Using IDL

RSI

IDL Version 6.2
July 2005 Edition

0705IDL62USG

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. RSI reserves the right to make changes to this document at any time and without notice.

Limitation of Warranty

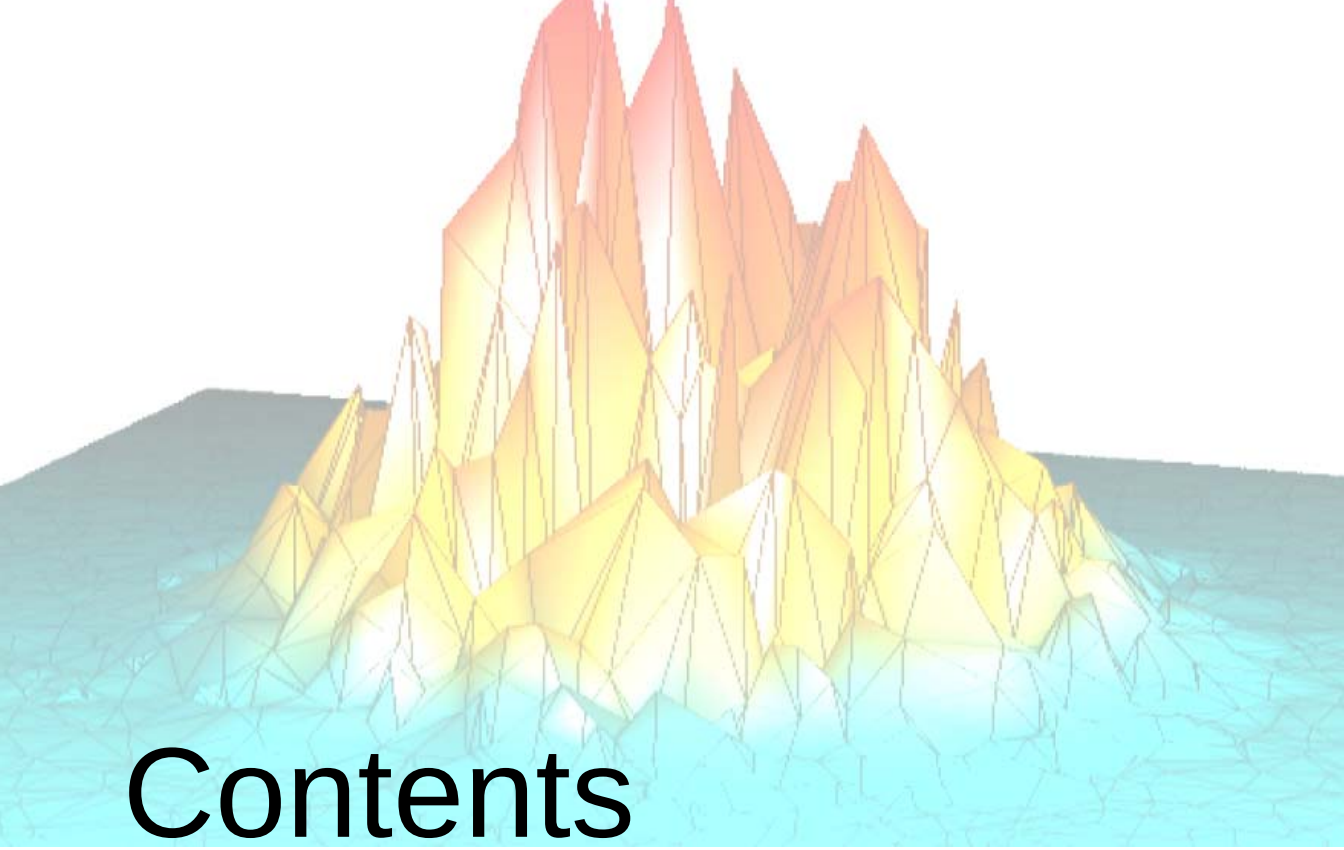
RSI makes no warranties, either express or implied, as to any matter not expressly set forth in the license agreement, including without limitation the condition of the software, merchantability, or fitness for any particular purpose.

RSI 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, RSI 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



Contents

Chapter 1
Introducing IDL 9

Overview of IDL 10
Supported File Formats 12
Launching IDL 15
Launching the iTools 17
Environment Variables Used by IDL 20
Command Line Options for IDL Startup 23
Startup Files 30
Message of the Day Files 31
Using Your Mouse with IDL 32
Using Keyboard Accelerators 33
Getting Help with IDL 35
Typographical Conventions 45
Quitting IDL 46

Reporting Problems 47

Chapter 2
The IDL Development Environment 51

Components of the IDLDE 52
File Menu 59
Edit Menu 63
Search Menu 65
Run Menu 67
Project Menu 73
Macros Menu 74
Window Menu 76
Help Menu 79
Printing in IDL 80
IDL Printer Setup in UNIX or Mac OS X 81

Chapter 3
Setting IDL Preferences 93

About IDL Preferences 94
Customizing IDL 95
General Preferences 97
Layout Preferences 100
Graphics Preferences 104
Editor Preferences 107
Startup Preferences 110
Font Preferences 112
Path Preferences 115

Chapter 4
Creating Development Environment Macros 119

What Are Macros? 120
Creating UNIX Macros 121
Creating Windows Macros 124
Command Stream Substitutions 126
Building IDL Example Macros 127

Chapter 5
Customizing IDL on Motif Systems 131
Using X Resources to Customize IDL 132
X Resources at the Command Line 136
Modifying the Control Panel 138
Action Routines 141

Chapter 6
Importing and Writing Data into Variables 149
Overview of Data Access in IDL 150
Accessing Files Using Dialogs 151
Reading ASCII Data 153
Reading Binary Data 154
Accessing Files Programmatically 156
Accessing Image Data Programmatically 158
Accessing Non-Image Data Programmatically 162
Using IDL Macros 164
File Access Routines 171

Chapter 7
Getting Information About Files and Data 173
Investigating Files and Data 174
Returning Image File Information 175
Returning Type and Size Information 179
Getting Information About SAVE Files 181
Returning Object Type and Validity 186
Returning Information About a File 188

Chapter 8
Graphic Display Essentials 189
IDL Visual Display Systems 190
IDL Coordinate Systems 193
Coordinates of 3-D Graphics 195
Coordinate Conversions 198
Interpolation Methods 201
Polygon Shading Method 203
Color Systems 204

Display Device Color Schemes 207
Colors and IDL Graphic Systems 209
Indexed and RGB Image Organization 213
Loading a Default Color Table 218
Using Fonts in Graphic Displays 221
Printing Graphics 222

