render step Z** — Stepping factor through the voxel matrix in the *z* direction

File and Edit Menu Keyboard Accelerators

The iTools now feature keyboard accelerators for the following operations:

- **File menu** — New, Open, Save, Print, Exit
- **Edit menu** — Undo, Redo, Cut, Copy, Paste, Delete
- **Operations menu, Macros submenu** — Run Macro, Start Recording, Stop Recording, Macro Editor
- **Help menu** — Help on iTools

Enhancements to Command Line Control of iTools

The new ITRESOLVE procedure is a convenience routine that resolves (compiles) all of the files used by the iTools system. This is useful when constructing SAVE files containing user code that requires the iTools framework. See “[ITRESOLVE](#)” in the *IDL Reference Guide* manual for details.

The new TOOL keyword to the ITGETCURRENT function allows you to retrieve an object reference to the currently selected iTool. See “[ITGETCURRENT](#)” in the *IDL Reference Guide* manual for details. Also see [Appendix A, “Controlling iTools from the IDL Command Line”](#) in the *iTool Developer’s Guide* manual, which describes how to programmatically control elements of an iTool.

The new ANNOTATION, FILE_READER, FILE_WRITER, and USER_INTERFACE keywords to the ITREGISTER procedure allow you to register additional iTool components with an iTool. See “ITREGISTER” in the *IDL Reference Guide* manual for details.

New MACRO_NAMES, STYLE_NAME, and BACKGROUND_COLOR keywords have also been added to each iTool routine. For more information, see “IDL Routine Enhancements” on page 59.

Enhanced Handling of Axes in Empty iTools

When all visualizations have been removed from a tool, axes are no longer displayed. This functionality has been programmatically exposed through new IDL*itVisualization* methods. See “IDL*itVisualization*” in the *IDL Reference Guide* manual for details.

Expanded Support of Format Codes

Plot axis labels, colorbar labels, contour level labels and contour legend captions now allow you to choose from an expanded list of predefined format codes or define a custom format code. The following enhancements have been made:

Axis Visualizations — additional predefined format options are available in the **Tick format** property.

Colorbar Visualization — additional predefined format options are available in the **Tick format** property.

Contour Levels — the new **Tick format code** and **Tick format** properties allow you to choose from a list of predefined format codes or create a custom format code. These properties are available when the **Label** property is set to **Value** or **Text**.

Contour Level Legend — three new properties have been added:

- **Use text from** — specifies the source of the text used for the contour legend
- **Text format code** — offers a number of predefined format code options or the ability to use a custom format code by selecting **Use Text Format Code**
- **Text format** — allows you to specify a custom format code

Additionally, examples of the predefined format codes are now shown in the **Text format code** drop-down list. A subset of the dropdown list items are shown in the following figure.

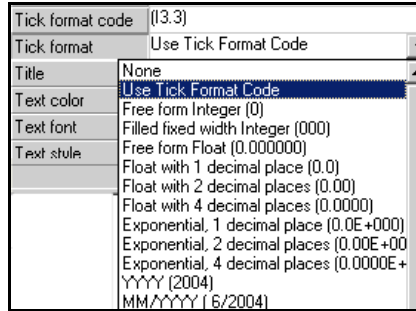


Figure 1-5: Subset of Available Predefined Format Codes

See [Appendix D, “Visualization Properties”](#) in the *iTool User’s Guide* manual for details on all available properties.

Visualization Enhancements

The following enhancements have been made to IDL's visualization functionality for the 6.1 release:

- [“Lighting and Color Enhancements to Objects”](#) below
- [“Alpha Channel Support for Object Graphics”](#) below
- [“Enhancements to Mapping Routines”](#) on page 24
- [“CMYK Support in Direct and Object Graphics”](#) on page 24
- [“Additional Support for Vector Graphics”](#) on page 25

Note

New cluster analysis and dendrite plotting functionality has also been added in this release. See [“Hierarchical Cluster Tree Support”](#) on page 28 for details.

Lighting and Color Enhancements to Objects

New properties have been added to `IDLgrPolygon` and `IDLgrSurface` to give more control over the lighting and coloring of polygons and surfaces, enabling them to simulate the appearance of materials such as shiny metals, rubber, and so on. For more information about these new properties, see [“IDLgrPolygon”](#) and [“IDLgrSurface”](#) in the *IDL Reference Guide* manual.

Alpha Channel Support for Object Graphics

A new `ALPHA_CHANNEL` property has been added to `IDLgrAxis`, `IDLgrContour`, `IDLgrPlot`, `IDLgrPolygon`, `IDLgrPolyline`, `IDLgrROI`, `IDLgrSurface`, `IDLgrSymbol`, and `IDLgrVolume` to allow you to specify the transparency of an object. Specifying transparency allows you to “see through” an object in order to see another object that was drawn before it. Modifications have also been made to various existing keywords and properties to these objects to support this functionality. Alpha channel support information has also been added in [“Alpha Channel and Objects”](#) in Chapter 23 of the *Using IDL* manual. Also see [“New IDL Object Properties”](#) on page 71.

Enhancements to Mapping Routines

The `MAP_CONTINENTS` and `MAP_GRID` procedures have been enhanced to allow display of continents and grid lines using UV (Cartesian) coordinates without the need to set up the `!MAP` system variable using the `MAP_SET` routine. See “[MAP_CONTINENTS](#)” and “[MAP_GRID](#)” in the *IDL Reference Guide* manual for details.

The new `MAP_STRUCTURE` keyword to the `MAP_IMAGE` function allows warping of an image to UV (Cartesian) coordinates without the need to set up the `!MAP` system variable using the `MAP_SET` routine. Additionally, the `MASK` keyword allows you to create a mask of “missing” values. See “[MAP_IMAGE](#)” in the *IDL Reference Guide* manual for details.

The new `FILL` keyword to the `MAP_PROJ_FORWARD` function allows you to perform a tessellation on polygons returned by the `POLYLINES` keyword after any clipping or splitting has been completed. The use of the `FILL` keyword avoids having to first pass your data through `MAP_PROJ_FORWARD`, and then again through the `IDLgrTessellator` object. See “[MAP_PROJ_FORWARD](#)” in the *IDL Reference Guide* manual for details.

CMYK Support in Direct and Object Graphics

IDL 6.1 now features CMYK (cyan, magenta, yellow, and black) color model support for PostScript output in addition to the default RGB (red, green, blue) color model. CMYK graphics are sometimes required for printing or publication. IDL’s support of CMYK PostScript output ensures that you can easily create publication-quality graphics. The direct graphics `DEVICE` procedure and the object graphics `IDLgrClipboard::Draw` method now support PostScript output using the CMYK color model via the `CMYK` keyword. See the following in the *IDL Reference Guide* for details:

- “[DEVICE](#)”
- “[IDLgrClipboard::Draw](#)”

Additional Support for Vector Graphics

The `IDLgrClipboard` and `IDLgrPrinter` destination objects allow objects in a scene, viewgroup, or view to be output as vector or bitmap graphics. Several new keywords, discussed in the following sections, provide additional control over the vector graphic output. See [“IDLgrClipboard::Draw”](#) and [“IDLgrPrinter::Draw”](#) in the *IDL Reference Guide* manual for complete reference information.

Note

When considering whether to choose bitmap or vector graphic output, see [“Guidelines for Choosing Bitmap or Vector Graphics”](#) in Chapter 34 of the *Using IDL* manual.

Smooth Shading

The `IDLgrClipboard::Draw` method `VECT_SHADING` keyword affects the appearance of surfaces and polygons when the `VECTOR` and `POSTSCRIPT` keywords have also been set. When `SHADING=1` (Gouraud shading) for `IDLgrSurface` or `IDLgrPolygon`, use this keyword to enable or disable smooth shading. See [“Smooth Shading in Vector Graphics”](#) in Chapter 34 of the *Using IDL* manual for details.

Text Rendering

The `IDLgrClipboard` or `IDLgrPrinter` `VECT_TEXT_RENDER_METHOD` keyword controls whether text appears as filled triangles or text primitives when the `VECTOR` keyword is also set. When text is rendered as text primitives, it can be edited by object-based graphics programs. See [“Text Rendering in Vector Graphics”](#) in Chapter 34 of the *Using IDL* manual for details.

Primitive Object Sorting

The `IDLgrPrinter` and `IDLgrClipboard` `Draw` methods support the `VECT_SORTING` keyword, which affects the appearance of the output when the `VECTOR` keyword has also been set. Use this keyword to simulate the depth buffer in Object Graphics in the output vector graphics file. See [“Primitive Object Sorting in Vector Graphics”](#) in Chapter 34 of the *Using IDL* manual for details.

Analysis Enhancements

The following enhancements have been made to IDL's data analysis functionality for the 6.1 release:

- [“New Unsharp-mask Filter”](#) on page 27
- [“Hierarchical Cluster Tree Support”](#) on page 28
- [“New Integer Arithmetic for PRODUCT and TOTAL”](#) on page 28
- [“Enhancements to WATERSHED”](#) on page 29
- [“Double-Precision Support for Spline Interpolation”](#) on page 29
- [“Double-Precision Support for Median Smoothing”](#) on page 29
- [“Absolute Values for MIN and MAX Functions”](#) on page 29
- [“MISSING Keyword to BILINEAR”](#) on page 29
- [“Complex Data Support for NORM and COND”](#) on page 30
- [“BESEL Functions and Negative Input”](#) on page 30

In addition to the previous additions, in IDL 6.1 several wavelet routines that were part of the Wavelet Toolkit have been integrated into the core IDL product. The Wavelet Toolkit is still available, but now provides only a GUI for wavelet analysis. The wavelet functions that are now part of the IDL library are:

- `WV_CWT` — Compute the continuous wavelet transform of an array
- `WV_DENOISE` — Denoise an array using the discrete wavelet transform
- `WV_DWT` — Compute the discrete wavelet transform of an array
- `WV_PWT` — Compute the partial wavelet transform of a vector
- `WV_FN_COIFLET` — Construct coiflet wavelet coefficients
- `WV_FN_DAUBECHIES` — Construct Daubechies wavelet coefficients
- `WV_FN_GAUSSIAN` — Construct the Gaussian wavelet function
- `WV_FN_HAAR` — Construct Haar wavelet coefficients
- `WV_FN_MORLET` — Construct the Morlet wavelet function
- `WV_FN_PAUL` — Construct the Paul wavelet function
- `WV_FN_SYMLET` — Construct symlet wavelet coefficients

Note

See [Chapter 4, “IDL Wavelet Toolkit Reference”](#) in the *IDL Wavelet Toolkit* manual for complete reference information.

New Unsharp-mask Filter

The new UNSHARP_MASK function lets you apply unsharp-mask sharpening filters to two-dimensional arrays or TrueColor images. Digital Unsharp Masking is a digital image processing technique that increases the contrast where subtle, fine details are set against a bright, diffuse background. The digital process works by subtracting from the original image an “unsharp mask” created by digitally blurring or smoothing a copy of the original image. This operation suppresses features which are smooth (those with structures on large scales) in favor of sharp features (those with structure on small scale), resulting in a net enhancement of the contrast of fine structure in the image. The following figure shows the original image (left) and filtered image (right).

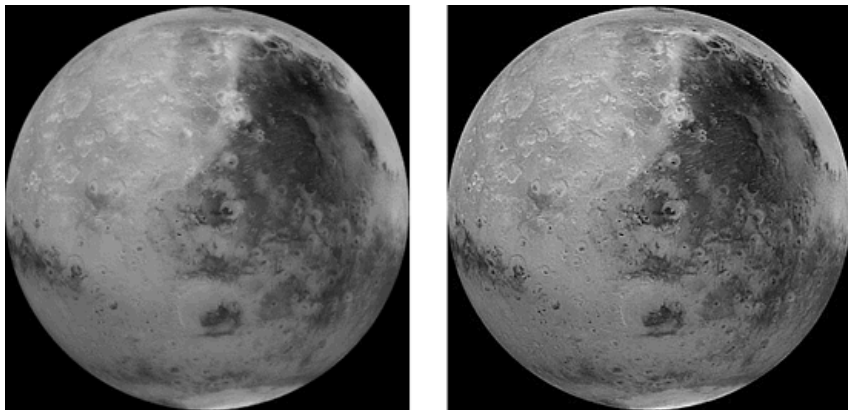


Figure 1-6: Enhancing Contrast with Unsharp Mask Filtering (Right)

See [“UNSHARP_MASK”](#) in the *IDL Reference Guide* manual for more information.

Hierarchical Cluster Tree Support

New functionality has been added to IDL 6.1 to support hierarchical cluster analysis and dendrite plotting (dendrograms). Hierarchical clustering joins together data points into successively larger clusters, using a distance (similarity) measure and linkage rules. This technique is of great importance in data exploration, as it allows researchers in a wide variety of fields to distill large amounts of multi-dimensional data down into more manageable and meaningful segments.

The following figure shows a sample dendrogram created using the new cluster analysis functionality.

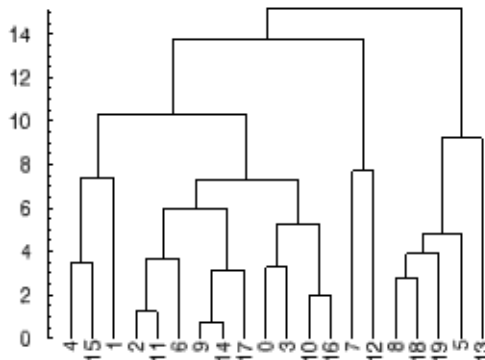


Figure 1-7: Dendrogram

The new `CLUSTER_TREE`, `DENDRO_PLOT`, `DENDROGRAM`, and `DISTANCE_MEASURE` routines all support this new functionality. For more information, see “[CLUSTER_TREE](#)”, “[DENDRO_PLOT](#)”, “[DENDROGRAM](#)”, and “[DISTANCE_MEASURE](#)” in the *IDL Reference Guide* manual.

New Integer Arithmetic for `PRODUCT` and `TOTAL`

The new `INTEGER` and `PRESERVE_TYPE` keywords to the `PRODUCT` and `TOTAL` routines allow you to use integer arithmetic and generate an integer result. See “[PRODUCT](#)” and “[TOTAL](#)” in the *IDL Reference Guide* manual for more information.

Enhancements to WATERSHED

In very “noisy” images (images with a very high number of watersheds), the number of regions may exceed the default short integer maximum return value of 32766 regions. Two new keywords have been added to gracefully deal with such cases:

- The LONG keyword causes the routine to return an array of long integers rather than an array of short integers.
- The NREGIONS keyword causes the routine to return the number of watershed regions detected.

See “[WATERSHED](#)” in the *IDL Reference Guide* manual for more information.

Double-Precision Support for Spline Interpolation

A new DOUBLE keyword adds double-precision support to the SPLINE and SPLINE_P routines, allowing you to carry additional significant digits when needed. Additionally, enhancements have been made to the *Xr* and *Yr* arguments for SPLINE_P. For more information on this new support, see “[SPLINE](#)” and “[SPLINE_P](#)” in the *IDL Reference Guide* manual.

Double-Precision Support for Median Smoothing

A new DOUBLE keyword adds double-precision support to the MEDIAN function. This provides support when passing very large integer inputs. Additionally, enhancements have been made to the *Array* argument. For more information, see “[MEDIAN](#)” in the *IDL Reference Guide* manual.

Absolute Values for MIN and MAX Functions

A new ABSOLUTE keyword to the MIN and MAX functions allows you to use absolute values when comparing data elements for both real and complex inputs. For more information, see “[MAX](#)” and “[MIN](#)” in the *IDL Reference Guide* manual.

MISSING Keyword to BILINEAR

A new MISSING keyword to the BILINEAR function lets you specify what value to return for data elements which are outside the bounds of an input array. For more information, see “[BILINEAR](#)” in the *IDL Reference Guide* manual.

Complex Data Support for NORM and COND

The NORM and COND functions now support complex data input. For more information on this functionality, see the entries for “COND” and “NORM” in the *IDL Reference Guide* manual.

BESEL Functions and Negative Input

The Bessel functions can now handle negative inputs. The existing documentation has been updated in the *IDL Reference Guide* under the individual BESEL functions.

Language Enhancements

The following enhancements have been made to the core language for the 6.1 release:

- [“Ability to Query and Selectively Restore SAVE File Contents”](#) below
- [“Access to Non-local Scope Variables”](#) on page 32
- [“New DESCRIPTION Keyword to SAVE and RESTORE”](#) on page 32
- [“Enhancements to SIZE”](#) on page 32
- [“Default Thread Pool Configuration”](#) on page 33
- [“Easy Restoration of !CPU System Variable Values”](#) on page 33
- [“Enhancements to Formatted I/O”](#) on page 33
- [“Enhancements to FILE_SEARCH”](#) on page 36
- [“Enhancement to CREATE_STRUCT”](#) on page 36

Ability to Query and Selectively Restore SAVE File Contents

The new `IDL_Savefile` object provides complete query and restore capabilities for IDL SAVE files (created with the `SAVE` procedure). Using `IDL_Savefile`, you can retrieve information about the number and size of the various items contained in a SAVE file (variables, common blocks, routines, *etc*). Individual items can be selectively restored from the SAVE file.

Use `IDL_Savefile` instead of the `RESTORE` procedure when you need to obtain detailed information on the items contained within a SAVE file without first restoring it, or when you wish to restore only selected items. Use `RESTORE` when you want to restore everything from the SAVE file using a simple interface.

For more information, see [“IDL_Savefile”](#) in the *IDL Reference Guide* manual.

Access to Non-local Scope Variables

Three new functions allow you to examine or alter variables located outside the local scope of the currently executing function or procedure. Programs, usually those with graphical user interfaces that import and export data from the caller's scope need to be able to access the user's data variables directly, without requiring them to explicitly pass those variables to the application as parameters. The new `SCOPE_VARFETCH` function is used to access those variables, while `SCOPE_VARNAME` is used to obtain the correct names with which to refer to the variables. The *iTools* are examples of programs that need to be able to perform such operations. Using these new functions also allows you to retrieve variable values in IDL Virtual Machine applications, removing the need to use the `EXECUTE` function. See "[SCOPE_LEVEL](#)", "[SCOPE_VARFETCH](#)", and "[SCOPE_VARNAME](#)" in the *IDL Reference Guide* manual for additional information.

In addition, you can retrieve information about variables in a particular scope using the `LEVEL` keyword to the `HELP` procedure. See "[HELP](#)" in the *IDL Reference Guide* manual for additional information.