Chapter 9
Map Projections 223

Overview of Mapping 224
Graphics Techniques for Mapping 225
Map Projection Types 227
Azimuthal Projections 228
Cylindrical Projections 237
Pseudocylindrical Projections 242
High-Resolution Continent Outlines 246
References 248

Chapter 10
Signal Processing 249

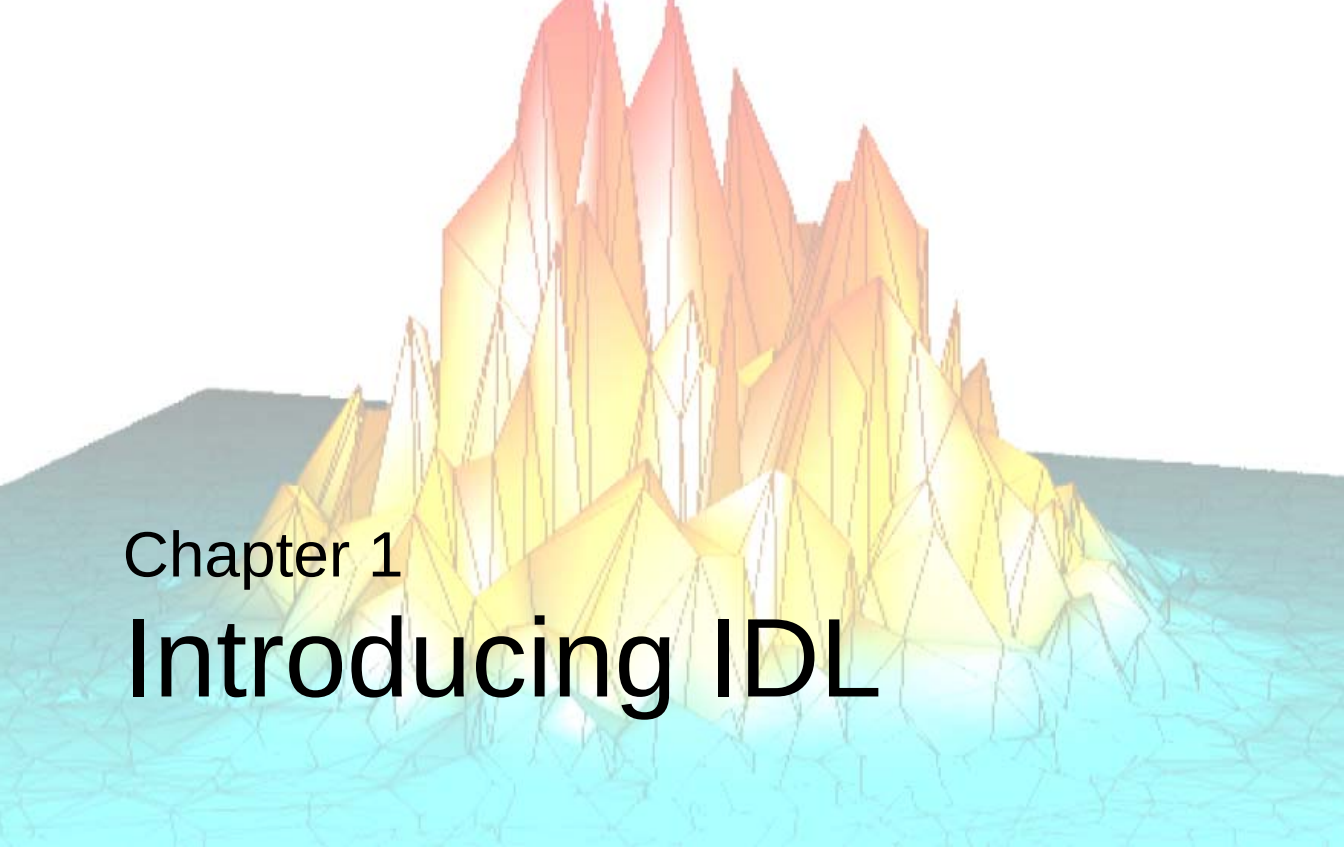
Overview of Signal Processing 250
Digital Signals 251
Signal Analysis Transforms 253
The Fourier Transform 254
Interpreting FFT Results 255
Displaying FFT Results 256
Using Windows 260
Aliasing 263
FFT Algorithm Details 264
The Hilbert Transform 265
The Wavelet Transform 267
Convolution 268
Correlation and Covariance 269
Digital Filtering 270
Finite Impulse Response (FIR) Filters 271
FIR Filter Implementation 273
Infinite Impulse Response Filters 275

References 278

Chapter 11
Mathematics 279

Overview of Mathematics in IDL 280
IDL's Numerical Recipes Functions 281
Correlation Analysis 282
Curve and Surface Fitting 286
Eigenvalues and Eigenvectors 288
Gridding and Interpolation 294
Hypothesis Testing 295
Integration 297
Linear Systems 302
Nonlinear Equations 309
Optimization 311
Sparse Arrays 313
Time-Series Analysis 316
Multivariate Analysis 319
References 325

Index 329



Chapter 1
Introducing IDL

This chapter includes information about IDL, the IDL documentation set, and how to contact RSI Technical Support. The following topics are covered in this chapter:

Overview of IDL 10
Supported File Formats 12
Launching IDL 15
Launching the iTools 17
Environment Variables Used by IDL 20
Command Line Options for IDL Startup . . . 23
Startup Files 30
Message of the Day Files 31
Using Your Mouse with IDL 32
Using Keyboard Accelerators 33
Getting Help with IDL 35
Typographical Conventions 45
Quitting IDL 46
Reporting Problems 47

Overview of IDL

IDL (the *Interactive Data Language*) is a complete computing environment for the interactive analysis and visualization of data. IDL integrates a powerful, array-oriented language with numerous mathematical analysis and graphical display techniques. Programming in IDL is a time-saving alternative to programming in FORTRAN or C. Using IDL, tasks which require days or weeks of programming with traditional languages can be accomplished in hours. You can explore data interactively using IDL commands and then create complete applications by writing IDL programs.

Analysis advantages include:

- Many numerical and statistical analysis routines—including Numerical Recipes routines—are provided for analysis and simulation of data. Compilation and execution of IDL commands provides instant feedback and hands-on interaction.
- Operators and functions work on entire arrays (without using loops), simplifying interactive analysis and reducing programming time.
- IDL's flexible input/output facilities allow you to read any type of custom data format. See "Supported File Formats" on page 12 for details.