New `DESCRIPTION` Keyword to `SAVE` and `RESTORE`

The new `DESCRIPTION` keyword to the `SAVE` procedure allows you to associate a descriptive string value with a `SAVE` file. The value of the `DESCRIPTION` string can be used by IDL programs to identify data; it can also be used as an aid in managing large collections of data or routines. See "[RESTORE](#)" or "[SAVE](#)" in the *IDL Reference Guide* manual for additional information.

Enhancements to `SIZE`

The new `FILE_OFFSET` keyword to the `SIZE` function allows you to retrieve the `ASSOC` file offset for the specified expression. The `SNAME` keyword allows you to retrieve the name of the structure definition for the specified expression, if it is a named structure. See "[SIZE](#)" in the *IDL Reference Guide* manual for additional information.

Default Thread Pool Configuration

The new `IDL_CPU_TPOOL_NTHREADS` environment variable allows you to set the default number of threads used by IDL in thread pool computations at startup. Set this environment variable to a value greater than 0 to specify the number of threads IDL should use. On systems shared by multiple users, you may wish to set this environment variable so that IDL uses the specified number of threads instead of defaulting to the number of CPUs present in the underlying hardware.

Easy Restoration of !CPU System Variable Values

Two new keywords to the `CPU` procedure provide the ability to set the `!CPU` system variable to previously defined sets of values. The `RESET` keyword loads the `!CPU` system variable with the values loaded when IDL starts up. The `RESTORE` keyword loads the values contained in a structure of type `!CPU` into the `!CPU` system variable. See “`CPU`” in the *IDL Reference Guide* manual for additional information.

Enhancements to Formatted I/O

IDL’s explicitly formatted input/output subsystem has been enhanced with the following new features:

New B Format Code

A new integer format code, `B`, has been added to IDL’s formatting engine. The `B` format code is used to read and write binary values. The syntax for the `B` format code is similar to the other integer format codes (`I`, `O`, and `Z`):

```
[n]B[-][w][.m]
```

where:

<i>n</i>	is an optional repeat count ($1 \leq n$) specifying the number of times the format code should be processed. If <i>n</i> is not specified, a repeat count of one is used.
----------	---

-	is an optional flag that specifies that string or numeric values should be output with the text left-justified. Normally, output is right-justified.
<i>w</i>	is an optional width specification ($0 \leq w \leq 256$). The variable <i>w</i> specifies the number of digits to be transferred.
<i>m</i>	is an optional minimum number ($1 \leq m \leq 256$) of nonblank digits to be shown on output. The field is zero-filled on the left if necessary. If <i>m</i> is omitted or zero, the output is padded with blanks to achieve the specified width. Note - The <i>m</i> parameter is ignored if <i>w</i> is zero.

See “[B, I, O, and Z Format Codes](#)” in Chapter 11 of the *Building IDL Applications* manual for complete details.

The B format code can also be used in C printf-style quoted strings, using the syntax “%b”.

For example, the following IDL statements:

```
PRINT, FORMAT='(B)', 3000
PRINT, FORMAT='(B15)', 3000
PRINT, FORMAT='(B14.14)', 3000
PRINT, FORMAT='(B0)', 3000
PRINT, FORMAT='(%" %15b")', 3000
```

Produce the following output:

```
101110111000
101110111000
00101110111000
101110111000
101110111000
```

Width Flags

+ Flag

The “+” character can be included before the *width* value of a numeric format code (B, D, E, F, G, I, O, Z or any of the numeric calendar formatting codes) to specify that positive numbers should be output with a “+” prefix. Normally, negative numbers are output with a “-” prefix and positive numbers have no sign prefix. Non-decimal numeric codes (B, O, and Z) allow the specification of the “+” flag, but ignore it.

See “[Syntax of Format Codes](#)” in Chapter 11 of the *Building IDL Applications* manual for complete details.

For example, the following the following IDL statements:

```
PRINT, FORMAT='(I0, ", ", I0)', -200, 200
PRINT, FORMAT='(I+0, ", ", I+0)', -200, 200
```

Produce the following output:

```
-200, 200
-200, +200
```

- Flag

The “-” character can be included before the *width* value of a string or numeric format code (A, B, D, E, F, G, I, O, Z or any of the calendar formatting codes) to specify that the output text should be left-justified rather than right-justified.

See “[Syntax of Format Codes](#)” in Chapter 11 of the *Building IDL Applications* manual for complete details.

For example, the following the following IDL statements:

```
PRINT, FORMAT='(I)', 234
PRINT, FORMAT='(I-)', 234
```

Produce the following output:

```
234
234
```

Zero Padding

If the width specification of a numeric format code (B, D, E, F, G, I, O, Z or any of the numeric calendar formatting codes) begins with the numeral zero, IDL will pad the value with zeroes rather than blanks. For example:

```
PRINT, FORMAT='(I08)', 300
```

produces the following output:

```
00000300
```

When padding values with zeroes, note the following:

1. If you specify the “-” flag to left-justify the output, specifying a leading zero in the *width* parameter has no effect, since there are no unused spaces to the left of the output value.
2. If you specify an explicit minimum width value (via the *m* width parameter) for an integer format code, specifying a leading zero in the *width* parameter has no effect, since the output value is already padded with zeroes on the left to create an output value of the specified minimum width.

See “[Syntax of Format Codes](#)” in Chapter 11 of the *Building IDL Applications* manual for complete details.

Natural-Width Output of Floating-Point Values

The floating-point format codes (F, D, E, G, and CSF) all accept width specifications of the form $[w.d]$ where w is an optional width specification ($0 \leq w \leq 256$) specifying the number of digits to be transferred and d is an optional width specification ($1 \leq d < w$) specifying either the number of positions after the decimal point (F, D, E, and CSF format codes) or the number of significant digits displayed (G format code).

Setting w to 0 (zero) means that you are requesting “natural width” output, meaning that the output contains no leading or trailing whitespace. In previous releases, if w was 0, IDL would ignore the value of d for the F, D, E, and CSF format codes. Now, if w is 0 and d is supplied for these codes, IDL will generate natural width output with the specified number of digits after the decimal point.

The behavior of the G format code has not changed. The d field specifies the number of significant digits of output, and as in previous releases, the value of d is used even when a natural width output is specified.

Enhancements to FILE_SEARCH

The new WINDOWS_SHORT_NAMES keyword to the FILE_SEARCH function provides greater control over how FILE_SEARCH matches Microsoft Windows “8.3 short names”. See “[FILE_SEARCH](#)” in the *IDL Reference Guide* manual for additional information.

Enhancement to CREATE_STRUCT

The new NAME keyword to the CREATE_STRUCT function allows you to create a named structure. This can be especially useful in an IDL Virtual Machine application. You can instantiate a structure defined in a SAVE file by passing in a string containing the structure name, avoiding the need to use the EXECUTE function. See “[CREATE_STRUCT](#)” in the *IDL Reference Guide* manual for additional information.

Runtime / Virtual Machine Enhancements

Widget Event Blocking in Runtime and Virtual Machine Modes

Beginning with IDL 6.1, the XMANAGER procedure honors the value of the NO_BLOCK keyword in Runtime and Virtual Machine modes. This makes it easier to develop IDL widget applications for distribution.

Because XMANAGER did not honor the NO_BLOCK keyword in previous releases, widget applications that worked properly (not blocking) when run in a licensed full version of IDL behaved differently when run in Runtime or Virtual Machine mode. This difference in behavior has been removed; widget applications should behave identically (with regard to blocking behavior) in all IDL licensing modes.

Note

Other differences between IDL's Virtual Machine mode and full licensed mode, such as the fact that programs that call the IDL EXECUTE function will not run in the IDL Virtual Machine, are still in effect.

If you have modified your widget application to work around the old blocking behavior by removing the NO_BLOCK keyword from calls to XMANAGER or by substituting the JUST_REG keyword, you may need to reinsert the NO_BLOCK keyword to achieve the desired behavior.

Additional Virtual Machine Enhancements

Other Virtual Machine enhancements include:

- When no SAVE file is supplied, the default browse directory for the Virtual Machine is now !DIR.
- When a SAVE file is supplied, the working directory is set to the location of the SAVE file. (This is true for IDL Runtime as well.)
- The IDL Virtual Machine can now be run directly from the Windows CD.
- The CREATE_STRUCT function NAME keyword and new SCOPE routines alleviate the need to use the EXECUTE function.
- Under Microsoft Windows, you can create a CD-ROM that contains all the files necessary to run a Virtual Machine application directly from the CD. See [“Distributing Your Application on a CD”](#) in Chapter 22 of the *Building IDL Applications* manual for details.

File Access Enhancements

The following enhancements have been made to IDL's file access capabilities in the IDL 6.1 release:

- [“New IDL JPEG2000 File Format Support”](#) on page 38
- [“New XML DOM Object Classes”](#) on page 39
- [“Expanded DICOM Support”](#) on page 39
- [“Revised Language Catalog System”](#) on page 40
- [“CDF Library Upgrade”](#) on page 40
- [“HDF5 Library Upgrade”](#) on page 40
- [“New Application User Directory Access”](#) on page 40
- [“Enhancements to READ_TIFF”](#) on page 41
- [“Enhancements to WRITE_TIFF”](#) on page 41
- [“New QUERY_TIFF Info Structure Fields”](#) on page 41
- [“GEOTIFF Support for QUERY_TIFF”](#) on page 41

Note

Also see [“New File Format Import/Export Accessibility in iTools”](#) on page 11 for information on exporting to .eps or .emf, and importing .shp files in iTools.

New IDL JPEG2000 File Format Support

IDL 6.1 includes support for the popular JPEG 2000 compression format, enabling researchers and software developers to achieve more efficient storage and handling of large images. JPEG 2000 is an open, international transmission standard and image compression format based on powerful wavelet technology. The format supports both lossless and lossy image compression, allowing you to store higher quality images in smaller files. JPEG 2000 support in IDL also provides unprecedented memory efficiency by letting you access, manipulate, and edit image data while the images are still in compressed form.

The new `IDLffJPEG2000` object class provides access to the features of the JPEG 2000 file format for both input and output. For more information, see [“IDLffJPEG2000”](#) in the *IDL Reference Guide* manual. Additionally, three new JPEG 2000 routines have been added to provide read, write and query functionality.

For more information on these new routines, see “[QUERY_JPEG2000](#)”, “[READ_JPEG2000](#)”, and “[WRITE_JPEG2000](#)” in the *IDL Reference Guide* manual.

New XML DOM Object Classes

IDL 6.1 supports reading, querying and writing XML files through a DOM (Document Object Model) interface. XML is eXtensible Markup Language, a popular storage standard used in sharing data across networks and the web. Support for reading XML through a SAX (Simple API for XML) interface was added in IDL 5.6. SAX can be preferable when parsing large files, while DOM allows complete file browsing and writing capabilities.

For more information on the new IDL object classes that provide access to an XML document via its document object model (DOM) see “[IDLffXMLDOM Classes](#)” in the *IDL Reference Guide* manual.

Expanded DICOM Support

The new IDLffDicomEx object, available as an optional, add-on module to IDL, greatly expands IDL’s DICOM capabilities. Previously, DICOM support was provided through the IDLffDICOM object. (See “[IDLffDICOM](#)” in the *IDL Reference Guide* for details.) For reference information, see the *Medical Imaging in IDL* manual. The IDLffDicomEx object offers the following enhancements over the IDLffDICOM object:

- Ability to read from and write to DICOM files. Using the IDLffDicomEx object, you can read, clone, or create a new DICOM file. The IDLffDICOM object only supports reading DICOM files.
- Ability to read and write both public and private attributes including sequences and sets of repeating tags within sequences (groups).
- Ability to read and write compressed DICOM files on Windows and UNIX platforms.
- Additional SOP class support.
- Ability to copy DICOM attributes from one file to another, and output all tags in a DICOM file to an ASCII file or to an IDL structure.

Note

The IDLffDicomEx object requires an additional-cost license key to access the functionality.

Revised Language Catalog System

The language catalog for widget labeling and message content has been revised to allow easier internationalization of IDL applications. The XML-based language catalogs are accessed using the new `IDLffLangCat` object. For more information, see “[IDLffLangCat](#)” in the *IDL Reference Guide* manual.

CDF Library Upgrade

IDL now uses the Common Data Format (CDF) library version 2.7r1.

HDF5 Library Upgrade

IDL now uses the Hierarchical Data Format version 5 (HDF5) library version 5-1.6.1.

New *Application User Directory* Access

The new `APP_USER_DIR` function provides access to the IDL *application user directory*. The application user directory is a location where IDL, and applications written in IDL, can store user-specific data that will persist between IDL sessions. For example, the IDL iTools store user-specified preferences, styles, and macros in the application user directory.

The application user directory is created automatically by IDL as a subdirectory (named `.idl`) of the user’s *home directory*. To prevent unrelated applications from encountering each other’s files, the `.idl` directory is organized into subdirectories with names specified by the application author. `APP_USER_DIR` simplifies cross-platform application development by providing a well-defined location for IDL applications to store their resource files, regardless of the platform or version of IDL. The uniform organization it enforces is also a benefit for IDL users, since it makes it easier for them to understand the meaning and importance of the files found in their `.idl` directory.

A related function, `APP_USER_DIR_QUERY`, allows you to locate application user directories that match a specified set of search criteria. This can be useful when migrating user data created for one version of an IDL application to a new version of the application.

For more information, see “[APP_USER_DIR](#)” and “[APP_USER_DIR_QUERY](#)” in the *IDL Reference Guide* manual.

Enhancements to READ_TIFF

Three new keywords have been added to the READ_TIFF routine. The keywords are: DOT_RANGE, ICC_PROFILE, and PHOTOSHOP. See “[READ_TIFF](#)” in the *IDL Reference Guide* manual for more details.

Enhancements to WRITE_TIFF

New keywords to the WRITE_TIFF routine include: CMYK, DESCRIPTION, DOCUMENT_NAME, DOT_RANGE, ICC_PROFILE, PHOTOSHOP, XPOSITION and YPOSITION. WRITE_TIFF now also includes in every TIFF file a DateTime tag, containing the file creation time. More notably, the new CMYK keyword supports publication-quality graphics using the cyan, magenta, yellow and black (CMYK) color model. Also, the XPOSITION and YPOSITION keywords have been added to match the TIFF 6.0 standard, specifying the location of data to be specified in raster or device space. See “[WRITE_TIFF](#)” in the *IDL Reference Guide* manual for more details.

New QUERY_TIFF *Info* Structure Fields

Four new fields have been added to the *Info* structure returned by QUERY_TIFF. The fields are: DESCRIPTION, DOCUMENT_NAME, DATE_TIME and POSITION. See “[QUERY_TIFF](#)” in the *IDL Reference Guide* manual for more details.

GEOTIFF Support for QUERY_TIFF

The new GEOTIFF keyword to QUERY_TIFF allows you to retrieve an anonymous structure containing the GeoTIFF tags and keys found in the file. GEOTIFF returns the same information as the GEOTIFF keyword to READ_TIFF, but avoids the overhead of reading in the entire image. For more information, see “[QUERY_TIFF](#)” in the *IDL Reference Guide* manual.

IDLDE Enhancements

The following enhancements have been made to the Windows IDL Development Environment in the IDL 6.1 release:

- [Intelligent File Naming](#)
- [Maintaining Cursor Position](#)
- [Enabling Alt Key Accelerators on Macintosh](#)

Intelligent File Naming

Under Microsoft Windows, when the **File** → **Save As...** option is selected for a new or existing file, the default file name is now the name of the last procedure or function in the file. This is also true when you select the **File** → **Save** option for a new file. On UNIX, the default file name remains *.pro. For portability between platforms, the default filename uses lowercase letters.

Maintaining Cursor Position

In prior releases, using the **Alt+Tab** keys to switch application focus to the IDLDE, or resetting the IDL session caused the cursor to be positioned in the Editor window regardless of its previous location. Now, the cursor position is maintained when the Command Line, Output Log, or Editor window has focus prior to resetting the IDL session, or bringing the IDLDE into focus using the **Alt+Tab** keys.


Enabling Alt Key Accelerators on Macintosh

The following new documentation has been added to the “Introducing IDL” chapter of the *Using IDL* manual, and to the “Widget Application Techniques” chapter of the *Building IDL Applications* manual:

If you are using IDL on a Macintosh and wish to use keyboard accelerators that use the **Alt** key, you will need to perform the following steps to make the **Apple** (Command) key to function as the **Alt** key:

1. Create a .xmodmap file in your home folder and add the following three lines to it:

```
clear mod1
clear mod2
add mod1 = Meta_L
```

When Apple's X11 program starts, this file will automatically be read, and the Apple key will be mapped to the left meta key , which for IDL's purposes is the **Alt** key. (Windows **Alt** key accelerators are mapped to the Macintosh **Apple** key, not the **Option (alt)** key.)

2. Run Apple's X11 program and change its preferences. Under **Input** in the X11 Preferences dialog, make sure that the following two items are *unchecked*:
 - Follow system keyboard layout
 - Enable key equivalents under X11

Note

You must relaunch Apple's X11 program for these changes to take effect.

Once you have performed these steps, keyboard shortcuts will operate in the normal Macintosh fashion — namely, pressing the **Apple** key in conjunction with X, C, and V will perform cut, copy and paste. The IDLDE's other shortcuts and any widget accelerators defined to use the **Alt** key will also work.

User Interface Toolkit Enhancements

The following enhancements have been made to the IDL's graphical user interface toolkit in the IDL 6.1 release:

- [“Tabbing in Widget Applications”](#) below
- [“Keyboard Accelerators for Button Widgets”](#) on page 47
- [“DIALOG_PICKFILE Routine Enhancements”](#) on page 48
- [“Table Widget Enhancements”](#) on page 49
- [“Property Sheet Widget Enhancements”](#) on page 50
- [“WIDGET_CONTROL and WIDGET_INFO Routine Enhancements”](#) on page 52

Tabbing in Widget Applications

New support for tabbing among widgets enables widget navigation using the **Tab** key. This allows users to quickly move between user interface elements of your widget application. Base, button, combobox, droplist, list, slider, tab, table, text, and tree widgets support the new `TAB_MODE` keyword. Set this keyword to a value indicating one of the following levels of tabbing support:

Value	Description
0	Disable navigation onto or off of the widget. This is the default unless the <code>TAB_MODE</code> has been set on a parent base. Child widgets automatically inherit the tab mode of the parent base.
1	Enable navigation onto and off of the widget.
2	Navigate only onto the widget.
3	Navigate only off of the widget.

Table 1-3: TAB_MODE Keyword Options

Note

In widget applications run on the UNIX platform, the Motif library controls what widgets are brought into and released from focus using tabbing. The `TAB_MODE` keyword value is always zero, and any attempt to change it is ignored when running a widget application on the UNIX platform.

See the following items in the *IDL Reference Guide* for details:

- `“WIDGET_BASE”`
- `“WIDGET_BUTTON”`
- `“WIDGET_COMBOBOX”`
- `“WIDGET_DROPLIST”`
- `“WIDGET_LIST”`
- `“WIDGET_SLIDER”`
- `“WIDGET_TAB”`
- `“WIDGET_TABLE”`
- `“WIDGET_TEXT”`
- `“WIDGET_TREE”`

Note

Several compound widgets also feature the `TAB_MODE` keyword. See [“IDL Routine Enhancements”](#) on page 59 for more information.

IDL GUIBuilder and Tab Mode

The IDL GUIBuilder also features a new Tab Mode attribute that defines to what degree tabbing can be used to navigate the widget hierarchy in a widget application. By default, this value is set to None for a top level base, and to Inherit for subsequent bases and widgets. Like the TAB_MODE keyword, the Tab Mode attribute applies to base, button, combobox, droplist, list, slider, tab, table, text, and tree widgets.

Allowable values are:

Value	Description
Inherit	Upon creation, the subsequent base inherits the tabbing support of the parent base. This is the default for child widgets.
None	Disallow tabbing into or out of the base. This is the default for top level bases.
In and Out	Allow tabbing into and out of the base.
In Only	Allow tabbing into the base only.
Out Only	Allow tabbing off of the base only.

Table 1-4: Allowable Tab Mode Values in IDL GUIBuilder

Note

The default Tab Mode of lower level bases and widgets is Inherit. The tabbing support defined for a top level or parent base is inherited by widget children unless otherwise specified. When the value is Inherit, look at the tabbing support of the parent base to determine what support the individual widget has for tabbing.

In the generated * .PRO file, this value is specified with the TAB_MODE keyword to the widget creation routine.

Keyboard Accelerators for Button Widgets

Button widgets now support the `ACCELERATOR` keyword. Accelerators allow you to activate button widget events using keyboard key combinations instead of requiring mouse clicks. These can enhance the usability of your IDL application.

Support for accelerators varies slightly by platform and usage:

- Under Microsoft Windows, accelerators can be defined for menu items and various types of `WIDGET_BUTTON` widgets.
- Under UNIX, accelerators can only be applied to menu items.
- Context menu items do not support accelerators on any platform.

See “[WIDGET_BUTTON](#)” in the *IDL Reference Guide* manual for more information.

Note

Special steps are required to enable accelerators that use the **Alt** key on Macintosh platforms. See “[Enabling Alt Key Accelerators on Macintosh](#)” in Chapter 30 of the *Building IDL Applications* manual for details.

Disabling Accelerators

Ordinarily, accelerators are processed before keyboard events reach widgets that have keyboard focus. Setting `IGNORE_ACCELERATORS` allows [WIDGET_DRAW](#) and widgets with an editable text area ([WIDGET_COMBOBOX](#), [WIDGET_PROPERTY SHEET](#), [WIDGET_TABLE](#) and [WIDGET_TEXT](#)) to receive keyboard events instead of the accelerator key combinations being captured by the accelerator. See “[Disabling Button Widget Accelerators](#)” in Chapter 30 of the *Building IDL Applications* manual for usage details and examples.

DIALOG_PICKFILE Routine Enhancements

A new Browse for Folder dialog is available under Microsoft Windows when the `DIRECTORY` keyword is set. You can select, or create and select, a directory using the new dialog. Other improvements include the availability of horizontal scrolling (when needed) to easily see entire filenames regardless of their length.

The following figure shows the new Browse for Folder dialog.

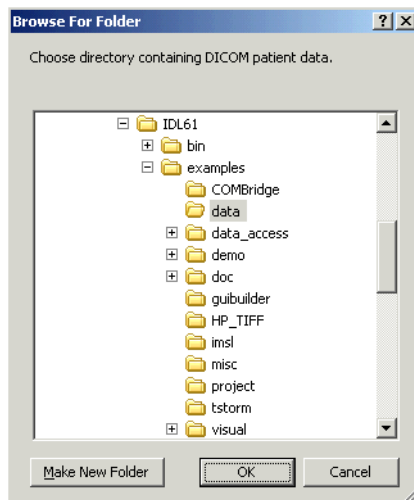


Figure 1-8: `DIALOG_PICKFILE` with `DIRECTORY` Keyword Set

Table Widget Enhancements

The `WIDGET_TABLE` function features several new keywords. The `NO_COLUMN_HEADERS` and `NO_ROW_HEADERS` keywords allow you to customize the display of a table. The `CONTEXT_EVENTS` keyword causes context menu events to be initiated when you right-click over a table widget. The `IGNORE_ACCELERATORS` keyword allows editable table cells to receive keyboard combinations mapped to an accelerator. For more information on the new table widget keywords and context menu event structure, see “[WIDGET_TABLE](#)” in the *IDL Reference Guide* manual.

Warning

The `WIDGET_CONTEXT` event structure associated with base, list, property sheet, table, text and tree widgets has been updated with new `ROW` and `COL` fields. This may require code changes as described in “[Avoiding Backward Compatibility Issues](#)” on page 82.

Property Sheet Widget Enhancements

The following enhancements to the WIDGET_PROPERTY SHEET function improve the usability and appearance of property sheets. Natural sizing and the ability to select multiple properties are internal changes that automatically improve property sheet usability. Other improvements allow you to programmatically control the selection and editability of properties. Enhancements include:

- Property sheets without an explicit size definition (lacking a specified SCR_XSIZE or XSIZE keyword value) are now *naturally sized*. Column widths depend on the contents of the components. Naturally sized property sheets allow the full contents of the longest cell to be visible in a column, as shown in the left-hand image in the following figure. When a size definition is provided, selecting the cell displays the list contents in a drop-down box that is wide enough for the longest item as shown in the right-hand image in the following figure.

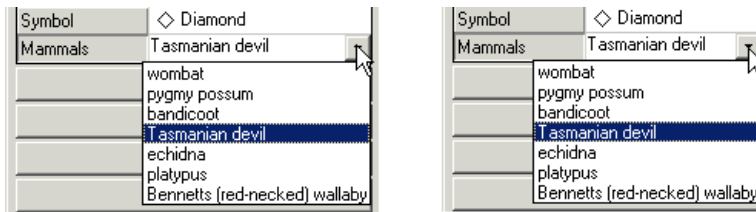


Figure 1-9: Property Sheet Column Sizing

See “[Property Sheet Sizing](#)” in Chapter 30 of the *Building IDL Applications* manual for more information.

- Setting the MULTIPLE_PROPERTIES keyword allows multiple properties to be selected at a single time by depressing the **Shift** key and left-clicking (to make adjacent selections), or by depressing the **Ctrl** key and left-clicking (to make nonadjacent selections). Properties can also be programmatically selected using the PROPERTY SHEET_SETSELECTED keyword as described in “[WIDGET_CONTROL](#)” in the *IDL Reference Guide* manual.
- To support multiple selection, the WIDGET_PROPSHEET_SELECT event structure IDENTIFIER field has been expanded, and there is a new NSELECTED field. See the “[Select Event](#)” section of “[WIDGET_PROPERTY SHEET](#)” in the *IDL Reference Guide* manual for details. Programs accessing this event structure *may* require code changes. See “[Avoiding Backward Compatibility Issues](#)” on page 82 for details.

- The new EDITABLE keyword allows you to mark a property sheet as read-only. You can select properties in a read-only property sheet, but cannot modify property values.
- The IGNORE_ACCELERATORS keyword allows editable text cells in a property sheet to receive keyboard combinations mapped to an accelerator.
- A property sheet can be surrounded by a border of a specified width using the FRAME keyword, or can appear to be inset by setting the SUNKEN_FRAME keyword.

Note

See “[WIDGET_PROPERTY SHEET](#)” in the *IDL Reference Guide* manual for details on the new keywords.

- The new *spinner* control associated with a selected number cell allows you to click, or click and hold the up or down arrows to change the numerical value. The appearance of a number cell on a property sheet is controlled by the VALID_RANGE keyword to IDLitComponent::RegisterProperty method. See “[IDLitComponent::RegisterProperty](#)” in the *IDL Reference Guide* manual for details.

WIDGET_CONTROL and WIDGET_INFO Routine Enhancements

The following sections describe enhancements that have been made to the WIDGET_INFO and WIDGET_CONTROL routines in IDL 6.1. Major areas of functionality added to widgets in this release, including tabbing and programmatic selection within property sheets, can be enabled and disabled using WIDGET_CONTROL or returned using WIDGET_INFO.

WIDGET_CONTROL Routine Enhancements

WIDGET_CONTROL routine enhancements include:

- **TAB_MODE** keyword — provides the ability to control tabbing support for widgets that support tabbing. See [“Tabbing in Widget Applications”](#) on page 44 for more information.
- **PROPERTY SHEET_SETSELECTED** keyword — allows programmatic selection of properties on property sheet widgets.
- **EDITABLE** keyword — defines a property sheet widget as read-only, allowing the user to select, but not modify properties.
- **MULTIPLE_PROPERTIES** keyword — enables or disables multiple property selection in a property sheet widget.

See [“WIDGET_CONTROL”](#) in the *IDL Reference Guide* manual for details.

WIDGET_INFO Routine Enhancements

WIDGET_INFO routine enhancements include:

- **TAB_MODE** keyword – provides the ability to determine the tabbing support of any widget.
- **PROPERTY SHEET_NSELECTED** keyword — returns the number of selected properties in a property sheet widget.
- **PROPERTY SHEET_SELECTED** keyword — returns the name (identifier) of each selected property in a property sheet widget.
- **MULTIPLE_PROPERTIES** keyword — returns a value indicating the support for multiple property selection in a property sheet widget.

See [“WIDGET_INFO”](#) in the *IDL Reference Guide* manual for details.

Documentation Enhancements

In addition to documentation for new and enhanced IDL features, the following enhancements to the IDL documentation set are included in the 6.1 release:

- [“New PDF Help System Index Utility”](#) below
- [“Note on Macintosh Online Help”](#) below
- [“Enhanced Acrobat Plug-in Control”](#) on page 54
- [“New Working with Maps in iTools Chapter”](#) on page 54
- [“New Working with Macros in iTools Chapter”](#) on page 54
- [“New Working with Styles in iTools Chapter”](#) on page 54
- [“Revised iTools Data Import/Export Chapter”](#) on page 55
- [“Additional iTool Developer’s Guide Chapters”](#) on page 55
- [“New Using the XML DOM Object Classes Chapter”](#) on page 55
- [“New Using Language Catalogs Chapter”](#) on page 56
- [“New Library Authoring Chapter”](#) on page 56
- [“New Medical Imaging in IDL Manual”](#) on page 56

New PDF Help System Index Utility

A new IDL widget-based index displays a searchable index of IDL’s PDF help system on UNIX platforms that support the IDL-Acrobat plug-in. The utility is launched automatically when using either IDL’s “?” facility or the ONLINE_HELP procedure.

Note

IDL for Macintosh does not support the IDL-Acrobat plug-in.

Note on Macintosh Online Help

RSI’s choice of Adobe Acrobat Reader as the IDL online help viewer on the Macintosh platform was based on the belief that Acrobat Reader would support the technologies (available on other UNIX platforms) necessary to create a full-featured help system. This assumption turns out not to have been correct.

As a result, RSI is actively searching for online help technology to replace our use of Adobe Acrobat, both on the Macintosh and on other UNIX platforms. We realize that the lack of a full-featured online help system on the Macintosh is a significant inconvenience for our users, we hope to be able to provide a robust replacement for the Acrobat help system in the very near future.

Enhanced Acrobat Plug-in Control

The new `SUPPRESS_PLUGIN_ERRORS` keyword to the `ONLINE_HELP` routine allows you to prevent warnings from being issued when the IDL-Acrobat plug-in is not available. See [“ONLINE_HELP”](#) in the *IDL Reference Guide* manual for additional information.

New Working with Maps in iTools Chapter

The new iMap tool allows you to easily display georeferenced image and contour data along with polyline, polygon and point data imported from ESRI Shapefiles. Several predefined shapefiles are provided, including continents, countries, rivers, lakes, states & provinces, and cities. The iMap tool allows you to quickly display visualizations by defining the data to be warped to the desired map projection, and manipulate visualizations by customizing map projection parameters. For more information, see [Chapter 15, “Working with Maps”](#) in the *iTool User’s Guide* manual.

New Working with Macros in iTools Chapter

The iTools now provide a *macro* mechanism, which allows you to record and replay a sequence of interactive operations. You can record a series of actions in one iTool or several iTools, save the series as a macro, and then apply it to a new set of data to save you from having to repeat the actions manually. A new chapter in the *iTool User’s Guide* guides you through the iTool macro capabilities and helps you get started using macros. For more information, see [Chapter 8, “Working with Macros”](#).

New Working with Styles in iTools Chapter

The iTools now provide a *style* mechanism, which gives you a convenient way to store and apply a set of properties to selected items in an iTool. A new chapter in the *iTool User’s Guide* describes the style capabilities and helps you get started using styles. For more information on styles, see [Chapter 9, “Working with Styles”](#).

Revised iTools Data Import/Export Chapter

In addition to the above-mentioned additions to the *iTool User's Guide*, the “Importing and Exporting Data” chapter has been significantly revised and expanded to reflect changes to the iTools Data Manager. See [Chapter 2, “Importing and Exporting Data”](#) and [Chapter 3, “Visualizations”](#) for details.

Additional *iTool Developer's Guide* Chapters

The *iTool Developer's Guide* manual has been expanded to include how-to material for the following topics:

- [Chapter 8, “Creating a Manipulator”](#) describes creating a custom manipulator that can be registered with an iTool. A manipulator is an iTool component class that defines a way the user can interact with visualizations in the iTool window using the mouse or keyboard.
- [Chapter 15, “Creating a Custom iTool Widget Interface”](#) describes creating a new IDL widget-based user interface that includes iTool components. This includes using a number of special compound widgets designed to work with the iTool system.
- [Appendix A, “Controlling iTools from the IDL Command Line”](#) describes how to programmatically control elements of an iTool. Topics include accessing the tool, retrieving identifiers of visualizations, selecting visualizations, setting properties on those visualizations, and executing operations.
- [Appendix B, “iTool Compound Widgets”](#) describes compound widgets that provide the base functionality needed to create an iTool user interface using IDL widgets as described in [Chapter 15, “Creating a Custom iTool Widget Interface”](#).

New Using the XML DOM Object Classes Chapter

IDL now allows access to an XML document via its document object model (DOM), using a set of object classes. A new chapter in *Building IDL Applications* guides you through the DOM capability and helps you get started using the DOM object classes. For more information, see [Chapter 4, “Using the XML DOM Object Classes”](#).

New Using Language Catalogs Chapter

IDL's new XML-based *language catalog* feature allows you to create a localized interface with text strings displayed in a language you select, using string translations you provide. For more information, see [Chapter 3, "Using Language Catalogs"](#).

New Library Authoring Chapter

To enhance the visibility of the importance of developing a consistent naming scheme for your library of routines, a new chapter provides information on naming conflicts and guidelines for library authors including how to convert existing libraries. See [Chapter 2, "Library Authoring"](#) for details.

New *Medical Imaging in IDL* Manual

The new IDLffDicomEx object expands IDL's DICOM capabilities, allowing you to read and write public and private attributes (including sequences and nested sequences) and compressed data in DICOM files. This new "[Medical Imaging in IDL](#)" manual provides complete reference information as well as details on supported platforms. The IDLffDicomEx object, available as an optional, add-on module to IDL, requires an additional-cost license.

New IDL Routines

The following new functions and procedures were added to IDL in this release. See the following topics in the *IDL Reference Guide* for complete reference information.

- “**APP_USER_DIR**” — The APP_USER_DIR function provides access to the IDL *application user directory*.
- “**APP_USER_DIR_QUERY**” — The APP_USER_DIR_QUERY function allows you to search for *application user directories*.
- “**CLUSTER_TREE**” — The CLUSTER_TREE function computes the hierarchical clustering for a set of m items in an n -dimensional space.
- “**CREATE_CURSOR**” — This function creates an image array from a string array that represents a 16 by 16 window cursor. The returned image array can be passed to the REGISTER_CURSOR procedure *Image* argument. This allows you to quickly design a cursor using a simple string array.
- “**DENDRO_PLOT**” — Given a hierarchical tree cluster, as created by CLUSTER_TREE, the DENDRO_PLOT procedure draws a two-dimensional dendrite plot on the current direct graphics device.
- “**DENDROGRAM**” — Given a hierarchical tree cluster, as created by CLUSTER_TREE, the DENDROGRAM procedure constructs a dendrogram and returns a set of vertices and connectivity that can be used to visualize the dendrite plot.
- “**DISTANCE_MEASURE**” — The DISTANCE_MEASURE function computes the pairwise distance between a set of items or observations.
- “**IMAP**” — The IMAP procedure creates an iTool and associated user interface configured to display and manipulate image and contour data that is georeferenced.
- “**ITRESOLVE**” — The ITRESOLVE procedure resolves all IDL code within the iTools directory, as well as all other IDL code required for the iTools framework. This procedure is useful for constructing SAVE files containing user code that requires the iTools framework.
- “**MAP_PROJ_IMAGE**” — The MAP_PROJ_IMAGE function warps an image (or other dataset) from geographic coordinates (longitude and latitude) to a specified map projection. (Use the MAP_SET or MAP_PROJ_INIT procedure to set up the desired map projection.) Optionally, the MAP_PROJ_IMAGE function can be used to warp an image in Cartesian (UV) coordinates from one map projection to another.