Visualization advantages include:

- Rapid 2D plotting, multi-dimensional plotting, volume visualization, image display, and animation allow immediate observation of your computation's results.
- Support for OpenGL-based hardware accelerated graphics.

Application development advantages include:

- IDL is a complete, structured language that can be used interactively and to create sophisticated functions, procedures, and applications.
- IDL's *Intelligent Tools* (iTools) can be customized with your own operations or data manipulations.
- IDL widgets can be used to quickly create multi-platform graphical user interfaces to your IDL programs.
- Existing FORTRAN and C routines can be dynamically-linked into IDL to add specialized functionality. Alternatively, C and FORTRAN programs can call IDL routines as a subroutine library or display engine.

- IDL programs run across all supported platforms (UNIX, Macintosh and Microsoft Windows) with little or no modification. This application portability allows you to easily support a variety of computers.

Supported File Formats

IDL supports accessing the following types of file formats.

Image File Formats

For specific routine and object information used in IDL to access these type of files, see the "Image Data Formats" category under "Input/Output" in the *IDL Quick Reference* manual.

Format	Description
BMP	Windows Bitmap format
DICOM	Digital Imaging and Communications in Medicine
GIF	Graphics Interchange Format
Interfile	Interfile version 3.3 format
JPEG	Joint Photographic Experts Group format
JPEG 2000	JPEG 2000 format
MPEG	Moving Picture Experts Group format
MrSID	Multi-resolution Seamless Image Database format
NRIF	NCAR Raster Interchange Format
PICT	Macintosh version 2 PICT files (bitmap only)
PNG	Portable Network Graphics format
PPM	PPM/PGM format
SRF	Sun Raster File format
TIFF	8-bit or 24-bit Tagged Image File format
X11 Bitmap	X11 Bitmap format used for reading bitmaps for IDL widget button labels
XWD	X Windows Dump format

Table 1-1: IDL-Supported Graphics Standards

Scientific Data Formats

IDL supports the HDF (Hierarchical Data Format), HDF-EOS (Hierarchical Data Format-Earth Observing System), CDF (Common Data Format), and NetCDF (Network Common Data Format) self-describing, scientific data formats. Collections of built-in routines provide an interface between IDL and these formats. For specific routine and object information used in IDL to access these type of files, see the "Scientific Data Formats" category under "Input/Output" in the *IDL Quick Reference* manual.

Format	Description
CDF	Common Data Format version 2.7r1
HDF	Hierarchical Data Format version 4.1r5
HDF5	Hierarchical Data Format version 5-1.6.3
HDF-EOS	Hierarchical Data Format-Earth Observing System version 2.8
NCDF	Network Common Data Format version 3.5

Table 1-2: IDL-Supported Scientific Data Formats

Other Data Formats

For specific routine and object information used in IDL to access these data types, see the "Other Data Formats" category under "Input/Output" in the *IDL Quick Reference* manual.

Format	Description
ASCII	American Standard Code for Information Interchange
Binary	Digital data encoded as a sequence of bits
DXF	Drawing eXchange Format
ESRI Shapefile	Stores non-topological geometry and attribute information
SYLK	Symbolic Link Format

Table 1-3: Other IDL-Supported File Formats

Format	Description
VRML	Virtual Reality Modeling Language
WAV	Microsoft Waveform Format
WAVE	Wavefront Advanced Data Visualizer Format
XDR	eXternal Data Representation
XML	eXtensible Markup Language

Table 1-3: Other IDL-Supported File Formats (Continued)

Launching IDL

To launch the IDL program, do one of the following:

On Windows platforms — Launching IDL means starting the IDL Development Environment application (no command-line mode is available under Windows). The IDL Development Environment is described in detail in Chapter 2, "The IDL Development Environment". To start IDL, double-click on the IDL icon or select IDL from the **Start** menu.

On UNIX platforms — IDL offers two interfaces:

- In *command-line mode*, IDL uses a text-only interface and sends output to your terminal screen or shell window. (Graphics are displayed in IDL graphics windows.) To start IDL in command-line mode, enter `idl` at the shell prompt.
- In *graphical mode*, IDL displays the IDL Development Environment, an X-windows application that allows you to select options from menus, work with a built-in editor, and more. The IDL Development Environment is described in detail in Chapter 2, "The IDL Development Environment". To start IDL in graphical mode, enter `idlde` at the shell prompt.

On the Macintosh MacOS X platform — IDL is launched in the same way as on UNIX platforms, except that you must explicitly open an X11 Terminal window.

- In *command-line mode*, IDL uses a text-only interface and sends output to your terminal screen or shell window. (Graphics are displayed in IDL graphics windows.) To start IDL in command-line mode, enter `idl` at the X11 Terminal window shell prompt.
- In *graphical mode*, IDL displays the IDL Development Environment, an X-windows application that allows you to select options from menus, work with a built-in editor, and more. The IDL Development Environment is described in detail in Chapter 2, "The IDL Development Environment". To start IDL in graphical mode, double-click on the IDL icon or enter `idlde` at the X11 Terminal window shell prompt.

Startup Options

You can specify options to the command that starts IDL. On UNIX platforms, simply append the option flag after the `idl` or `idlde` command at the shell prompt. On Windows platforms, modify the **Target** field of the properties dialog for the IDL icon to include the option flag. See "Command Line Options for IDL Startup" on page 23 for a listing of the available startup options.

Troubleshooting

When IDL is ready to accept a command, it displays the IDL> prompt. If IDL does not start, take the following action depending upon the operating system you are running:

- Windows: Be sure that the path listed in the **Properties** dialog for the IDL icon accurately reflects the location of the IDL executable file `idlde.exe`.
- UNIX: Be sure that your `PATH` environment variable includes the directory that contains IDL.