- “**QUERY_JPEG2000**” — The QUERY_JPEG2000 function allows you to obtain information about a JPEG2000 image file without having to read the file.
- “**READ_JPEG2000**” — The READ_JPEG2000 function extracts and returns image data from a JPEG2000 file.
- “**SCOPE_LEVEL**” — The SCOPE_LEVEL function returns the scope level of the currently running procedure or function.
- “**SCOPE_VARFETCH**” — The SCOPE_VARFETCH function returns variables outside the local scope of the currently running procedure or function.
- “**SCOPE_VARNAME**” — The SCOPE_VARNAME function returns the names of variables outside of the local scope of the currently running procedure or function.
- “**UNSHARP_MASK**” — The UNSHARP_MASK function performs an unsharp-mask sharpening filter on a two-dimensional array or a TrueColor image. For TrueColor images the unsharp mask is applied to each channel.
- “**WRITE_JPEG2000**” — The WRITE_JPEG2000 procedure writes image data into a JPEG2000 file.

IDL Routine Enhancements

The following IDL routines have updated keywords, arguments, or return values in this release. See the following topics in the *IDL Reference Guide* for complete reference information unless otherwise noted.

“BESELI”, “BESELJ”, “BESELK”, and “BESELY” — BESSEL* functions can now handle negative inputs and return complex results.

“BILINEAR” — The BILINEAR function includes a new keyword:

- MISSING assigns a value to return for elements outside the bounds of a specified data array, *P*.

“COND” — The COND function now supports complex data.

“CPU” — The CPU function includes the following new keywords:

- RESET allows you to reset the values contained in the !CPU system variable.
- RESTORE allows you to populate the !CPU system variable with values contained in a specified structure.

“CREATE_STRUCT” — The CREATE_STRUCT function includes a new keyword:

- NAME allows you to create a named structure.

“CW_ANIMATE” — The CW_ANIMATE function includes the following new keyword:

- TAB_MODE allows you to define tabbing behavior among widgets in an application.

“CW_ARCBALL” — The CW_ARCBALL function includes the following new keyword:

- TAB_MODE allows you to define tabbing behavior among widgets in an application.

“CW_BGROU” — The CW_BGROU function includes the following new keyword:

- TAB_MODE allows you to define tabbing behavior among widgets in an application.

“CW_CLR_INDEX” — The CW_CLR_INDEX function includes the following new keyword:

- TAB_MODE allows you to define tabbing behavior among widgets in an application.

“CW_COLORSEL” — The CW_COLORSEL function includes the following new keyword:

- TAB_MODE allows you to define tabbing behavior among widgets in an application.

“CW_DEFROI” — The CW_DEFROI function includes the following new keyword:

- TAB_MODE allows you to define tabbing behavior among widgets in an application.

“CW_FIELD” — The CW_FIELD function includes the following new keyword:

- TAB_MODE allows you to define tabbing behavior among widgets in an application.

“CW_FILESEL” — The CW_FILESEL function includes the following new keyword:

- TAB_MODE allows you to define tabbing behavior among widgets in an application.

“CW_FORM” — The CW_FORM function includes the following new keyword:

- TAB_MODE allows you to define tabbing behavior among widgets in an application.

“CW_FSLIDER” — The CW_FSLIDER function includes the following new keyword:

- TAB_MODE allows you to define tabbing behavior among widgets in an application.

“CW_LIGHT_EDITOR” — The CW_LIGHT_EDITOR function includes the following new keyword:

- TAB_MODE allows you to define tabbing behavior among widgets in an application.

“CW_ORIENT” — The CW_ORIENT function includes the following new keyword:

- TAB_MODE allows you to define tabbing behavior among widgets in an application.

“CW_PALETTE_EDITOR” — The CW_PALETTE_EDITOR function includes the following new keyword:

- TAB_MODE allows you to define tabbing behavior among widgets in an application.

“CW_PDMENU” — The CW_PDMENU function includes the following new keyword:

- TAB_MODE allows you to define tabbing behavior among widgets in an application.

“CW_RGBSLIDER” — The CW_RGBSLIDER function includes the following new keyword:

- TAB_MODE allows you to define tabbing behavior among widgets in an application.

“CW_TMPL” — The CW_TMPL function includes the following new keyword:

- TAB_MODE allows you to define tabbing behavior among widgets in an application.

“CW_ZOOM” — The CW_ZOOM function includes the following new keyword:

- TAB_MODE allows you to define tabbing behavior among widgets in an application.

“DEVICE” — The DEVICE procedure includes the following new keyword:

- CMYK allows you to generate PostScript output using the CMYK (cyan, magenta, yellow, and black) color model.

“EXECUTE” — The EXECUTE procedure includes the new argument:

- *QuietExecution* suppresses reporting of execution errors.

“FILE_SEARCH” — The FILE_SEARCH function includes the following new keyword:

- WINDOWS_SHORT_NAMES searches 8.3 short names as well as the real file names for a match.

“GETENV” — The GETENV functions features the following modified keyword:

- ENVIRONMENT is now available on all platforms

“HELP” — The HELP procedure includes the following new keyword:

- LEVEL allows you to return variables for routines other than the currently executing routine.

“ICONTOUR” — The ICONTOUR procedure includes the following new keywords:

- BACKGROUND_COLOR allows you to set the tool’s background color.
- GRID_UNITS allows you to specify the contour grid’s units when a map projection is inserted.
- MACRO_NAMES allows you to specify one or more macros to be applied to the created visualizations.
- STYLE_NAME allows you to specify the name of a user-defined or system style to be applied to the created visualizations.

“IDLITSYS_CREATETOOL” — The IDLITSYS_CREATETOOL function includes the following new keyword:

- MACRO_NAMES allows you to specify one or more macros to be applied to the created visualizations.
- STYLE_NAME allows you to specify the name of a user-defined or system style to be applied to the created visualizations.

“IIMAGE” — The IIMAGE procedure includes the following new keywords:

- BACKGROUND_COLOR allows you to set the tool’s background color.
- GRID_UNITS allows you to specify the image grid’s units when a map projection is inserted.
- MACRO_NAMES allows you to specify one or more macros to be applied to the created visualizations.
- STYLE_NAME allows you to specify the name of a user-defined or system style to be applied to the created visualizations.

“I PLOT” — The I PLOT procedure includes the following new keywords:

- BACKGROUND_COLOR allows you to set the tool’s background color.
- MACRO_NAMES allows you to specify one or more macros to be applied to the created visualizations.
- STYLE_NAME allows you to specify the name of a user-defined or system style to be applied to the created visualizations.

“ISURFACE” — The ISURFACE procedure includes the following new keywords:

- BACKGROUND_COLOR allows you to set the tool’s background color.
- MACRO_NAMES allows you to specify one or more macros to be applied to the created visualizations.
- STYLE_NAME allows you to specify the name of a user-defined or system style to be applied to the created visualizations.

“ITGETCURRENT” — The ITGETCURRENT function includes the following new keyword:

- TOOL allows you to return an object reference to the current tool.

“ITREGISTER” — The ITREGISTER procedure features the following new keywords:

- ANNOTATION, FILE_READER, FILE_WRITER and USER_INTERFACE allow you to register annotations, file readers and writers, and user interfaces.
- DEFAULT allows you to specify that the registered item is the default for its type.

“IVOLUME” — The IVOLUME procedure includes the following new keywords:

- BACKGROUND_COLOR allows you to set the tool’s background color.
- EXTENTS_TRANSPARENCY allows you to specify the percent transparency of a volume’s boundary.
- MACRO_NAMES allows you to specify one or more macros to be applied to the created visualizations.
- STYLE_NAME allows you to specify the name of a user-defined or system style to be applied to the created visualizations.

“LSODE” — The LSODE function includes the following new keyword:

- QUIET allows you to suppress error messages.

“MAP_CONTINENTS” — The MAP_CONTINENTS procedure features the following new keyword:

- MAP_STRUCTURE allows you to draw using UV (Cartesian) coordinates, bypassing the value stored in the !MAP system variable.

“MAP_GRID” — The MAP_GRID procedure features the following new keyword:

- MAP_STRUCTURE allows you to draw using UV (Cartesian) coordinates, bypassing the value stored in the !MAP system variable.

“MAP_IMAGE” — The MAP_IMAGE function features the following new keywords:

- MAP_STRUCTURE allows you to warp an image using UV (Cartesian) coordinates, bypassing the value stored in the !MAP system variable.
- MASK allows you to create a mask of “missing” values.

“MAP_PROJ_FORWARD” — The MAP_PROJ_FORWARD function includes the following new keyword:

- FILL allows you to perform a tessellation on the returned polygons.

“MAX” — The MAX function includes the following new keyword:

- ABSOLUTE allows you to determine the maximum value from the absolute value of each element.

“MEDIAN” — The MEDIAN function includes the following new keyword:

- DOUBLE allows you to force the use of double-precision arithmetic in computations.

“MIN” — The MIN function includes the following new keyword:

- ABSOLUTE allows you to determine the minimum value from the absolute value of each element.

“NORM” — The NORM function now supports complex data.

“ONLINE_HELP” — The ONLINE_HELP procedure features the following new keyword:

- SUPPRESS_PLUGIN_ERRORS allows you to prevent warnings regarding the unavailability of the IDL-Acrobat plug-in under Unix.

“PRODUCT” — The PRODUCT function includes the following new keywords:

- INTEGER allows you to perform the product operation using integer arithmetic
- PRESERVE_TYPE allows you to perform the product operation and return a result of the same type as the input.

- “QUERY_TIFF”** — The QUERY_TIFF function includes new Info structure fields and a new keyword:
- DESCRIPTION field allows you to return ImageDescription tag contents.
 - DOCUMENT_NAME field allows you to return DocumentName tag contents.
 - DATE_TIME field allows you to return the DateTime tag contents.
 - POSITION field contains x and y offsets.
 - GEOTIFF keyword allows you to return the GeoTIFF information present in a file.
- “READ_TIFF”** — The READ_TIFF function includes the following new keywords:
- DOT_RANGE allows you to return the TIFF DotRange tag value.
 - ICC_PROFILE allows you to return the TIFF ICC_PROFILE tag value.
 - PHOTOSHOP allows you to return the TIFF PHOTOSHOP tag value.
- “REGION_GROW”** — The REGION_GROW function includes the following new keyword:
- NAN allows you to treat the special floating-point values *NaN* and *Infinity* as missing values.
- “RESTORE”** — The RESTORE procedure includes the following new keyword:
- DESCRIPTION allows you to return a save file description.
- “SAVE”** — The SAVE procedure includes the following new keyword:
- DESCRIPTION allows you to specify a save file description.
- “SIZE”** — The SIZE function includes the following new keywords, with related field additions in the STRUCTURE keyword:
- FILE_OFFSET allows you to return the ASSOC file offset.
 - SNAME allows you to return the structure definition of the given expression.
- “SPLINE”** — The SPLINE function includes the following new keyword:
- DOUBLE allows you to force the use of double-precision arithmetic in computations.

“SPLINE_P” — The SPLINE_P function includes the following new keyword and modified arguments:

- DOUBLE keyword allows you to force the use of double-precision arithmetic in computations.
- Xr and Yr arguments provide support for double-precision arithmetic.

“TOTAL” — The TOTAL function includes the following new keywords:

- INTEGER allows you to perform the total operation using integer arithmetic.
- PRESERVE_TYPE allows you to perform the total operation and return a result of the same type as the input.

“WATERSHED” — The WATERSHED function features the following new keywords:

- LONG allows you to return an array of long integers.
- NREGIONS allows you to return the total number of regions within the given image.

“WIDGET_BASE” — The WIDGET_BASE function includes the following new keyword:

- TAB_MODE allows you to define tabbing behavior among widgets in an application.

“WIDGET_BUTTON” — The WIDGET_BUTTON function includes the following new keywords:

- TAB_MODE allows you to define tabbing behavior among widgets in an application.
- ACCELERATOR allows you to activate a button widget using a keyboard combination.

“WIDGET_COMBOBOX” — The WIDGET_COMBOBOX function includes the following new keyword:

- TAB_MODE allows you to define tabbing behavior among widgets in an application.
- IGNORE_ACCELERATORS keyword allows the editable text area to receive keyboard combinations mapped to a WIDGET_BUTTON accelerator.

“WIDGET_CONTROL” — The WIDGET_CONTROL function includes the following new keywords:

- EDITABLE allows you to specify whether or not property sheet properties can be edited.
- PROPERTYSHEET_SETSELECTED allows you programmatically select property sheet properties.
- TAB_MODE allows you to define tabbing behavior among widgets in an application.

“WIDGET_DRAW” — The WIDGET_DRAW function includes the following new keyword:

- IGNORE_ACCELERATORS allows the draw widget to receive keyboard combinations mapped to a WIDGET_BUTTON accelerator.

“WIDGET_DROPLIST” — The WIDGET_DROPLIST function includes the following new keyword:

- TAB_MODE allows you to define tabbing behavior among widgets in an application.

“WIDGET_INFO” — The WIDGET_INFO function includes the following new keywords:

- PROPERTYSHEET_NSELECTED allows you to return the number of selected properties in a property sheet.
- PROPERTYSHEET_SELECTED allows you to return a list of selected properties in a property sheet.
- TAB_MODE allows you to return the defined tabbing support of a widget.

“WIDGET_LIST” — The WIDGET_LIST function includes the following new keyword:

- TAB_MODE allows you to define tabbing behavior among widgets in an application.

“WIDGET_PROPERTYSHEET” — The WIDGET_PROPERTYSHEET function includes the following new keywords and event structure enhancements:

- EDITABLE keyword allows you to mark a property sheet as read-only or editable.
- FRAME keyword allows you to place a border around a property sheet.
- IGNORE_ACCELERATORS keyword allows the editable text area to receive keyboard combinations mapped to a WIDGET_BUTTON accelerator.

- `MULTIPLE_PROPERTIES` keyword enables the selection of multiple properties in a property sheet.
- `SUNKEN_FRAME` keyword allows the property sheet to appear indented.
- `WIDGET_PROPSHEET_SELECT` event structure includes new `NSELECTED` field and a modified `IDENTIFIER` field.

“WIDGET_SLIDER” — The `WIDGET_SLIDER` function includes the following new keyword:

- `TAB_MODE` allows you to define tabbing behavior among widgets in an application.

“WIDGET_TAB” — The `WIDGET_TAB` function includes the following new keyword:

- `TAB_MODE` allows you to define tabbing behavior among widgets in an application.

“WIDGET_TABLE” — The `WIDGET_TABLE` function includes the following new keywords:

- `CONTEXT_EVENTS` provides support for firing context events when right-clicking over a table widget.
- `IGNORE_ACCELERATORS` allows the editable cells to receive keyboard combinations mapped to a `WIDGET_BUTTON` accelerator.
- `NO_COLUMN_HEADER` allows you to design a table without column headers.
- `NO_ROW_HEADER` allows you to design a table without row headers.
- `TAB_MODE` allows you to define tabbing behavior among widgets in an application.

“WIDGET_TEXT” — The `WIDGET_TEXT` function includes the following new keywords:

- `IGNORE_ACCELERATORS` keyword allows the editable text area to receive keyboard combinations mapped to a `WIDGET_BUTTON` accelerator.
- `TAB_MODE` allows you to define tabbing behavior among widgets in an application.

“WIDGET_TREE” — The `WIDGET_TREE` function includes the following new keyword:

- `TAB_MODE` allows you to define tabbing behavior among widgets in an application.

“WRITE_TIFF” — The WRITE_TIFF routine includes the following new keywords:

- CMYK allows you to specify a TIFF file as one using the CMYK (cyan, magenta, yellow and black) color model.
- DESCRIPTION allows you to specify the ImageDescription tag value.
- DOCUMENT_NAME allows you to specify the DocumentName tag value.
- DOT_RANGE allows you to specify the DotRange tag value.
- ICC_PROFILE allows you to specify the ICC_PROFILE byte array.
- PHOTOSHOP allows you to specify the PHOTOSHOP byte array.
- XPOSITION allows you to specify the offset along the left side of the image.
- YPOSITION allows you to specify the offset along the top of the image.

“WV_DWT” — The WV_DWT function includes the following new keyword:

- N_LEVELS allows you to set the number of wavelet levels to compute in pyramid algorithm

Note

See [Chapter 4, “IDL Wavelet Toolkit Reference”](#) in the *IDL Wavelet Toolkit* manual for more information.

New IDL Object Classes

The following new object classes were added to IDL in this release. See the following topics in the *IDL Reference Guide* for complete reference information unless otherwise noted.

“IDL_Savefile” — A class that provides complete query and restore capabilities for IDL SAVE files (created with the SAVE procedure).

“IDLffJPEG2000” — A class that 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” — A class that finds and loads an XML language catalog.

“IDLffXMLDOM Classes” — A set of IDL classes that allow you to create a representation of an XML Document Object Model.

IDLffDicomEx — The new IDLffDicomEx object class provides the ability to read and write DICOM files including compressed data, and public and private attributes. See **“Expanded DICOM Support”** on page 39 for more information.

New IDL Object Properties

The following IDL object classes have new properties in this release. See the following topics in the *IDL Reference Guide* for complete reference information.

“IDLgrAxis” — The IDLgrAxis object includes the following new property:

- ALPHA_CHANNEL defines the transparency of the entire axis.

“IDLgrContour” — The IDLgrContour object features the following new property:

- ALPHA_CHANNEL defines the transparency of the contour.

“IDLgrPlot” — The IDLgrPlot object includes the following new property:

- ALPHA_CHANNEL defines the transparency of the plot.

“IDLgrPolygon” — The IDLgrPolygon object includes the following new properties:

- ALPHA_CHANNEL defines the transparency of the polygon.
- AMBIENT, DIFFUSE, EMISSION, SHININESS, and SPECULAR properties provide material definition capabilities.

“IDLgrPolyline” — The IDLgrPolyline object includes the following new property:

- ALPHA_CHANNEL defines the transparency of the polyline.

“IDLgrROI” — The IDLgrROI object includes the following new property:

- ALPHA_CHANNEL defines the transparency of the ROI.

“IDLgrSurface” — The IDLgrSurface object includes the following new properties:

- ALPHA_CHANNEL defines the transparency of the surface.
- AMBIENT, DIFFUSE, EMISSION, SHININESS, and SPECULAR properties provide material definition capabilities.

“IDLgrSymbol” — The IDLgrSymbol object includes the following new property:

- ALPHA_CHANNEL defines the transparency of the symbol.

“IDLgrVolume” — The IDLgrVolume object features the following new property:

- ALPHA_CHANNEL defines the transparency of the volume.

“IDLgrWindow” — The IDLgrWindow object features the following new properties:

- `MINIMUM_VIRTUAL_DIMENSIONS` identifies the minimum window canvas size.
- `ZOOM_BASE` defines the value by which the current zoom of the window is multiplied.
- `ZOOM_NSTEP` returns the number of times a window has been zoomed to reach the current zoom level.