Launching the iTools

The IDL Intelligent Tools (iTools) are a set of interactive utilities that combine data analysis and visualization with the task of producing presentation quality graphics. Based on the IDL Object Graphics system, 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 enjoying the convenience of a point-and-click environment. Each tool is designed around a specific visualization type:

- Two and three dimensional plots (line, scatter, polar, and histogram style)
- Surface representations
- Contour maps
- Image displays
- Volume visualizations
- Maps

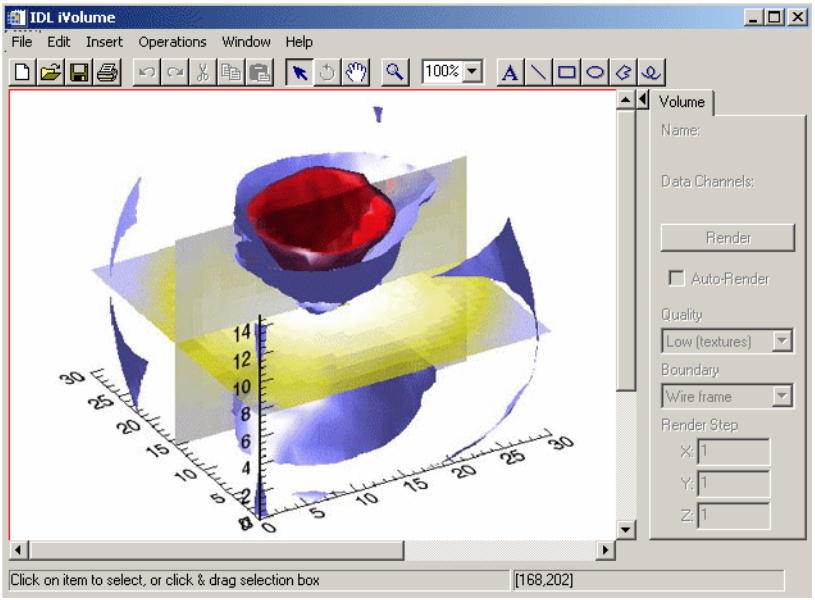


Figure 1-1: Black Hole Density Data in the iVolume Tool

For detailed information on the new iTools and how to use them, see the *iTool User's Guide*.

The iTools are built upon an object-oriented framework, or set of object classes, that serve as the building blocks for their interface and functionality. IDL programmers can easily use this framework to create custom data analysis and visualization environments. Such custom Intelligent Tools may be called from within a larger IDL application, or they may serve as the foundation for a complete application in themselves. For more information on creating your own custom iTools, see the *iTool Developer's Guide*.

Starting an iTool

To get started using the new IDL iTools, from the IDLDE command line, simply type the name of the tool you wish to call. The tools available are:

- **iContour**
- **iImage**
- **iPlot**
- **iSurface**
- **iVolume**
- **iMap**

You can also launch an iTool using these other methods:

- From Windows:

  Start → **Programs** → **RSI IDL 6.1** → **iTools** → *iTool Name*

- From the IDLDE:

  File → **New** → **Visualization** → *iTool Name*

Loading Data into an iTool

There are multiple options for loading your data into your chosen iTool for visualization:

- **Command Line Argument** — At the IDL Command Line enter:

  ```
  mydata = RANDOMU(SEED, 45)
  iPlot, mydata
  ```

  This option allows you to have control over parameters and keyword options for setting up the way you wish your plot (or other visualization) to appear.

- **File → Open** — The quickest way to create a default visualization of your data.
- **File → Import → IDL variable** — This will invoke the IDL Import wizard.
- **File → Import → From a File** -— This also invokes the IDL Import wizard.
- **Insert → Visualization** — This method gives you parameter control similar to using the command line.

Note

For more detailed information on loading data into the iTools, see Chapter 2, "Importing and Exporting Data" in the *iTool User's Guide* manual.

The iTools Data Manager

All data used by any iTool is first loaded into the iTools *Data Manager*, which keeps track of which data items are associated with an iTool visualization. The Data Manager provides a convenient and structured environment in which to import and view files and variables.

The process of loading data into the Data Manager is entirely automatic if you specify data when launching an iTool at the IDL command line or if you open a data file using the **Open** command from the iTool's **File** menu. In these cases, the iTool will import the data in the specified file or variable and create a visualization of the default type for the selected data and the iTool you are using.

If you want more control over the process of creating a visualization, you can load data into the Data Manager manually, either from a data file or from one or more variables that exist in your current IDL session. Once a data item is placed in the Data Manager, it is available to all iTools until it is removed.

Environment Variables Used by IDL

When IDL starts, it checks for the presence of a number of environment variables. If one of these environment variables exists, its value is used in one of two ways:

- As the value for a *preference*
- To configure IDL's environment in such a way that it can load and run

Preferences

Preferences are internal values that control various aspects of the environment IDL presents to its users. While user preference values are most often retrieved from preference files, the value of any preference can be defined by setting an environment variable of the same name to the appropriate value. For example, to set the value of the IDL_PATH preference, which supplies the initial value of the !PATH system variable, you would define an environment variable named IDL_PATH.

If an environment variable corresponding to a preference exists, its value will be used as the value of that preference unless the value is explicitly overridden with a value set at the command line when invoking IDL. See Appendix E, "IDL Preferences" in the *IDL Reference Guide* manual for a detailed description of IDL's preferences system and the precedence given to different sources for preference values.

Non-Preference Environment Variables

IDL checks the following environment variables at startup, but does not use the values as the values of IDL preferences.

CLASSPATH

The IDL-Java bridge uses the value of the CLASSPATH environment variable to locate user-defined Java classes.

DISPLAY

On UNIX platforms, IDL uses the DISPLAY environment variable to choose which X display is used to display graphics.

HOME

IDL uses the value of the HOME environment variable when storing user-specific information in the local file system.

Note

Under Microsoft Windows, the HOME environment variable might not be set in all cases. If it is not set, IDL first attempts to substitute the USERPROFILE environment variable (which usually looks something like `C:\Documents and Settings\`*`username`* where *username* is the login name of the current user). If USERPROFILE is not set, IDL uses the value of the first of the following it finds: the TEMP environment variable, the TMP environment variable, or the Windows system directory.

IDLJAVAB_CONFIG

The IDL-Java bridge uses the value of the IDLJAVAB_CONFIG environment variable to locate the IDL-Java bridge configuration file. See "Initializing the IDL-Java Bridge" in Chapter 8 of the *External Development Guide* manual for additional details.

IDLJAVAB_LIB_LOCATION

The IDL-Java bridge uses the value of the IDLJAVAB_LIB_LOCATION environment variable to determine which JVM shared library within a given Java version to use. See "Initializing the IDL-Java Bridge" in Chapter 8 of the *External Development Guide* manual for additional details.

LM_LICENSE_FILE

IDL's FLEXlm-based license manager uses the value of the LM_LICENSE_FILE environment variable to determine where to search for valid license files. Consult the license manager documentation for details.

PATH

When IDL asks for an operating system resource such as a shell, the executable file for that resource must be located in the operating system's path. While IDL itself does not use the value of the PATH environment variable explicitly, its value does affect IDL's behavior when attempting to launch other applications.

TERM

On UNIX platforms, IDL uses the environment variable TERM to determine the type of terminal in use when IDL is in command-line mode.

Setting Environment Variables

The process used to set environment variables varies depending on the operating system you are using.

UNIX and MacOS X Systems

On UNIX systems, environment variables are generally specified in a file read by your shell program at startup. Syntax for setting environment variables varies depending on the shell you are using, as does the file you use to specify the variables. If you are unsure how to set environment variables on your system, consult the system documentation or a system administrator.

For example, to set the environment variable IDL_PATH to the value `/usr/local/idl` when using a C shell (`csh`), you would add the following line to your `.cshrc` file:

```
setenv LM_LICENSE_FILE /usr/local/idl/license/license.dat
```

Similarly, to set the same variable when using a Bourne shell (`sh`), you would add the following lines to your `.profile` file:

```
LM_LICENSE_FILE="/usr/local/idl/license/license.dat" \
  ; export LM_LICENSE_FILE
```

Microsoft Windows Systems

On Microsoft Windows systems, environment variables are set in the Environment Variables dialog, which is accessible from the System Control panel. Some Windows versions allow you to set environment variables either only for the user you logged in as (“user variables”) or for all users (“system variables”). Setting IDL environment variables as user variables means that other users who log on to the computer will not have access to your environment variable values.

Command Line Options for IDL Startup

You can alter some IDL behaviors by supplying command-line switches along with the command used to invoke IDL. The following table shows the IDL command-line switches and the IDL interfaces to which they apply:

Switch	UNIX
	`idl`
	`idlde`
		Windows
		`idlde.exe`
			Windows
			`idlrt.exe`

-32	•		
-arg	•	•	•
-args	•	•	•
-autow	•		
-demo	•	•	•
-e	•	•	•
-em	•		•
-novm	•	•	•
-nw	•		
-pref	•	•	•
-queue	•	•	•
-quiet	•	•	•
-rt	•		•
-student	•	•	•
-ulicense	•	•	•
-vm	•		•
-w	•		

Table 1-4: Command Line Switches

Preference Switches

In addition to the switches listed above, you can specify the value of IDL preferences when invoking IDL. See "Specifying Preferences at the Command Line" on page 29 for details.

X Defaults

In addition to the switches listed above, there are numerous command-line switches that control the *appearance* of the IDL Development Environment on UNIX systems. Those options are not listed here, and future versions of the UNIX Development Environment might not continue to support them. See "X Resources at the Command Line" in Chapter 5 for details.

Batch Mode

IDL can also be started in non-interactive "batch" mode by specifying the name of a batch file at startup time. See Chapter 3, "Executing Batch Jobs in IDL" in the *Building IDL Applications* manual for details.

Command-Line Switches

The following command line switches can be used when invoking IDL. Unless otherwise noted, switches can be combined and can be specified in any order.

-32

Syntax: `-32`

Starts IDL in 32-bit mode. If this switch is not set, IDL starts in 64-bit mode by default for those platforms that support 64-bit. If you have not installed the 64-bit version, the 32-bit version will automatically be started. If you have not installed the 32-bit version, this flag will not work.

This switch is only available on UNIX platforms.

-arg

Syntax: `-arg` *value*

Specifies a single command line option to be saved for later access via the COMMAND_LINE_ARGS function. The *value* string is saved. Multiple `-arg` switches are allowed; the values are saved in the order specified. The `-arg` option can be used to pass program-specific information from the command line to IDL programs.

-args

Syntax: `-args` *value1 value2 ... valueN*

Specifies one or more command line options to be saved for later access via the COMMAND_LINE_ARGS function. When IDL sees the `-args` option, it takes any command-line arguments that follow it and saves them all. There can only be one `-args` option on an IDL command line, and it is always the final option. The `-args` switch can be used with the `-arg` switch; if both switches are specified, occurrences of `-arg` must come first, and the values specified by -args are saved following any values specified by `-arg`.

-autow

Syntax: `-autow`

Starts IDL with the graphical user interface if possible. If for any reason IDL cannot display the graphical user interface, it starts in command-line mode.

This switch is only available on UNIX platforms.

-demo

Syntax: `-demo`

Forces IDL to run in seven-minute demo mode.

-e

Syntax: `-e` *IDL_statement*

Specifies a single IDL statement to be executed. Once the statement has executed, IDL waits for any widget applications to exit, and then IDL itself exits. Only the last `-e` switch on the command line is honored.

Note

If the IDL statement includes spaces, it must be enclosed in quote marks. Under UNIX the statement can be enclosed in either single or double quotes, but under Microsoft Windows the statement must be enclosed in double quotes.

Under UNIX, the `-e` switch always uses the command line interface (that is, the `idlde` command followed by the `-e` switch behaves like the `idl` command followed by the `-e` switch).

Under Microsoft Windows, the `idlde` command displays the full development environment, but the user is not prompted for IDL commands to execute. This mode

is primarily useful because the output log window is visible, and will show any output generated by the IDL statement. The `idlrt` command also supports the `-e` option, and in this mode requires a standard IDL license. Since `idlrt` does not display the output generated by IDL statements, it is primarily of use for widget based applications that provide a graphical user interface to their functionality.

Note

Because the `-e` switch causes IDL to exit as soon as the statement is complete, if the IDL statement being executed produces graphics, you may wish to delay the exit until the user has a chance to view the graphics. In such a case, you must explicitly cause IDL to wait before exiting. For example, the following will produce a plot of one cycle of a sinusoid:

```
idlde -e "PLOT, SIN(FINDGEN(628)/100) & t=DIALOG_MESSAGE('Done')"
```

The plot will remain on the screen until the user dismisses the dialog, at which point IDL will exit.

-em

Syntax: `-em=`*file*

Starts IDL with an embedded license. The *file* argument should be an IDL `.sav` file that contains an embedded (“unlimited right to distribute”) IDL license. See Chapter 24, “Distributing Runtime Mode Applications” in the *Building IDL Applications* manual for details on creating applications with an embedded IDL license.

This switch is accepted on UNIX platforms and by the `idlrt.exe` application on Microsoft Windows platforms.

-novm

Syntax: `-novm`

Forces IDL to run in seven-minute demo mode rather than Virtual Machine mode if no license is available. This switch can only be used in conjunction with the `-rt` switch or the `idlrt.exe` executable.

If IDL attempts to load and run an IDL application in runtime mode, but finds no license available, it will load the application in Virtual Machine mode by default. Setting the `-novm` switch prevents the application from running in Virtual Machine mode, and instead causes it to run in demo mode.

-nw

Syntax: `-nw`

Starts IDL in command-line mode no matter what. Note that specifying `idlde -nw` at the shell prompt will start IDL in command-line mode.

This switch is only available on UNIX platforms.

-pref

Syntax: `-pref=`*file*

Loads the specified preference file. The *file* argument should be a text file containing IDL preference/value pairs. See Appendix E, "IDL Preferences" in the *IDL Reference Guide* manual for a detailed description of IDL's preferences system, the format of preference files, and the precedence given to different sources for preference values.