“IDLitComponent” — The IDLitComponent object features the following new property:

- `COMPONENT_VERSION` identifies the version of the IDLitComponent object.

New IDL Object Methods

The following IDL object classes have new methods in this release. See the following topics in the *IDL Reference Guide* for complete reference information.

- “**IDLgrWindow::GetDimensions**” — This new method returns the visible dimensions of the IDLgrWindow object.
- “**IDLgrWindow::SetCurrentZoom**” — This new method enlarges, shrinks, or resets the IDLgrWindow object.
- “**IDLgrWindow::ZoomIn**” — This new method increases the size of the IDLgrWindow by a set amount.
- “**IDLgrWindow::ZoomOut**” — This new method decreases the size of the IDLgrWindow by a set amount.
- “**IDLitComponent::Restore**” — This new method performs any transitional work required after an object of this class has been restored from a SAVE file.
- “**IDLitComponent::UpdateComponentVersion**” — This new method updates the value of the COMPONENT_VERSION property for the specified object to match the version associated with the current release of IDL.
- “**IDLitContainer::FindIdentifiers**” — The new method allows you to retrieve the full identifiers for items within an iTool container.
- “**IDLitDataContainer::Add**” — This new method adds items to the data container object, and sends out onDataChange and onDataComplete messages to all observers of the data container.
- “**IDLitManipulator::RegisterCursor**” — This new method defines the appearance and name of a cursor associated with a custom manipulator.
- “**IDLitManipulatorManager::GetDefaultManipulator**” — This new method returns a reference to the manipulator that was most recently added as the default manipulator.
- “**IDLitParameter::GetParameterAttribute**” — This new method allows you to retrieve the value of an iTool visualization parameter attribute.
- “**IDLitParameter::QueryParameter**” — This new method allows you to retrieve the names of registered parameters of an iTool visualization.
- “**IDLitParameter::SetParameterAttribute**” — This new method allows you to set the value of an iTool visualization parameter attribute.
- “**IDLitTool::ActivateManipulator**” — This new method activates a manipulator that has been registered with this tool.

- “IDLitTool::FindIdentifiers”** — The new method allows you to retrieve the full identifiers for items within the tool container.
- “IDLitTool::RegisterCustomization”** — This new method registers an operation class that represents the graphics customization operation to be associated with this tool.
- “IDLitTool::RegisterStatusBarSegment”** — This new method registers a status bar segment with this tool.
- “IDLitTool::UnRegisterCustomization”** — This new method unregisters an operation class (that was previously registered as the graphics customization operation associated with this tool).
- “IDLitTool::UnRegisterStatusBarSegment”** — This new method unregisters a status bar segment (that was previously registered with this tool).
- “IDLitVisualization::BeginManipulation”** — This new method handles notifications that a manipulator is about to act on the visualization.
- “IDLitVisualization::EndManipulation”** — This new method handles notifications that a manipulator has finished acting on the visualization.
- “IDLitVisualization::GetRequestedAxesStyle”** — This new method returns the axes style requested by the visualization, if any.
- “IDLitVisualization::Move”** — This new method repositions an object, identified by its index number, within the container.
- “IDLitVisualization::On2DRotate”** — This methods handles notifications regarding changes to the rotation of the parent dataspace.
- “IDLitVisualization::OnAxesRequestChange”** — This method handles notifications when the axes of the contained object are changed.
- “IDLitVisualization::OnAxesStyleRequestChange”** — This method handles notifications when the axes style of the contained object is changed.
- “IDLitVisualization::OnDimensionChange”** — This method handles notifications when the dimensionality of the visualization’s data changes.
- “IDLitVisualization::OnWorldDimensionChange”** — This method handles notifications when the dimensionality of the visualization’s parent dataspace changes.
- “IDLitVisualization::RequestsAxes”** — This method indicates whether or not the visualization requests axes.
- “IDLitVisualization::Restore”** — This new method performs any transitional work required after an object of this class has been restored from a SAVE file.

- “**IDLitVisualization::Rotate**” — This method rotates the visualization about a given axis by a given angle.
- “**IDLitVisualization::SetAxesRequest**” — This method defines whether or not the current visualization requests axes.
- “**IDLitVisualization::SetAxesStyleRequest**” — This method defines the style of axes requested by the visualization.

IDL Object Property Enhancements

The following IDL object classes have enhanced properties in this release. See the following topics in the *IDL Reference Guide* for complete reference information.

- “**IDLgrContour**” — The C_COLOR property has been enhanced to support RGBA color definitions for object transparency.
- “**IDLgrPlot**” — The VERT_COLORS property has been enhanced to support RGBA color definitions for object transparency.
- “**IDLgrPolygon**” — The VERT_COLORS property has been enhanced to support RGBA color definitions for object transparency.
- “**IDLgrPolyline**” — The VERT_COLORS property has been enhanced to support RGBA color definitions for object transparency.
- “**IDLgrSurface**” — The VERT_COLORS property has been enhanced to support RGBA color definitions for object transparency.

IDL Object Method Enhancements

The following IDL object classes have enhanced methods in this release. See the following topics in the *IDL Reference Guide* for complete reference information.

“IDLgrBuffer::Select” — This method features the following new keyword:

- SUB_SELECTION returns objects contained in the selected object, select targets, and objects displayed at given coordinates.

“IDLgrClipboard::Draw” — This method features the following new keywords:

- CMYK specifies CMYK (cyan, magenta, yellow, and black) color mode.
- VECT_SHADING extends polygon shading capabilities in vector output.
- VECT_SORTING extends object sorting capabilities in vector output.
- VECT_TEXT_RENDER_METHOD extends text editing capabilities within vector output.

“IDLgrPrinter::Draw” — This method features the following new keywords:

- VECT_SORTING extends object sorting capabilities within vector output.
- VECT_TEXT_RENDER_METHOD extends text editing capabilities within vector output.

“IDLgrWindow::Select” — This method features the following new keyword:

- SUB_SELECTION returns objects contained in the selected object, select targets, and objects displayed at given coordinates.

“IDLitMessaging::ProgressBar” — This method features the following new keyword:

- CANCEL allows you to define the string used on the Cancel button of the progress bar.

“IDLitMessaging::StatusMessage” — This method features the following new keyword:

- SEGMENT_IDENTIFIER allows you to specify which segment of the status bar should be updated with the specified message.

“IDLitTool::DisableUpdates” — This method features the following new keyword:

- PREVIOUSLY_DISABLED returns 1 if the tool had previously been disabled, or 0 otherwise.

“IDLitTool::Register” — This method features the following new keyword:

- **DEFAULT** allows you to specify that the registered item is the default for its type.

“IDLitTool::RegisterFileReader” — This method features the following new keyword:

- **DEFAULT** allows you to specify that the registered item is the default for its type.

“IDLitTool::RegisterFileWriter” — This method features the following new keyword:

- **DEFAULT** allows you to specify that the registered item is the default for its type.

“IDLitTool::RegisterOperation” — This method features the following new keywords:

- **ACCELERATOR** specifies a string giving the keyboard accelerator to be used for the operation’s menu item.
- **CHECKED** indicates that a “checked” menu item should be used.
- **DISABLE** indicates the operation’s associated menu item should appear disabled (insensitive) when initially created.
- **DROPLIST_EDIT** specifies whether a droplist associated with an operation should be editable.
- **DROPLIST_INDEX** specifies the index of the initial selection for a droplist associated with an operation.
- **DROPLIST_ITEMS** specifies the names of items on a droplist associated with an operation.
- **SEPARATOR** indicates a menu separator should be placed before this operation.

“IDLitVisualization::Remove” — This method features the following new keyword:

- **NO_UPDATE** can be set to keep the scene from being updated after objects are removed.

“IDLitWindow::DoHitTest” — This method features the following new keyword:

- **ORDER** controls the order in which objects are returned in the hit test list when the objects exist at the same depth.

“IDLitWindow::GetSelectedItems” — This method features the following new keyword:

- *ALL* can be set to return all classes of selected items instead of just selected visualizations.

“IDLitWindow::OnKeyboard” — This method features several updated and new arguments:

- *IsAlphaNumeric* is now named *IsASCII*.
- *Character* and *KeySymbol* argument descriptions have been updated and expanded.
- *X*, *Y*, *Press*, *Release*, and *Modifiers* arguments are new in this release.

ION 6.1 Enhancements

ION (*IDL On the Net*) is a family of add-on modules for IDL. ION Script and ION Java are packages for publishing IDL-driven applications on the Web. They are included on the IDL CD as an optional feature. An extra-cost ION license is required to use ION Script and ION Java. For more information on ION, see “Introduction to ION” in the ION manual.

This section discusses the following new features and enhancements in ION 6.1:

Support for Secure HTTP (HTTPS)

ION Script now supports the use of the secure HTTP protocol (HTTPS). Aside from the need to modify URLs to use “https” rather than “http,” no changes are necessary to existing ION Script applications.

Features Obsoleted in IDL 6.1

The following features were present in IDL Version 6.0 but became obsolete in Version 6.1. These features have been replaced with a new keyword to an existing routine or by a new routine that offers enhanced functionality. These obsoleted features should not be used in new IDL code.

Obsolete Routines

The following routines have been replaced with new functionality:

Routine	Replaced By
FINDFILE	FILE_SEARCH
MSG_CAT_CLOSE	IDLffLangCat
MSG_CAT_COMPILE	IDLffLangCat
MSG_CAT_OPEN	IDLffLangCat
IDLffLanguageCat	IDLffLangCat

Obsolete Arguments or Keywords

The arguments or keywords to the following methods have been removed:

Routine	Argument or Keyword
IDLITSYS_CREATETOOL	PANEL_LOCATION keyword
IDLitVisualization::Add	GROUP keyword
IDLitVisualization::GetCenterRotation	DATA keyword
IDLitVisualization::GetProperty	GROUP_PARENT keyword

Avoiding Backward Compatibility Issues

Although RSI strives to maintain backward compatibility with previous versions, some enhancements can require changes to your code. In IDL 6.1, several event structures have been enhanced with new fields. If you have a SAVE file accessing instances of these structures created in a previous release, use the `RELAXED_STRUCTURE_ASSIGNMENT` keyword to `RESTORE` to avoid errors when restoring the SAVE file.

Specifically, the event structure fields have changed as follows:

- `QUERY_TIFF` info structure — see [“New QUERY_TIFF Info Structure Fields”](#) on page 41
- `WIDGET_CONTEXT` event structure — The `WIDGET_CONTEXT` event structure associated with base, list, property sheet, table, text and tree widgets has been updated with new `ROW` and `COL` fields.
- `WIDGET_PROPSHEET_SELECT` event structure — The `WIDGET_PROPSHEET_SELECT` event structure has a new `NSELECTED` field so support multiple property selection in a property sheet.

Requirements for this Release

IDL 6.1 Requirements

Hardware Requirements for IDL 6.1

The following table describes the supported platforms and operating systems for IDL 6.1:

Platform	Vendor	Hardware	Operating System	Supported Versions
Windows	Microsoft	Intel x86 ^c	Windows	2000, XP
Macintosh ^{b,c}	Apple	PowerMac G4, G5 ^c	OS X	10.3
UNIX ^c	HP	PA-RISC 32-bit	HP-UX	11.0
	HP	PA-RISC 64-bit ^a	HP-UX	11.0
	IBM	RS/6000 32-bit	AIX	5.1
	IBM	RS/6000 64-bit ^a	AIX	5.1
	Intel	Intel x86 ^c	Linux ^d	Red Hat 7.1, 9, Enterprise 3.x ^{b,f}
	SGI	Mips 32-bit	IRIX	6.5.1
	SGI	Mips 64-bit ^a	IRIX	6.5.1
	SUN	SPARC 32-bit ^c	Solaris	8, 9
	SUN	SPARC 64-bit ^a	Solaris	8, 9

Table 1-5: Hardware Requirements for IDL 6.1

On platforms that provide 64-bit support, IDL can be run as either a 32-bit or a 64-bit application. When both versions are installed, the 64-bit version is the default. The 32-bit version can be run by specifying the `-32` switch at the command line, as follows:

```
% idl -32
```

^a The DXF file format and IDL DataMiner are not supported on 64-bit IDL platforms.

^b IDL DataMiner is not supported on Mac OS X or Red Hat Enterprise 3.x.

^c For UNIX and Mac OS X, the supported versions indicate that IDL was either built on (the lowest version listed) or tested on that version. You can install and run IDL on other versions that are binary compatible with those listed.

^d IDL 6.1 was built on the Linux 2.4 kernel with `glibc 2.2` using Red Hat Linux. If your version of Linux is compatible with these, it is possible that you can install and run IDL on your version.

^e IDLffDicomEx is supported on Windows, Mac OS X, Linux, and SPARC 32-bit Solaris.

^f For Red Hat Enterprise users: Some functionality in this IDL release requires the `libstdc++` compatibility libraries to be installed. To check if the libraries are already installed, issue the following command:

```
rpm -qa | grep compat
```

and look for `compat-libstdc++-<version_number>`. If this is not listed, install the appropriate RPM for your distribution (from your installation CD-ROMs).

Software Requirements for IDL 6.1

The following table describes the software requirements for IDL 6.1:

Platform	Software Requirements
Windows	Internet Explorer 5.0 or higher.
Macintosh	Apple X11 X-Windows manager

Table 1-6: Software Requirements for IDL 6.1

ION 6.1 Requirements

Hardware Requirements for ION 6.1

The following table describes the supported platforms and operating systems for ION 6.1::

Platform	Vendor	Hardware	Operating System	Supported Versions
Windows	Microsoft	Intel x86	Windows	2000, XP
UNIX ^a	Intel	Intel x86	Linux ^b	Red Hat 7.1, 9, Enterprise 3.x ^c
	SGI	Mips 32-bit	IRIX	6.5.1
	SUN	SPARC 32-bit	Solaris	8, 9

Table 1-7: Hardware Requirements for ION 6.1

^a For UNIX, the supported versions indicate that ION was either built on the lowest version listed or tested on that version. You can install and run ION on other versions that are binary compatible with those listed.

^b ION 6.1 was built on the Linux 2.4 kernel with `glibc 2.2` using Red Hat Linux. If your version of Linux is compatible with these, it is possible that you can install and run ION 6.1 on your version.

^c For Red Hat Enterprise users: Some functionality in this IDL release requires the `libstdc++` compatibility libraries to be installed. To check if the libraries are already installed, issue the following command:

```
rpm -qa | grep compat
```

and look for `compat-libstdc++-<version_number>`. If this is not listed, install the appropriate RPM for your distribution (from your installation CD-ROMs).

Web Server Requirements for ION 6.1

In order to use ION, you must install an HTTP Web server. ION has been tested with the following Web server software:

- Apache Web Server version 2.0 for Windows, Linux, and Solaris.
- Apache Web Server version 1.3.14 for IRIX. This version is included with the IRIX operating system.
- Microsoft Internet Information Server (IIS) version 5.0 for Windows 2000 Server and version 5.1 for Windows XP Professional.

If you do not already have Web server software, the IDL 6.1 CD-ROM contains the following Apache Web Server software:

- Windows — Version 2.0.45
- Linux — Version 2.0.43
- Solaris — Version 2.0.43
- IRIX — Version 1.3.14

Note

For more information on Apache software for your platform, see <http://www.apache.org>.

Web Browser Requirements for ION 6.1

ION 6.1 supports the HTTP 1.0 protocol. The following are provided as examples of popular Web browsers that support HTTP 1.0:

- Netscape Navigator versions 4.7 and 6.0
- Microsoft Internet Explorer versions 5.5 and 6.0

Browsers differ in their support of HTML features. As with any Web application, you should test your ION Script or Java application using Web browsers that anyone accessing your application is likely to be using.

Java Virtual Machine Requirements for ION 6.1

The following are provided as examples of popular Web browsers that are shipped with the required JVMs:

- Netscape Navigator versions 4.7 and 6.0
- Microsoft Internet Explorer versions 5.5 and 6.0

Browsers differ in their support of Java features. As with any Web application, you should test your ION Java application using Web browsers that anyone accessing your application is likely to be using.

IDL-Java Bridge

IDL now supports the use of Java objects. You can access Java objects within your IDL code using the IDL-Java bridge, a built-in feature of IDL 6.1. The IDL-Java bridge enables you to take advantage of special Java I/O, networking, and third party functionality.

The IDL-Java bridge is installed by default in a standard IDL installation. See [Chapter 8, “Using Java Objects in IDL”](#) in the *External Development Guide* manual for details.

Hardware Requirements for the IDL-Java Bridge

The following table describes the platforms and operating systems for the IDL-Java bridge:

Platform	Vendor	Hardware	Operating System	Supported Versions
Windows	Microsoft	Intel x86	Windows	2000, XP
Macintosh ^a	Apple	PowerMac G4	OS X	10.3
UNIX ^a	Intel	Intel x86	Linux ^b	Red Hat 7.1, 9, Enterprise 3.x ^c
	SGI	Mips 32-bit	IRIX	6.5.1
	SUN	SPARC 32-bit	Solaris	8, 9
	SUN	SPARC 64-bit	Solaris	8, 9

Table 1-8: Hardware Requirements for the IDL-Java Bridge

On platforms that provide 64-bit support, IDL can be run as either a 32-bit or a 64-bit application. When both versions are installed, the 64-bit version is the default. The 32-bit version can be run by specifying the `-32` switch at the command line, as follows:

```
% idl -32
```

^a For UNIX and Mac OS X, the supported versions indicate that IDL was either built on (the lowest version listed) or tested on that version. You can install and run IDL on other versions that are binary compatible with those listed.

^b IDL 6.1 was built on the Linux 2.4 kernel with `glibc 2.2` using Red Hat Linux. If your version of Linux is compatible with these, it is possible that you can install and run IDL on your version.

^c For Red Hat Enterprise users: Some functionality in this IDL release requires the `libstdc++` compatibility libraries to be installed. To check if the libraries are already installed, issue the following command:

```
rpm -qa | grep compat
```

and look for `compat-libstdc++-<version_number>`. If this is not listed, install the appropriate RPM for your distribution (from your installation CD-ROMs).

Java Virtual Machine Requirements for the IDL-Java Bridge

IDL supports version 1.3.1 and greater on all platforms with the following exceptions:

- The supported version on Macintosh is 1.3.x
- SUN SPARC 64-bit has only version 1.4.x and greater



Chapter 2: Library Authoring

The following topics are covered in this chapter:

Overview of Library Authoring	90	Advice for Library Authors	93
Recognizing Potential Naming Conflicts . . .	91	Converting Existing Libraries	94

Overview of Library Authoring

Library authors provide an invaluable resource to the IDL community — they develop domain-specific programs and applications that implement knowledge far beyond RSI’s level of expertise. User library code is often freely available, supported, and documented. However, as the number of library authors and routines continues to grow, it becomes increasingly important for authors to adhere to a routine naming convention within their libraries that avoids conflicts with core IDL functionality.

Most user libraries start out as small collections of code, and then grow. Initially, the naming issue is not very important. Over time, the library grows in complexity and number of users. Because this is often a gradual process, the importance of naming is not obvious until there is a conflict with IDL system functionality, or a conflict with another library author’s code.

An understanding of the way IDL resolves routines during program execution reveals why new IDL system procedures and functions may periodically conflict with pre-existing routines written by users in the IDL community. (See [“How IDL Resolves Routines”](#) on page 91 for step-by-step routine resolution details.)

The fact that IDL system routines always take precedence over user routines provides the following benefits:

- The IDL environment remains reliable and consistent — a call to FFT always returns the IDL version of the FFT function.
- It eliminates a great deal of path searching, which translates into faster execution speed.

In contrast, if user routines took precedence over system routines, a given installation could radically alter the meaning of common and basic IDL constructs simply by creating user routines with the names of IDL system routines. This would result in conflicts when sharing code, degradation of the common IDL language core, and ultimately, the reduced usefulness of IDL.

Although the way IDL handles the search for routines is simple, efficient, and reliable, it is not perfect. The potential for namespace conflicts exists. It is important to recognize and take steps to avoid these naming conflicts as described in the following sections:

- [“Recognizing Potential Naming Conflicts”](#) on page 91
- [“Advice for Library Authors”](#) on page 93
- [“Converting Existing Libraries”](#) on page 94

Recognizing Potential Naming Conflicts

IDL favors simple names, and it blurs the user level distinction between system routines and user routines. The reason for this has everything to do with IDL's orientation towards *ad hoc* analysis. The primary goal is transparency. Names should make sense, be easy to remember, and not require too much typing. Language transparency also results in very human-readable code. In conjunction with the way IDL searches for routines, this may cause either user level or system level conflicts.

User Level Conflicts

In the user level case, an IDL user writes a routine that is not part of the base release of IDL, and places it in a local library. At some later date, a new version of IDL is installed that contains a new IDL library routine with the same name as the user's routine. Depending on the order of the directories in the user's path, one of these two routines is executed. If the user's routine is used, IDL library code that calls the routine will get the wrong version and fail in strange and mysterious ways. If the IDL routine is used, the IDL library will be satisfied, but the user's library will get the wrong version, also with bad results.

System Level Conflicts

The system level case is similar, but harder to work around. In this case, the user creates a local routine, as before. However, the new version of IDL contains a system routine with the same name. In this case, IDL will always choose to use the system routine, and the user routine simply vanishes from view never to be called again. The order of the search path is meaningless in this case because the search path is not even consulted. A system routine always has precedence over a user routine.

Choosing Routine Names to Avoid Conflicts

Naming conflicts can result in costly and time consuming problems; carefully considered names make everything easier. On the surface, naming routines seems like a trivial issue, but names are very important. It is crucial to adopt and consistently adhere to a routine naming strategy to avoid conflict. The core idea of this convention (described in detail in [“Advice for Library Authors”](#) on page 93) is to prefix all library routine names with a unique identifier, one indicative of your organization or project. Research Systems reserves routine names that are generic, and those with an “IDL” or “RSI” prefix on behalf of the entire IDL community. Prefixing your user library routines significantly reduces the risk of namespace collisions with IDL routines.

As a library author, your decision to follow a routine prefixing strategy benefits the entire IDL community. This convention translates into simplicity and reliability, allowing IDL system routines to always take precedence over user routines. It also raises the visibility of your routines, readily distinguishing them as part of your library.

Note

For instructions on how to prefix an existing user library, see [“Converting Existing Libraries”](#) on page 94.

Advice for Library Authors

An ordinary IDL programmer needs only to solve his or her own problems to the desired level of quality, reusability, and robustness. Life is more difficult for an author of a library of IDL routines. In addition to the challenges facing any programmer, library authors face additional challenges:

- The structure and organization of the library needs to encourage reuse and generality.
- Library code must be more robust than the usual program. Stability of implementation, and especially of interface, are very important.
- Errors must be gracefully handled whenever possible. See [Chapter 19, “Controlling Errors”](#) for more on error control.
- The most useful libraries are written to work correctly on a wide variety of platforms, without requiring their users to be aware of the details.
- Documentation must be provided, or the library will not find users.
- Libraries must be able to co-exist with other code over which they have no control. Authors must not alter the global environment in ways that cause conflicts, and they must also take care to prefix the names of all routines, common blocks, systems variables, and any other global resources they use. This prevents a library from conflicting with other libraries on the same system, and protects the library from changes to IDL that may occur in newer releases.

Prefixing Routine Names

The use of a proper prefix minimizes the risk of a namespace collision as described in [“Recognizing Potential Naming Conflicts”](#) on page 91. In selecting a prefix for your library, you should select a name that is short, mnemonic, and unlikely to be chosen by others. For example, such a name might use the name of your organization or project in an abbreviated form.

Non-prefixed names and names prefixed by “IDL” or “RSI” are reserved by RSI. New names of these forms can and will appear without warning in new versions of IDL, and should be avoided when naming new library routines.

Converting Existing Libraries

Many libraries that already exist do not follow the naming guidelines provided in “[Advice for Library Authors](#)” on page 93. Such libraries are bound to experience an occasional conflict with new versions of IDL. The best solution to avoid conflicts is to perform a systematic one-time conversion to a prefixed naming scheme.

Any existing library is likely to already have users. Assuming that non-prefixed names were used in such libraries, it is not possible to simply change the names. Such conversions require time to carry out, and once that has happened, it takes time for users to adjust and alter their usage. However, the actual conversion can go very quickly, and with proper planning it is easy to offer a backwards compatibility option for your users. Use the following steps to convert an existing library:

1. Generate a list of all files containing routines to be renamed.
2. Using this list, build an IDL batch file that uses `.COMPILE` on each file.
3. Start a fresh IDL session, execute the batch file, and use `HELP, /ROUTINES` to get a complete list of all compiled routines. Only IDL user library routines (those `.PRO` files shipped with the IDL distribution) should not contain a prefix.
4. As you rename each routine to its prefixed form, write a non-prefixed wrapper routine with the old name that calls the new version. Such wrappers are easy to write in IDL, using the `_REF_EXTRA` keyword to pass keywords through to the real routine. See “[Keyword Inheritance](#)” on page 81 for details.
5. Use the `COMPILE_OPT OBSOLETE` compilation directive in such wrappers so that IDL will recognize them as obsolete routines. See “[Setting Compilation Options](#)” on page 96 and `COMPILE_OPT` in the *IDL Reference Guide* for more information on `COMPILE_OPT`. These compatibility wrappers serve the following purposes:
 - You can use them to migrate your library to fully prefixed form over time, since the wrapper will be used any place you failed to change to calling the new name. This enhances the stability of the library and gives you time to do a careful job.
 - Once you are finished, you can provide them to your customers as a bridge, so that their old code continues to work.
 - As you change the names of routines, use `grep` (or a similar file searching tool) to locate uses of that name, and convert them to the new form as well.

6. Iterate, using the batch file mentioned above to find any remaining non-prefixed uses of the library names. Since your wrappers specified the `COMPILE_OPT OBSOLETE` directive, you can set the `!WARN` system variable to help you pinpoint such uses. You are done when your batch file reveals no more unprefixed names.

Once the conversion is done, you can use the compatibility wrappers to smoothly transition your users to the new names. You should keep the wrappers in a separate subdirectory, and even consider making them optional. Doing this raises the end user's awareness of the issue and may convince them to convert to using the new names sooner rather than later.

When you add new routines to your library, ensure that they use the proper prefix. Do not provide non-prefixed wrapper routines for new routines. There is no backward compatibility issue in this case, and they are not needed.

Although the one time hit of prefixing an existing library can consume some time and effort, there are benefits that accrue from doing it. When new versions of IDL are released, the odds of the library working with the new version without encountering any name clashes are extremely high. Use of a consistent prefix also raises the profile of the library to the end user, raising their level of understanding and appreciation for the work it does.



Chapter 3: Using Language Catalogs

The following topics are covered in this chapter:

What Is a Language Catalog?	98	Using the IDLffLangCat Class	101
Creating a Language Catalog File	99	Widget Example	104

What Is a Language Catalog?

A *language catalog* is a set of text strings in a particular language, created as key name/value pairs. Applications can use these catalogs to fill in the names of menu items, buttons, and other elements of a user interface, for example. The use of different language catalogs, then, can support an application's internationalization: for example, letting a user decide what language to use for installing and running the application.

There are two main advantages of using a language catalog in a separate file, rather than having these strings embedded in the application:

- The strings do not consume memory until the application loads them
- You can edit the catalog to add new languages and new strings without directly involving the application

In addition, because the language catalogs are in XML format, you can easily read and edit the files in any text editor.

Implementing language catalog functionality requires two parts:

- The creation of a language catalog (`.cat`) file, which contains the name/value pairs in the desired languages. See [“Creating a Language Catalog File”](#) on page 99 for information on requirements for a language catalog file's structure.
- The creation of an `IDLffLangCat` object, which provides access and use of the keys in the catalog file. See [“Using the IDLffLangCat Class”](#) on page 101 for information on creating and using a language catalog object.

Creating a Language Catalog File

A language catalog (.cat) file contains the XML that defines the text strings as key name/value pairs within a single <IDLffLangCat> tag. The tag can contain four optional attributes, as described in [Table 3-1](#).

Attribute	Description
APPLICATION	Name of the application that will use the keys in the file
VERSION	Version of IDLffLangCat for which the file was created
DATE	Date of the file's creation or last modification, as desired
AUTHOR	Author of the file

Table 3-1: IDLffLangCat Tag Attributes

Note

You cannot perform queries on VERSION, DATE, and AUTHOR. These attributes are more like XML comments on the tag; they are informational only.

The following XML snippet, extracted from the iTools menu catalog file that comes with the IDL installation, illustrates the basic file structure:

```
<IDLffLangCat APPLICATION="itools menu" VERSION="1.0"
  AUTHOR="RSI" >
  <LANGUAGE NAME="English" >
    <KEY NAME="Menu:File">File</KEY>
    <KEY NAME="Menu:File:New">New</KEY>
    <KEY NAME="Menu:File:Open">Open...</KEY>
  </LANGUAGE>
</IDLffLangCat>
```

The <IDLffLangCat> tag can contain any number of LANGUAGE tags. Each LANGUAGE tag must have a NAME attribute denoting the language contained therein.

Each LANGUAGE tag can contain any number of KEY tags. Each KEY tag must have a NAME attribute denoting the name of the key.

Note

All text between the open and close `KEY` tags will be part of the string returned by the query, including any line feeds, carriage returns, and spaces.

The catalog file can contain keys for one or more languages. Whether there is a single catalog file containing multiple languages, or multiple catalog files, each containing a single language, is personal preference.

By keeping each language separate in the tag definition, you can easily cut and paste an entire block and then change the strings of one language to another language while keeping all the keys intact. This technique also allows for the possibility of having different languages in separate files. Note that the keys in any one language need not match those of another language (although in most cases they will).

Note

IDL supports catalog files written in 8-bit Latin alphabet languages. Also, you must have the corresponding fonts installed on your machine before you can use a particular language.

Storing and Loading Language Catalog Files

The catalog files included with IDL are in the `/resource/langcat` directory of the IDL installation and end in a `.cat` extension. These files contain the English keys for iTools menus, dialogs, and messages and are provided to support the use of applications using iTools functionality in other languages. All catalog files must end with the `.cat` extension if `APP_NAME` is used to locate the files.

You can create custom catalog files and place them in a location of your choice. You typically use a full path to access these catalog files through the creation of an `IDLffLangCat` object (see “[Using the IDLffLangCat Class](#)” on page 101 for more information).

You can specify a catalog either by giving the full path of the catalog file or files, or by providing an application name or names and, optionally, an application path or paths. If no path is specified, only the current directory is searched. For all application paths, all `.cat` files found in any of the directories listed are searched for all given applications.

On a similar note, if IDL finds a duplicate key name while loading keys, IDL will use the string corresponding to the last key found with the given name.

Using the IDLffLangCat Class

You use the IDLffLangCat class to find and load an XML language catalog. The class also provides methods for retrieving text strings by matching key names.

Creating a Language Catalog Object

The IDL installation comes with an English language catalog for the iTools menu, called `itoolsmenu_eng.cat`, in the `/resource/langcat` directory of the IDL installation. To load the keys in this file:

```
oLangCat = OBJ_NEW( 'IDLffLangCat', 'ENGLISH', $
    APP_NAME='itools menu', $
    APP_PATH=FILEPATH('', $
    SUBDIRECTORY=['resource','langcat','itools']), /VERBOSE )
```

This command searches the given directory for language catalog keys in English that match the application of ‘itools menu.’ In fact, if there are any other language catalog files, besides `itoolsmenu_eng.cat`, containing keys whose `LANGUAGE` value is ‘ENGLISH’ and `APPLICATION` value is ‘itools menu,’ the object adds those keys as well. The matches must be exact in that ‘itools menu2,’ for example, is not a match; however, the matching is not case-sensitive (i.e., ‘ENGLISH’ and ‘English’ are both matches for `LANGUAGE`).

Note

Whenever the object encounters a key (or language) that already exists, the key (or language) is overwritten with the new value.

The `VERBOSE` flag on the command sends all catalog-loading messages to the IDLDE output window. This list contains details resulting from the object’s initialization (the names and numbers of keys loaded, keys overwritten, etc.).

Adding Application Keys

You might want to add keys for a different application to an existing language catalog object. To do so:

```
retval = oLangCat->AppendCatalog( APP_NAME='itools ui', $
    APP_PATH=FILEPATH('', $
    SUBDIRECTORY=['resource','langcat','itools']) )
```

This command searches the given directory for keys matching an `APPLICATION` value of ‘itools ui’ and appends them to `oLangCat`. The method returns a value indicating success or failure of the operation.

Getting and Setting Languages

To return the available languages in a language catalog object:

```
oLangCat->GetProperty, AVAILABLE_LANGUAGES=availLangs
```

This command stores the list of available languages as a string array in `availLangs`.

To set the current language of a language catalog object (the language used for query searches and matching):

```
oLangCat->SetProperty, LANGUAGE='English'
```

You can use these two methods for getting and setting other properties of a language catalog object. For the list of available object properties, see “[IDLffLangCat Properties](#)” in the *IDL Reference Guide* manual.

Comparisons such as those done with the Query method (see “[Performing Queries](#)” on page 102) are case insensitive, but the values returned by the GetProperty method are exactly as the last encountered value. The exception is that all key names are returned in uppercase. For example, if File 1 has LANGUAGE='English' and File 2 has LANGUAGE='engLIsh', then 'engLIsh' will be returned, although only one ENGLISH language exists in the current catalog.

Performing Queries

To populate the text fields of a widget or other interface object, for example, you can query a language catalog object for key values it contains. IDL performs the search on the NAME attribute of the keys; matches are not case-sensitive.

```
keyVal = oLangCat->Query( 'Menu:File:New', $
    DEFAULT_STRING='Key not found' )
```

This command searches `oLangCat` for keys with the NAME value of ‘Menu:File:New’ and returns the match in `keyVal`. If `oLangCat` finds a match in the current language, `keyVal` will hold that value string. If a given key does not exist in the current language, the default language is queried (if it exists). If there are still no matches, the default string is returned.

You can use more than one key in a query by passing an array of strings to the Query method (e.g., ['Menu:File:New', 'Menu:File:Open']). Similarly, you can supply an array of strings for the DEFAULT_STRING keyword. In such a case, only those values in the array whose indices match the missing keys will be returned. If you do not specify DEFAULT_STRING, a null string will be returned instead.

Destroying a Language Catalog Object

You can destroy a catalog object as you would any other IDL object, as follows:

```
OBJ_DESTROY, oLangCat
```

Destroying a language catalog object does not affect any files from which the object drew its keys.

Widget Example

This example creates a widget with two buttons whose text strings change between two languages, depending on the selection from a drop-down list.