This feature is of particular interest to those writing stand-alone applications in IDL, possibly using the runtime or Virtual Machine modes of operation. The use of a command-line preference file allows authors of such applications to control the values of preferences important to their applications in a way that is user-adjustable and not hardwired into the code of their application.

-queue

Syntax: `-queue`

Causes IDL to wait for a license to become available before beginning an IDL task such as batch processing. This switch is useful for users of counted floating licenses who need their IDL process to run in licensed mode rather than in seven-minute demo mode.

-quiet

Syntax: `-quiet`

Suppresses printing of the IDL announcement and the `motd.txt` file. See "Message of the Day Files" on page 31 for details on message-of-the-day files.

-rt

Syntax: `-rt=`*file*

Starts IDL with a runtime license. If the *file* argument is specified, it should be an IDL `.sav` file. If the *file* argument is not specified, IDL attempts to run a file named

`runtime.sav`. See Chapter 24, "Distributing Runtime Mode Applications" in the *Building IDL Applications* manual for details on creating runtime applications.

This switch is accepted on UNIX platforms and by the `idlrt.exe` application on Microsoft Windows platforms. It is, however, redundant when using the `idlrt.exe` application.

-student

Syntax: `-student`

Forces IDL to start in student mode. This switch is useful for testing IDL applications that should run in student mode.

-ulicense

Syntax: `-ulicense`

Check out a unique license even if IDL is already running on the same display. If IDL has checked out a unique license using this flag, the user is allowed to change the DISPLAY environment variable after IDL has started.

-vm

Syntax: `-vm=`*file*

Starts the IDL Virtual Machine. If the *file* argument is specified, it should be an IDL `.sav` file. If the *file* argument is not specified, IDL displays a file selection dialog. See Chapter 25, "Distributing Virtual Machine Applications" in the *Building IDL Applications* manual for details on creating applications that run in the IDL Virtual Machine.

This switch is accepted on UNIX platforms and by the `idlrt.exe` application on Microsoft Windows platforms.

-w

Syntax: `-w`

Starts IDL with the graphical user interface. This is the same as entering `idlde` at the command prompt.

This switch is only available on UNIX platforms.

Specifying Preferences at the Command Line

In addition to the command line switches described above, the value of any IDL preference can be specified at the command line using the following syntax:

```
idlcommand -PREFERENCE value
```

where *idlcommand* is the command used to launch IDL (one of `idl`, `idlde`, or `idlrt`), *PREFERNCE* is the name of an IDL preference (note the leading hyphen), and *value* is the value for the preference. For example, to set the value of the IDL_MORE preference to one when launching IDL in command-line mode on a UNIX machine, you would use the following command line:

```
idl -IDL_MORE 1
```

Any number of preference values can be specified the command line. See Appendix E, "IDL Preferences" in the *IDL Reference Guide* manual for a detailed description of IDL's preferences system and the precedence given to different sources for preference values.

Using Switches Under Windows

Under Microsoft Windows, applications can be launched either from the prompt in a Command Window or by double-clicking on the application icon. If you launch IDL from a command prompt, simply specify the switch on after the name of the IDL executable you are using. For example, to start IDL in Virtual Machine mode using the `-vm` switch, you would use the following command line:

```
C:\RSI-Directory\bin\bin.x86\idlrt.exe -vm=file.sav
```

where *`RSI-Directory`* is the directory where you have installed IDL and *`file.sav`* is the name of the SAVE file you wish to restore and run.

If you launch IDL by double-clicking on the application icon, set switches by modifying the *target* specified in the application's shortcut properties to include the switch.

Startup Files

A *startup file* is a batch file that is executed automatically each time the IDL is started. The name of the startup file is specified by the IDL_STARTUP preference. (See Appendix E, “IDL Preferences” in the *IDL Reference Guide* manual for information on IDL’s preferences system.)

Common uses for startup files include the following:

- Restoring variable data contained in a `.sav` file or reading in commonly used data
- Setting common keywords to the DEVICE procedure
- Specifying shared or private color maps for PseudoColor devices

Startup files are executed one statement at a time. It is not possible to define program modules (procedures, functions, or main-level programs) in the startup file. For more information on creating batch files, see Chapter 3, “Executing Batch Jobs in IDL” in the *Building IDL Applications* manual.

Message of the Day Files

When IDL starts, it displays the contents of the `motd.txt` file, located in the `help/motd` subdirectory of the IDL distribution, in the Output Log. You can use this *Message of the Day* file to provide information to IDL users every time IDL starts.

In addition, IDL will display the contents a file with the name *`platform`*`.txt` located in the `help/motd` subdirectory of the IDL distribution, where *platform* is a string corresponding to the current operating system platform. For example, on Microsoft Windows systems, IDL displays a file named `win32.txt`.

You can determine the correct name for this file on a given platform by using the following IDL command:

```
PRINT, !VERSION.OS
```

and appending the "`.txt`" extension to the operating system name.

If you do not want to see either the `motd.txt` file or the platform-specific file each time IDL starts, remove them from the `help/motd` subdirectory of the IDL distribution.

Note

The `motd.txt` and platform-specific files are simply an ASCII text files—not IDL programs or batch files. To execute a series of IDL commands, select a startup file as described in "Startup Files" on page 30.

Using Your Mouse with IDL

IDL supports the use of mice with up to three buttons. Because some systems use mice with one or two buttons, IDL provides mechanisms for simulating a three-button mouse.

Using a Two-Button Mouse

Many mice used with Microsoft Windows systems have only two buttons. To simulate a middle-button press, hold down the CONTROL key and press the left mouse button.

Using a Macintosh (One-Button) Mouse

Many mice used with Macintosh systems have only one button. The X Window System software provided with MacOS X provides multi-button mouse emulation, allowing you to configure the system to generate middle- and right-button press events. See your MacOS X system documentation for details.

Using Keyboard Accelerators

IDL supports the use of keyboard *accelerators* or *shortcuts* in three different contexts: in the IDL Development Environment (menu actions), in the IDLDE Editor window, and in IDL widget applications. For information on development environment keyboard shortcuts, see one of the following:

- Chapter 2, "The IDL Development Environment" provides descriptions of each available menu item including keyboard shortcuts
- "Editor Window Keyboard Shortcuts" in Chapter 2 of the *Building IDL Applications* manual describes keyboard shortcuts specifically designed for use in the Editor window

Keyboard shortcuts can also be defined for individual buttons and menu items in an IDL widget application. Defining shortcut key combinations is the responsibility of the IDL programmer who creates the widget application; if you are using a widget application and are unsure about whether keyboard shortcuts have been defined, contact the author of the widget application. For information on adding keyboard accelerators to your own widget applications, see "Enhancing Widget Application Usability" in Chapter 30 of the *Building IDL Applications* manual.