The following language catalogs are two separate files (as denoted by the `<IDLffLangCat>` tag for each) and should be placed on your system as such.

```
<?xml version="1.0"?>
<!-- $Id: myButtonsText.eng.cat,v 1.1 2004 rsiDoc Exp $ -->
<IDLffLangCat APPLICATION="myOpenButtons" VERSION="1.0"
  AUTHOR="RSI">
  <LANGUAGE NAME="English">
    <KEY NAME="Button:OpenFile">Open File</KEY>
    <KEY NAME="Button:OpenFolder">Open Folder</KEY>
  </LANGUAGE>
</IDLffLangCat>

<?xml version="1.0"?>
<!-- $Id: myButtonsText.fr.cat,v 1.1 2004 rsiDoc Exp $ -->
<IDLffLangCat APPLICATION="myOpenButtons" VERSION="1.0"
  AUTHOR="RSI">
  <LANGUAGE NAME="French">
    <KEY NAME="Button:OpenFile">Ouvrir le Fichier</KEY>
    <KEY NAME="Button:OpenFolder">Ouvrir le Dossier</KEY>
  </LANGUAGE>
</IDLffLangCat>
```

To use the following code, save it in a `.pro` file. You do not have to run it from the same directory containing the language catalog files.

```
; Routine to change the language of the button labels.
PRO button_language_change, pstate
  vLangString = (*pstate).vlang

  ; Access the language catalog to retrieve string values.
  oLangCat = OBJ_NEW( 'IDLffLangCat', vLangString, $
    APP_NAME='myOpenButtons' , APP_PATH=(*pstate).vpath)
  ; Access and store language-specific strings in the structure.
  strOpenFile = oLangCat->Query( 'Button:OpenFile' )
  strOpenFolder = oLangCat->Query( 'Button:OpenFolder' )
  WIDGET_CONTROL, (*pstate).pb1, SET_VALUE=strOpenFile
  WIDGET_CONTROL, (*pstate).pb2, SET_VALUE=strOpenFolder
END

; Event handler for 'Open File' button.
PRO button_file, event
  sFile = DIALOG_PICKFILE( TITLE='Select image file' )
```



```

END

; Event handler for 'Open Folder' button.
PRO button_folder, event
    sFolder = DIALOG_PICKFILE( /DIRECTORY, $
        TITLE='Choose the directory in which to store the data' )
END

; Event handler for 'Language' droplist.
PRO button_language_event, event
    WIDGET_CONTROL, event.top, GET_UVALUE = pstate
    ; Access user's language selection and store it in the pointer.
    IF event.index EQ 0 THEN (*pstate).vlang = 'English'
    IF event.index EQ 1 THEN (*pstate).vlang = 'French'
    ; Call the procedure to change the button text.
    button_language_change, pstate
END

; Widget-creation procedure
PRO button_language
    ; Prompt for path to catalog files
    vpath=dialog_pickfile( TITLE='Select directory that ' + $
        'contains *.cat files', /DIRECTORY )
    IF vpath EQ '' THEN return

    ; Create a top level base. Not specifying tab mode uses default
    ; value of zero (do not allow widgets to receive or lose focus).
    tlb = WIDGET_BASE( /COLUMN, TITLE = "Language Change", $
        XSIZE=220, /BASE_ALIGN_CENTER )
    ; Create the button widgets.
    bbase = WIDGET_BASE( tlb, /COLUMN )
    pb1 = WIDGET_BUTTON( bbase, VALUE='Open File', $
        UVALUE='openFile', XSIZE=105, EVENT_PRO='button_file' )
    pb2 = WIDGET_BUTTON( bbase, VALUE='Open Folder', $
        UVALUE='openFolder', XSIZE=105, EVENT_PRO='button_folder' )
    ; Create a drop-down list indicating available catalogs.
    vLangList = ['English', 'French']
    langDrop = WIDGET_DROPLIST( tlb, VALUE=vLangList, $
        TITLE='Language' )
    ; Draw the widgets and activate events.
    WIDGET_CONTROL, tlb, /REALIZE

    ; Create the state structure.
    state = { $
        pb1:pb1, $
        pb2:pb2, $
        vlang:'', $
        vpath:vpath $
    }

```

```
pstate = PTR_NEW( state, /NO_COPY )
WIDGET_CONTROL, tlb, SET_UVALUE=pstate
XMANAGER, 'button_language', tlb

; Clean up pointers.
PTR_FREE, pstate
END
```



Chapter 4:

Using the XML DOM Object Classes

The following topics are covered in this chapter:

About the Document Object Model	108	Using the XML DOM Object Classes . . .	117
About the XML DOM Object Classes . . .	111	Tree-Walking Example	123

About the Document Object Model

The Document Object Model (DOM) describes the content of XML data in the form of a document object, which contains other objects that describe the various data elements of the XML document. The DOM also specifies an interface for interacting with the objects in the model. This is the interface exposed to the IDL user.

For more information on XML, see [“About XML”](#) on page 628.

When to Use the DOM

There are two basic types of parsers for XML data: object-based and event-based. The DOM is object-based and as such has advantages in certain situations over an event-based parser such as SAX. In general, use the DOM:

- To access an XML document in any order (SAX must parse in file order)
- To write to a file (SAX does not support modifying or creating XML data)

For more information on the difference between the two parsers, see [“About XML Parsers”](#) on page 628.

About the DOM Structure

Here is an example of an XML file that is used in an application to define a weather-monitoring plug-in component:

```
<?xml version="1.0" encoding="UTF-8"?>
<plugin type="tab-iframe">
  <name>Weather.com Radar Image [DEN]</name>
  <description>600 mile Doppler radar image for DEN</description>
  <version>1.0</version>
  <tab>
    <icon>weather.gif</icon>
    <tooltip>DEN Doppler radar image</tooltip>
  </tab>
</plugin>
```

The contents of this file constitute an XML document. When you want to work with this data, you can use IDL to load the file, parse it, and store it in memory in DOM format. The sample file listed above is stored in the DOM structure as shown in [Figure 4-1](#).

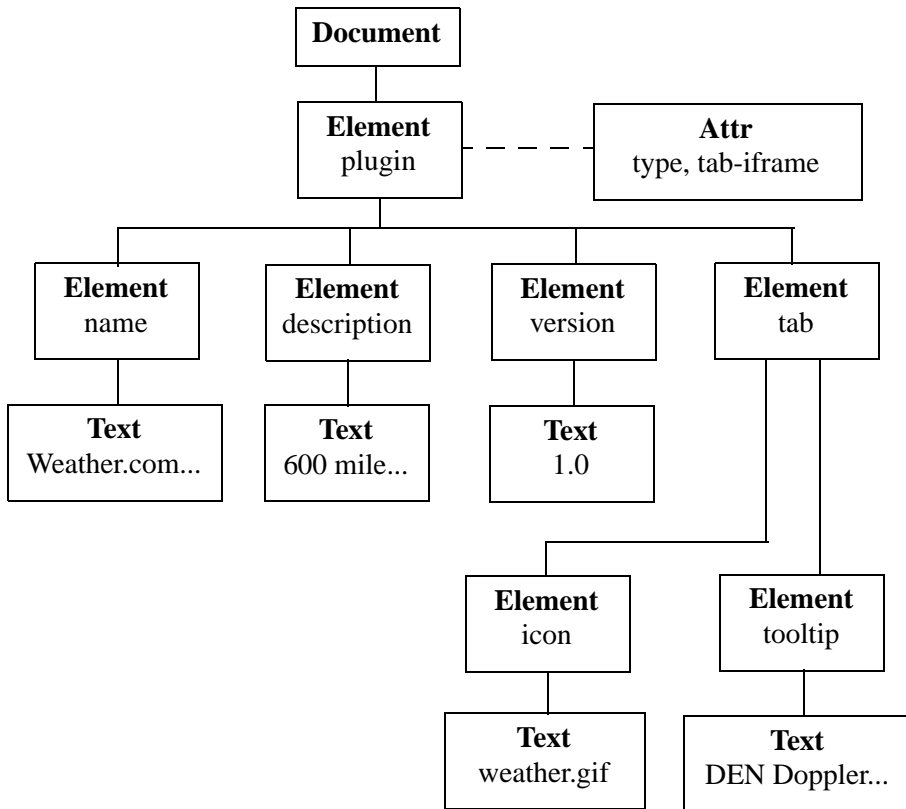


Figure 4-1: XML DOM Tree Structure: Plug-in Example

The DOM structure is a tree of nodes, where each node is represented as a box in the figure. The type of each node is in **boldface**. The contents of the node are in normal type.

Note that whitespace and newline characters can appear in this tree as text nodes, but are omitted in this picture for clarity. It is important to keep this in mind when exploring the DOM tree. There are parsing options available that can prevent the creation of ignorable-whitespace nodes (see [“Working with Whitespace”](#) on page 121).

The attribute node (Attr) is not actually a child of the element node, but is still associated with it, as indicated by the dotted line.

How IDL Uses the DOM Structure

To access the XML data in the structure, you need to create a set of IDL objects that correspond to the portion of the DOM tree in which you are interested. You use the following process to create the DOM tree and the corresponding IDL objects:

1. Create an `IDLffXMLDOMDocument` object.
2. Load the XML file. This step parses the XML data from the file and creates the DOM tree in memory.
3. Use the `IDLffXMLDOMDocument` object to create IDLffXMLDOM objects that essentially mirror portions of the DOM tree, as shown in [Figure 4-2](#).

You then use the IDLffXMLDOM objects to access the actual XML data contained in the DOM tree.

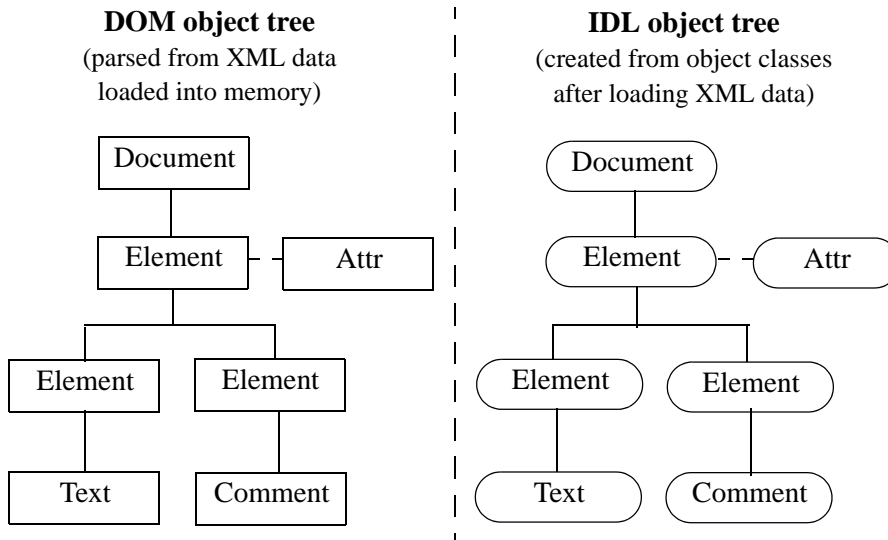


Figure 4-2: The DOM and IDL Trees

The creation and destruction of the IDL objects do not alter the DOM structure. There are explicit methods for modifying the DOM structure. The IDL objects are merely access objects that are used to manipulate the DOM tree nodes.

About the XML DOM Object Classes

The IDL XML DOM support is provided by a set of IDL object classes, all starting with *IDLffXMLDOM*. These classes provide access to the XML document via the DOM. The IDLffXMLDOM objects do not in themselves maintain a copy of the document data. Instead, they provide access to the data stored in the DOM document structure.

IDLffXMLDOMNode Class Hierarchy

One of the key object classes is [IDLffXMLDOMNode](#). Because it is an abstract class, you will never create an instance of this class. The node is the basic DOM data structure used to map each DOM data element. The nodes are organized in a classic tree structure, according to the layout of the data in the document.

The following classes are derived from [IDLffXMLDOMNode](#), where each class is named *IDLffXML<node type>* (e.g., [IDLffXMLDOMAttr](#)):

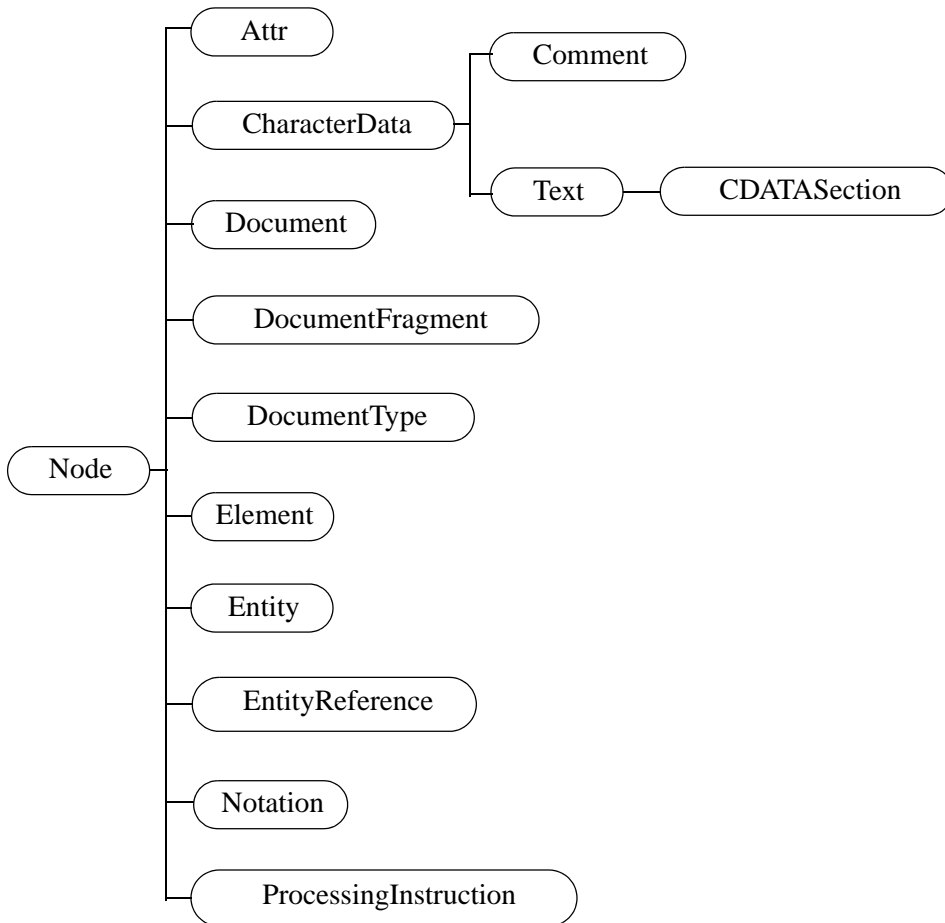


Figure 4-3: The IDLffXMLDOMNode Class Hierarchy

These classes represent the data that can be stored in an XML document. Except for the [IDLffXMLDOMDocument](#) class, you do not instantiate any of them directly. To begin working with the IDL XML DOM interface, you use the OBJ_NEW function to create an [IDLffXMLDOMDocument](#) object. You then use this object to browse and modify the document. This document object also creates objects using the derived classes to give you access to the various parts of the document. For example:

```
oChild = oMyDOMDocument->GetFirstChild()
```

creates an IDL object of one of the node types, depending on what the first child in your document actually is. The newly created IDL object refers to the first child node of the document and does not modify the document in any way.

You then use the `oChild` object's methods to get data from the node, modify the node, or find another node.

Because of the class hierarchy, all the methods in a superclass are available to its subclasses. For example, to determine which methods are available for use by an object of the `IDLffXMLDOMText` class, you would have to look at the methods belonging to the `IDLffXMLDOMText`, `IDLffXMLDOMCharacterData`, and `IDLffXMLDOMNode` classes.

Note

The `IDLffXMLDOMCharacterData` class is a special abstract class that provides character-handling facilities for its subclasses. You will never create an instance of this class.

IDLffXMLDOM Object Helper Classes

IDL provides a set of other classes to assist you in navigating the DOM tree. These classes are:

- `IDLffXMLDOMNodeList` — contains a list of children of a node. You can create node lists using the `GetElementsByTagName` and `GetChildNodes` methods, for example.
- `IDLffXMLDOMNamedNodeMap` — contains a list of attributes from an element node that are looked up by attribute name.

These classes contain nodes that are subclasses of `IDLffXMLDOMNode`. Node lists and named node maps are active collections of nodes that are updated as the DOM tree is modified. That is, they are not static snapshots of a DOM tree in a given state; the list contents are modified as the DOM tree is modified. While this dynamic update is useful because you do not have to take specific action to update a list after modifying the tree, it can be confusing in some situations.

Suppose you want to delete all the children of an element node. The following code seems to make sense:

```
oList = oElement->GetChildNodes()
n = oList->GetLength()
FOR i=0, n-1 DO $
    oDeleted = oElement->RemoveChild(oList->Item(i))
```

This approach does not work as expected because after the first child is deleted, the list is updated so it contains one fewer object, and the indexes of all remaining objects are decremented by one. As the loop continues, some items are not deleted, and eventually an error occurs when the loop index i exceeds the length of the shortened list.

The following code performs the intended deletion, by changing the parameter to the `Item` method from i to 0:

```
oList = oElement->GetChildNodes()
n = oList->GetLength()
FOR i=0, n-1 DO $
    oDeleted = oElement->RemoveChild(oList->Item(0))
```

This code works because each time the first child is deleted, the list is automatically updated to place another object in the first position.

The following approach might be more appealing:

```
oList = oElement->GetChildNodes()
n = oList->GetLength()
FOR i=n-1, 0, -1 DO $
    oDeleted = oElement->RemoveChild(oList->Item(i))
```

This code works because it deletes items from the end of the list, rather than from the beginning.

IDL Node Ownership

Whenever you create an `IDLffXMLDOM` node object with a method such as `IDLffXMLDOMNode::GetFirstChild`, you are also creating an ownership relationship between the created node object and the node object that created it.

Working from the previous plug-in example (see “[About the DOM Structure](#)” on page 108), suppose that you have an object reference, `oName`, to an instance of the `IDLffXMLDOMELEMENT` class that refers to the first child of the plug-in node:

```
oName = oDocument->GetFirstChild()
```

Using `oName`, you can issue the following call:

```
oDescription = oName->GetNextSibling()
```

The description and name DOM nodes are siblings of each other in the DOM tree, as shown in [Figure 4-1](#). The IDL object `oDescription` refers to the description node in the DOM tree, and the IDL object `oName` refers to the name node in the DOM tree. However, the `oDescription` object is owned by the `oName` object because `oName` created `oDescription`.

You might understand this relationship better by realizing that the parent/sibling relationships in the DOM tree reflect the DOM tree structure and that the ownership relationships among the IDL access objects are due to the creation of the IDL access objects. Because `oName` created `oDescription`, `oName` destroys `oDescription` when `oName` is destroyed, even though they refer to siblings in the DOM tree. Bear in mind that destroying these access objects does not affect the DOM tree itself.

This parent relationship among `IDLffXMLDOM` objects is useful for cleaning them up. Because all of the objects that might have been created during the exploration of a DOM tree are all ultimately descendants of an `IDLffXMLDOMDocument` node, simply destroying the document object is sufficient to clean up all the nodes. Unless you are concerned with cleaning up some access objects at a particular time (to save memory, for example), you can simply wait to clean them all up when you are finished with the data by destroying the `IDLffXMLDOMDocument` node.

To reduce memory requirements, you can destroy node objects that are no longer needed. For example, if you wanted to explore all the children of a given element, you might use the following code:

```
oFirstChild = oElement->GetFirstChild()
oChild = oFirstChild
WHILE OBJ_VALID(oChild) DO BEGIN
    PRINT, oChild->GetNodeValue()
    oChild = oChild->GetNextSibling()
ENDWHILE
OBJ_DESTROY, oFirstChild
```

This approach works well because all the node objects created during the exploration of the children by the `GetNextSibling` method are destroyed when `oFirstChild` is destroyed. While it would seem that objects “lost” to the reassignment of `oChild` would not be accessible for destruction, the chain of `oChild` objects keeps track of them and destroys them all when the head of the chain, saved in `oFirstChild`, is destroyed.