Enabling Alt Key Accelerators on Macintosh

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.

Getting Help with IDL

IDL's online help system provides access to information on all aspects of IDL. The complete IDL documentation set is available online in HTML format. To use the IDL online help system, do one of the following:

- Enter the `?` command (optionally followed by a routine or object name) at the IDL command prompt
- Call the ONLINE_HELP procedure at the IDL command prompt or within an IDL program
- If you are running the IDL Development Environment (IDLDE), select the **Help** option from the menu bar
- Select **IDL Help** from the Microsoft Windows Start menu
- Double-click on the **IDLHelp** Macintosh icon

In addition to the online help format, IDL documentation is available in a set of Adobe Acrobat PDF files located on the IDL CD-ROM. See "Using the PDF Documentation Set" on page 43 for details.

Using the IDL Online Help Viewer

IDL's online help system uses a cross-platform help viewer — *IDL Assistant* — based on the help viewer used by the Qt development toolkit from Trolltech. This section describes how to use the IDL Assistant application. For information on creating help content that uses the IDL Assistant for your own IDL applications, see Chapter 23, "Providing Online Help For Your Application" in the *Building IDL Applications* manual.

The Main Window

The IDL Assistant *main window* contains the text of the current topic. Within the main window you can:

- Follow hypertext links to other topics, or to sections within the current topic
- Navigate to the next or preceding topic using arrows at the top of the topic screen
- Display multiple topics simultaneously using the tabbed interface
- Create new tabs and close existing tabs using icons to the right and left of the tabs

- Perform common tasks including display of the next/previous topic, tab management, text sizing, copying text to the clipboard, and finding text within the topic using the context menu

The Sidebar

The IDL Assistant *sidebar* provides four tabs that allow you to navigate through the IDL documentation set. All of the tabs provide a context menu that allows you to open the selected topic the current tab, a new tab, or a new window.

The Contents Tab

The **Contents** tab displays a hierarchical listing of the contents of the various books in the IDL documentation set.

The Index Tab

The **Index** tab provides a keyword index of the contents of the IDL documentation set. Enter a text string in the **Look For:** field to see keywords that match the string.

The Search Tab

The **Search** tab allows you to search the text of the IDL documentation set for words or phrases. Text matching your search string is highlighted when a topic is displayed in the main window.

Tip

Words or phrases entered in the **Search** tab are *not* case sensitive.

To search for words, enter one or more strings in the **Searching for:** field, separated by spaces and click **Search**. IDL Assistant displays a list of topics that contain all of the words you entered.

To search for a phrase, enclose the phrase in single or double quote marks.

Warning

The IDL documentation set is quite large. The results of a full-text search query may take several moments to appear in the **Search** tab.

The list of topics containing the search words or phrase is displayed as a list ranked roughly according to the number of occurrences of the words or phrases, with the topics containing the largest number of occurrences listed given higher rankings.

Each entry in the list of topics is followed by an abbreviation of the title of the manual in which the topic appears. See “Book Name Abbreviations” on page 38 for the list of abbreviations.

Allowed Characters

The following characters are allowed in the **Search** tab:

- Letters (upper- and lower-case)
- Numbers (`0-9`)
- Quote marks (single (`'`), double (`"`), backwards (`` ` ``))
- Exclamation marks (`!`), colons (`:`), and periods (`.`)
- Spaces
- Hyphens (`-`)
- Underscores (`_`)
- Asterisk (`*`) as a wildcard matching one or more unspecified characters

Note

The `*` character cannot be used within quotes or at the beginning of a string.

All other characters are disallowed; you cannot enter them in the **Searching for:** field.

Warning

Searches that contain single-character strings (such as “a” or “8”) are not allowed and will return no results. This is true even when the single character is combined with a punctuation character such as a hyphen. For example, searching for the string “8-bit” will return no results.

Examples

`convol`	List all topics that contain the word “convol”
`convol*`	List all topics that contain a word *beginning* with “convol”
`base widget`	List all topics that contain the word “base” *and* the word “widget”
`"base widget"`	List all topics that contain the phrase “base widget”

Book Name Abbreviations

The following abbreviations of book titles are used in the list of topics returned by the search:

`bld`	Building IDL Applications
`dm`	DataMiner Guide
`edg`	External Development Guide
`gs`	Getting Started with IDL
`img`	Image Processing Guide
`inst`	Installing and Licensing IDL
`ionj`	ION Java User's Guide
`ions`	ION Script User's Guide
`itd`	iTool Developer's Guide
`itu`	iTool User's Guide
`med`	Medical Imaging in IDL
`obj`	Object Programming
`obs`	Obsolete Features
`ref`	IDL Reference Guide
`sdf`	Scientific Data Formats
`use`	Using IDL
`wav`	Wavelet Toolkit User's Guide
`wn`	What's New in IDL

The Bookmarks Tab

The **Bookmarks** tab allows you to save links to specific topics in the IDL documentation set for easy reference.

The Menu Bar

The IDL Assistant *menu bar* runs across the top of the IDL Assistant window, and provides access to the features listed below. Keyboard shortcuts to invoke various menu items are listed in the menus themselves.

Menu	Item	Function
File	**New Window**	Open a new IDL Assistant window.
	Add Tab	Open a new tab displaying the same topic as the currently selected tab.
	Close Tab	Close the currently selected tab.
	Print	Print the contents of the currently selected tab. See "Printing" on page 42 for details.
	Close	Close the current IDL Assistant window.
	Exit	Close all IDL Assistant windows.
Edit	**Copy**	Copy text selected in the main window to the system clipboard.
	Find in Text...	Search for a text string in the currently displayed topic.
	Find Next	Find the next instance of the text string in the currently displayed topic.
	Find Previous	Find the previous instance of the text string in the currently displayed topic.
	Settings...	Display the Settings dialog. See "Settings" on page 42 for details.