Trying to destroy node objects inside the loop as follows does not work as expected:

```
oChild = oElement->GetFirstChild()
WHILE OBJ_VALID(oChild) DO BEGIN
    PRINT, oChild->GetNodeValue()
    oNext = oChild->GetNextSibling()
    OBJ_DESTROY, oChild
    oChild = oNext
ENDWHILE
```

This code fails because when `oChild` is destroyed for the first time, it also destroys `oNext`, causing the loop to exit after the first iteration.

If there is a very large number of children, waiting until the end of the loop to destroy the list might be too inefficient. Using a node list, as in the following code, is an alternative:

```
oList = oElement->GetChildNodes()
n = oList->GetLength()
FOR i=0, n-1 DO BEGIN
    oChild = oList->Item(i)
    PRINT, oChild->GetNodeValue()
    OBJ_DESTROY, oChild
ENDFOR
OBJ_DESTROY, oList
```

Although `oList` requires some space to maintain the list, there is only one valid node connected to `oChild` in memory each time through the loop.

Saving and Restoring IDLffXMLDOM Objects

IDL does not save IDLffXMLDOM objects in a SAVE file. If you restore a SAVE file that contains object references to IDLffXMLDOM objects, the object references are restored, but are set to null object references.

The IDLffXMLDOM objects are not saved because they contain state information for the external Xerces library. This state information is not available to IDL and cannot be restored. The contents of the XML file might also have changed, which would also make any saved state invalid.

It is recommended that applications either complete any DOM operations before saving their data in a SAVE file or reload the DOM document as part of restoring their state.

Using the XML DOM Object Classes

Continuing from the weather plug-in example (see [“About the DOM Structure”](#) on page 108), this section describes how to use the IDL XML DOM object classes, namely how to do the following actions:

- Load an XML document
- Read XML data from a document
- Modify existing XML data
- Create new XML data
- Destroy IDLffXMLDOM objects

Loading an XML Document

Although the DOM tree structure is in memory after the XML file is loaded, you cannot directly access the data from IDL until you have created IDLffXMLDOM objects to access them. The DOM loads and parses the XML data into a tree structure, but you need to create a document object to access that data through a mirroring IDL tree structure.

To prepare the interface, load the document:

```
oDocument = OBJ_NEW('IDLffXMLDOMDocument')
oDocument->Load, FILENAME='sample.xml'
```

This code causes the DOM tree structure to be formed in memory. You could also perform the same action in one line:

```
oDocument = OBJ_NEW('IDLffXMLDOMDocument', FILENAME='sample.xml')
```

Be aware that either of these examples will discard an existing DOM tree referenced by `oDocument`. You can load and reload an XML file as often as desired, but each loading action will overwrite, not add to, the existing tree and remove its objects from memory.

Reading XML Data

Suppose that you want to print the name of the plug-in. The plug-in element node is the first and only child of the document node. A document node can have only one element child node, which represents the containing element for the entire document (for comparison, consider that an HTML file has only one `<HTML></HTML>` pair). The name of the element node is the first element child of the plug-in element. There may be several ways to locate a desired piece of data using the IDL XML DOM classes. The following example illustrates one way to find the plug-in name.

First, access the first child of the document, which is the plug-in element:

```
oPlugin = oDocument->GetFirstChild()
```

The `GetFirstChild` method creates an `IDLffXMLDOMElement` node object and returns its object reference, which is stored in `oPlugin`.

Next, ask the plug-in for a list of all of its child element nodes. The `oPlugin` object creates an `IDLffXMLDOMNodeList` object and places all the child element nodes in the list. You could have asked for only the name element, but by asking for them all, you will have the other elements in the list in case you need to look at them later.

```
oNodeList = oPlugin->GetElementsByTagName('*')
```

You know from the design of the XML data, perhaps as defined in a DTD, that the name element must always be the first child of a plug-in element. You can access the name as follows:

```
oName = oNodeList->Item(0)
```

You also know that the name element can only contain a text node. Getting access to the text node lets you print the data that you want.

```
oNameText = oName->GetFirstChild()
PRINT, oNameText->GetNodeValue()
```

This command prints out:

```
Weather.com Radar Image [DEN]
```

Note that the `oPlugin` and the `oName` objects are of type `IDLffXMLDOMElement`, and the `oNameText` object is of type `IDLffXMLDOMText`. The `oName` and `oNameText` objects are created by the `GetFirstChild` and `Item` methods, using the object class that is appropriate for the type of data in the DOM tree. You used the `GetElementsByTagName` method to get the child elements of the plug-in, without having to sort through the whitespace text nodes that are present.

At this point, you have four IDL objects in addition to the root document object that give you access to only the portion of the DOM tree to which these objects correspond. You can create additional objects to explore other parts of the tree and destroy objects for parts that you are no longer interested in.

Modifying Existing Data

You can also modify XML data and write the result back out to a file.

```
oDocument = OBJ_NEW('IDLffXMLDOMDocument')
oDocument->Load, FILENAME='sample.xml'
oPlugin = oDocument->GetFirstChild()
oNodeList = oPlugin->GetElementsByTagName('*')
oName = oNodeList->Item(0)
oNameText = oName->GetFirstChild()
oNameText->SetNodeValue('Weather.com Radar Image [PDX]')
oDocument->Save, FILENAME='sample2.xml'
OBJ_DESTROY, oDocument
```

This code modifies the name node to change the airport to Portland, Oregon, and writes the modified XML to a new file. Please note that if you save to an existing file (e.g., using `sample.xml` instead of `sample2.xml` at the end of this example), the current XML data will replace the file entirely.

Creating New Data

You can create an [IDLffXMLDOMDocument](#) object and start adding nodes to it without loading a file.

```
oDocument = OBJ_NEW('IDLffXMLDOMDocument')
oElement = oDocument->CreateElement('myElement')
oVoid = oDocument->AppendChild(oElement)
oDocument->Save, FILENAME='new.xml'
OBJ_DESTROY, oDocument
```

This code creates the following XML file:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<myElement/>
```

Note that `<myElement/>` is XML shorthand for `<myElement></myElement>`.

Destroying IDLffXMLDOM Objects

Suppose that you are done with the name node and want to look at the description.

```
OBJ_DESTROY, oName
oDesc = oNodeList->Item(1)
oDescText = oDesc->GetFirstChild()
PRINT, oDescText->GetNodeValue()
```

This code destroys the `oName` object and `oNameText` with it because it was created by `oName`'s `GetFirstChild` method. This automatic destruction cleans up all the objects that you might have created from the `oName` node. You can then fetch the description element from the node list and print its name in the same manner. The name node is still in the node list and can be fetched again from the node list with the `Item` method, if needed.

Finally,

```
OBJ_DESTROY, oDocument
```

destroys the top-level object that you originally created with the `OBJ_NEW` function and also destroys any other objects that were created directly or indirectly from the `oDocument` object.

You can write the first code sample above more compactly because of the ability of the `IDLffXMLDOMDocument` object to clean up all the objects it and its children created:

```
oDocument = OBJ_NEW('IDLffXMLDOMDocument')
oDocument->Load, FILENAME='sample.xml'
PRINT, (((oDocument->GetFirstChild()->
  GetElementsByTagName('name'))->
  Item(0))->GetFirstChild()->GetNodeValue())
OBJ_DESTROY, oDocument
```

Under normal circumstances, the three object references created by the calls to the `GetFirstChild` and `GetElementsByTagName` methods would be lost because the object references to these three objects were not stored in IDL user variables. However, these objects are cleaned up by the document object when it is destroyed.

For additional information, see [“Orphan Nodes”](#) on page 122.

Please note:

- In general, you should not use the `OBJ_NEW` function to create any `IDLffXMLDOM` objects except for the top-level document object. Use the methods such as `GetFirstChild` to create the objects.

- You can destroy objects obtained from the various methods (e.g., `GetFirstChild`) at any time by the `OBJ_DESTROY` procedure.
- Objects destroyed by the `OBJ_DESTROY` procedure also destroy objects that they created.
- Destroying objects does *not* modify the DOM structure. That is, destroying any of the `IDLffXMLDOM` objects does not modify the data in the DOM tree. There are explicit methods for modifying DOM tree data. Destroying `IDLffXMLDOM` objects only removes your ability to access the DOM tree data.

Working with Whitespace

The XML parser is very particular about whitespace because all characters in an XML document define the content of that document. Whitespace consists of spaces, tabs, and newline characters, all of which are commonly used to format documents to make them easier to work with. In many cases, this whitespace is unimportant with respect to the document content. It is there only for presentation and does not affect the actual data stored in the XML document. However, in some cases, for example with `CDATA` or text node information, the whitespace might be important.

When whitespace is not important, IDL can treat it as ignorable. In many circumstances, you might want the parser to skip over this ignorable whitespace and not place it in the DOM tree so that you do not need to deal with it when visiting nodes in the DOM tree.

For example, the following two XML fragments produce different DOM trees when parsed with the default parser settings:

```
<stateList>
  <state>Colorado</state>
</stateList>

<stateList><state>Colorado</state></stateList>
```

In the first fragment, the `stateList` element has two child nodes that the second fragment does not. They are text nodes containing whitespace, a newline, and some tabs or spaces.

For the parser to distinguish between non-ignorable and ignorable whitespace, there must be a DTD or schema associated with the XML document, and it must be used to validate the document during parsing. This implies that a `VALIDATION_MODE` of 1 or 2 must be used when loading the XML document with the `IDLffXMLDOMDocument::Load` method.

Once validation is established, you can either:

- Tell the parser not to include ignorable text nodes in the DOM tree by setting the `EXCLUDE_IGNOREABLE_WHITESPACE` keyword in the `IDLffXMLDOMDocument::Load` method. If you select this option, the DOM trees for each of the above two fragments are the same.
- Check each text node in the DOM tree with the `IDLffXMLDOMText::IsIgnorableWhitespace` method.

Orphan Nodes

You can remove nodes from the DOM tree by using the `IDLffXMLDOMNode::RemoveChild` and `IDLffXMLDOMNode::ReplaceChild` methods. When these nodes are removed from the tree, they are owned by the DOM document directly and have no parent (since they are not in the tree anymore). Similarly, when these methods are used, the IDLffXMLDOM objects' ownership is changed as well because the IDL tree (made by creating the document interface and adding nodes) must mirror the underlying DOM tree.

If you issue the following command:

```
oMyRemovedChild = oMyElement->RemoveChild(oMyChild)
```

`oMyChild` is no longer owned by `oMyElement` and becomes owned by the document object to which all these nodes belong. Here, `oMyRemovedChild` and `oMyChild` are actually object references to the same object. The function method syntax provides a convenient way to create a new object reference variable with a new name that reflects the new status of the removed object, and you can use either name to access the orphaned node.

After removal, the orphan node is loosely associated with the document via the ownership relationship and would not be included in the output if the DOM tree were written to a file. You can insert the node back into the DOM tree with an `InsertBefore` or `AppendChild` method.

If the document that contains orphan nodes is destroyed, the orphan nodes are lost. More specifically, DOM tree orphan nodes are not written out to a file if they are orphans at the time that the `IDLffXMLDOMDocument::Save` method is used to save the tree, and the IDL node objects referring to the orphans are destroyed when the document object is destroyed.

Tree-Walking Example

The following code traverses a DOM tree using pre-order traversal.

```

PRO sample_recurse, oNode, indent

    ; "Visit" the node by printing its name and value
    PRINT, indent GT 0 ? STRJOIN(REPLICATE(' ', indent)) : '', $
        oNode->GetNodeName(), ':', oNode->GetNodeValue()

    ; Visit children
    oSibling = oNode->GetFirstChild()
    WHILE OBJ_VALID(oSibling) DO BEGIN
        SAMPLE_RECURSE, oSibling, indent+3
        oSibling = oSibling->GetNextSibling()
    ENDWHILE
END

PRO sample
    oDoc = OBJ_NEW('IDLffXMLDOMDocument')
    oDoc->Load, FILENAME="sample.xml"
    SAMPLE_RECURSE, oDoc, 0
    OBJ_DESTROY, oDoc
END

```

This program generates the following output for the plug-in file (see [“About the DOM Structure”](#) on page 108):

```

#document:
  plugin:
    #text:

    name:
      #text:Weather.com Radar Image [DEN]
    #text:

    description:
      #text:600 mile Doppler radar image for DEN
    #text:

    version:
      #text:1.0
    #text:

    tab:
      #text:

    icon:

```

```

        #text:weather.gif
#text:

    tooltip:
        #text:DEN Doppler radar image
#text:

#text:

```

The program above created an IDLffXMLDOM object for every node it encountered and did not destroy them until the document was destroyed. Another approach, illustrated in the program below, cleans up the nodes as it proceeds:

```

PRO sample_recurse2, oNode, indent
    ;; "Visit" the node by printing its name and value
    PRINT, indent gt 0 ? STRJOIN(REPLICATE(' ', indent)) : '', $
        oNode->GetNodeName(), ':', oNode->GetNodeValue()

    ;; Visit children
    oNodeList = oNode->GetChildNodes()
    n = oNodeList->GetLength()
    for i=0, n-1 do $
        SAMPLE_RECURSE2, oNodeList->Item(i), indent+3
    OBJ_DESTROY, oNodeList
END

PRO sample2
    oDoc = OBJ_NEW('IDLffXMLDOMDocument')
    oDoc->Load, FILENAME="sample.xml"
    SAMPLE_RECURSE2, oDoc, 0
    OBJ_DESTROY, oDoc
END

```

Please note that document and text nodes do not have node names, so the `GetNodeName` method always returns '#document' and '#text,' respectively.



Index

Symbols

- .sav file
 - description, [32](#)
 - query and restore, [31](#)

A

- accelerators
 - disabling, [47](#)
 - enabling, [47](#)
 - iTools, [20](#)
- ALPHA_CHANNEL property, [23](#)
- application user directory, [40](#)

B

- B format code, [33](#)
- backward compatibility, [82](#)

C

- CMYK color model, [24](#)
- creating XML data, [119](#)
- cropping
 - iImage, [13](#)
 - iTool manipulator, [15](#)
 - iTool operation, [14](#)

D

- dendrograms, [28](#)

destroying IDLffXMLDOM objects, 120
 DIALOG_PICKFILE routine, directory selection, 48
 DICOM, expanded support, 39
 DOM object classes, 111
 helper classes, 113
 Node, 111
 node ownership, 114
 saving and restoring, 116
 using, 117
 drag quality, 19

E

exporting
 Encapsulated Postscript, 11
 Enhanced Metafile, 11

F

format codes
 - Flag, 35
 + flag, 34
 B, 33
 zero padding, 35

H

help system
 Macintosh, 53
 UNIX, 53
 hierarchical cluster analysis, 28

I

IDL GUIBuilder, tabbing, 46
 IDL Virtual Machine
 application creation, 37
 widget blocking, 37

IDL_Savefile object, 31
 IDLffDicomEx object, 39
 IDLffXMLDOM
 destroying objects, 120
 orphan nodes, 122
 tree-walking example, 123
 IDLffXMLDOM object classes, 111
 helper classes, 113
 IDLffXMLDOMNode, 111
 node ownership, 114
 saving and restoring, 116
 using, 117

IDLgrText, vector graphics, 11
 iImage, enhancements, 13

ION

 availability, 80
 HTTPS support, 80

iTools

 command line, 55
 Data Manager, 17
 Data Manager enhancements, 15
 exporting
 Encapsulated Postscript, 11
 Enhanced Metafile, 11
 importing
 ESRI Shapefiles, 12
 JPEG 2000, 12
 macro mechanism, 12
 manipulator, 55
 style mechanism, 13
 ITRESOLVE procedure, 20

J

JPEG 2000
 iTools support, 12

L

- language catalog
 - definition, [98](#)
 - file, creating, [99](#)
 - widget example, [104](#)
- language catalog file
 - loading, [100](#)
 - storing, [100](#)
- language catalog object
 - adding keys, [101](#)
 - creating, [101](#)
 - destroying, [103](#)
 - languages
 - getting, [102](#)
 - setting, [102](#)
 - performing queries, [102](#)
- libraries
 - converting to prefixed, [94](#)
 - naming, [93](#)
- library authoring
 - benefits of, [90](#)
 - conversion wrappers, [94](#)
 - converting to prefixed, [94](#)
 - naming conventions, [91](#), [93](#)
 - prefixing routines, [91](#)
- library of routines
 - authoring, [89](#)
 - authoring conventions, [93](#)
 - converting existing, [94](#)
 - prefixing, [91](#)
- loading an XML document, [117](#)

M

- Macintosh, Alt key accelerators, [42](#)
- macros, iTools history, [12](#)
- mapping
 - Cartesian coordinates, [24](#)
 - iMap tool, [12](#)
- modifying XML data, [119](#)

- multi-threading, default number, [33](#)

N

- names, reserved, [93](#)
- namespace collisions, [91](#)
- naming conflicts, [91](#)

O

- object, transparency, [23](#)
- obsolete routines, [81](#)

P

- platforms
 - supported
 - IDL, [83](#)
 - ION, [85](#)
- prefixing libraries, [94](#)

R

- reading XML data, [118](#)
- rendering volume, iVolume, [20](#)
- requirements
 - IDL, [83](#)
 - ION, [85](#)
- reserved names, [93](#)
- resolving routine, [90](#)
- routines
 - conflicting names, [91](#)
 - naming, [93](#)
 - obsolete, [81](#)

S

- Shapefiles, iTools support, [12](#)
- styles, iTools, [13](#)

supported platforms

IDL, [83](#)

ION, [85](#)

T

tabbing

IDL GUIBuilder, [46](#)

widget navigation, [44](#)

transparency, [23](#)

U

UNIX, PDF help system, [53](#)

unsharp masking filter, [27](#)

V

variables, accessing non-local, [32](#)

vector graphics

inserting EMF file, [11](#)

text rendering, [11](#)

W

WIDGET_PROPERTY SHEET function

selection, [50](#)

sizing, [50](#)

widgets

accelerators, [47](#)

iTools compound, [55](#)

tabbing, [44](#)

using iTools, [55](#)

wrapper routines

compatibility wrappers, [95](#)

library conversion, [94](#)

X

XML

DOM

creating data, [119](#)

destroying objects, [120](#)

loading a document, [117](#)

modifying data, [119](#)

object classes, [111](#)

orphan nodes, [122](#)

reading data, [118](#)

tree-walking example, [123](#)

whitespace in, [121](#)

XML document

creating data, [119](#)

destroying objects, [120](#)

loading, [117](#)

modifying data, [119](#)

orphan nodes, [122](#)

reading data, [118](#)

whitespace in, [121](#)

XML DOM classes, [39](#)