Table 1-5: IDL Assistant Menus

Menu	Item	Function
View	**Zoom in**	Increase the text size in the main window. See "Text Zoom" on page 41 for important notes.
	Zoom out	Decrease the text size in the main window. See "Text Zoom" on page 41 for important notes.
	Views...	Control display of the Sidebar and Standard toolbar. **Note -** The Line Up feature realigns the toolbar if it has been moved.
Go	**Previous**	Display the current tab's previous topic.
	Next	Display the current tab's next topic.
	Home	Display the IDL online help Home page.
	Next Tab	Select the tab to the right of the current tab, if any.
	Previous Tab	Select the tab to the left of the current tab, if any.
Bookmark	**Add Bookmark**	Create a bookmark for the currently selected topic.
	Bookmark list	Existing bookmarks are displayed at the bottom of this menu.
Help	**IDL Assistant Manual**	Display this help topic.
	About IDL Assistant	Display information about IDL Assistant.
	What's This?	Display context-sensitive pop-up help about some portions of the IDL Assistant interface.

Table 1-5: IDL Assistant Menus

The Tool Bar

The IDL Assistant *tool bar* provides quick access to a subset of the features available via the menubar.

Icon	Name	Function
	Previous	Display the current tab's previous topic.
	Next	Display the current tab's next topic.
	Home	Display the IDL online help Home page.
	Copy	Copy text selected in the main window to the system clipboard.
	Find in Text	Search for a text string in the currently displayed topic.
	Print	Print the contents of the currently selected tab. See "Printing" on page 42 for details.
	Zoom in	Increase the text size in the main window. See "Text Zoom" on page 41 for important notes.
	Zoom out	Decrease the text size in the main window. See "Text Zoom" on page 41 for important notes.
	What's this?	Display context-sensitive pop-up help about some portions of the IDL Assistant interface.

Table 1-6: IDL Assistant Toolbar

Text Zoom

Select **Zoom in** or **Zoom out** from the **View** menu to change the size of the text in the IDL Assistant main window.

The smoothness of the text zoom operation depends on the ability of the operating system to provide fonts of the appropriate size for the zoomed text. On platforms that provide robust font-management mechanisms, the **Zoom** operations will work smoothly. On platforms that provide more limited font support, a single **Zoom** operation may, depending on the current text size and font support, change the text size for only some text elements in the main window, or none at all. In these cases, repeated applications of the **Zoom** operations may change the text size.

If you find that the text zooming feature does not work adequately with the default fonts, try changing the fonts used by IDL Assistant (see "Settings" on page 42 for details.) On platforms that use a set of fixed-size fonts, choosing a font with a larger number of available sizes will allow smoother text zooming.

Printing

Select **Print** from the **File** menu or toolbar to display a platform-native **Print** dialog that allows you to select a printer on which to print.

Note

Currently, the only text range option available is **All**. Printing all will print the entire contents of the topic currently displayed in the main window.

Tip

The quality of the printed output from IDL Assistant depends on the platform and printer in use. For high-quality printed output, consider printing from the PDF version of the document you are viewing. See "Using the PDF Documentation Set" on page 43 for details.

Settings

Select **Settings** from the **Edit** menu to display a tabbed dialog that allows you to control several IDL Assistant settings.

General Tab

The **General** tab allows you to select fonts for text display in the main window. By default, the **Font** is set to Helvetica, and the **Fixed Font** is set to Courier.

Tip

Depending on the configuration of your system, you may be able to select alternate fonts that provide better appearance or smoother zooming behavior than the defaults. This is especially true on UNIX systems that have a limited set of fonts available. Trying different font settings may improve both the legibility of the text and the ability to zoom in the IDL Assistant viewer.

The **General** tab also allows you to select a color for hyperlinks and specify whether the links should be underlined. Depending on your platform, changing these values may not produce the effect you expect.

Web Tab

The **Web** tab allows you to define the web browser that should be invoked when you click on a hyperlink that refers to a web site rather than to a file in the IDL documentation set.

The **Web** tab also allows you to specify an HTML file that should be displayed when you select **Home** from the **Go** menu or click the **Home** toolbar icon. By default, the home page is defined as

```
<IDL_DIR>/help/online_help/home.html
```

where <IDL_DIR> is the full path to your IDL installation.

PDF Tab

The **PDF** tab allows you to define a Portable Document Format (Adobe Acrobat) file browser that should be invoked when you click on a hyperlink that refers to a PDF file.

Using the PDF Documentation Set

The complete IDL documentation set is available in a set of Adobe Portable Document Format (PDF) files. The PDF documentation set is hyperlinked, provides navigational bookmarks in the bookmarks pane, and provides a compiled full-text search index.

Adobe Systems Inc. created the Portable Document Format in the early 1990s, basing it on their PostScript language. PDF is intended to allow documents to be displayed in exactly the same manner on a wide variety of computing platforms.

The IDL PDF files are electronic representations of the individual books in the documentation set, and can be either viewed on screen or printed (in full or in part) on a local printer. When viewed on-screen, the PDF books provide hyperlinked cross-references, tables of contents, and indices, allowing for speedy navigation through the set. In addition, some versions of the Adobe Acrobat software provide a fast full-text search capability, using a pre-compiled full-text index of the entire document set.

Viewing PDF Files

Viewing PDF files requires a separate application, not included in the IDL installation. Various PDF viewing applications are in wide use, and one or more may already be installed on your system.

The PDF version of the IDL documentation set is designed to be viewed using Adobe Acrobat or Adobe Reader. Other third-party PDF viewers (notably GhostScript and Apple’s Preview) are available, but these viewers may not support all of the features

available when viewing the IDL PDF files in Adobe Acrobat. Inter-document hyperlinks may or may not work correctly when using other viewers, and the compiled full-text search index (the Acrobat “Search” feature) will almost surely not work correctly in other viewers.

The Adobe Reader software is available at no charge directly from Adobe:

```
www.adobe.com/reader
```

Locating the PDF Documentation Set

The PDF version of the documentation set is not installed with IDL. The PDF files are located in the `info/docs` subdirectory of the IDL installation CD-ROM.

Typographical Conventions

The following typographical conventions are used throughout the IDL documentation set:

- UPPER CASE type
  IDL functions and procedures, and their keywords are displayed in UPPER CASE type. For example, the calling sequence for an IDL procedure looks like this:

  ```
  CONTOUR, Z [, X, Y]
  ```

- Mixed Case type
  IDL object class and method names are displayed in Mixed Case type. For example, the calling sequence to create an object and call a method looks like this:

  ```
  object = OBJ_NEW('IDLgrPlot')
  object -> GetProperty, ALL=properties
  ```

- *Italic type*
  Arguments to IDL procedures and functions — data or variables you must provide — are displayed in italic type. In the above example, *Z*, *X*, and *Y* are all arguments.

- Square brackets ([ ])
  Square brackets used in calling sequences indicate that the enclosed arguments are optional. Do not type the brackets. In the above CONTOUR example, *X* and *Y* are optional arguments. Square brackets are also used to specify array elements.

- `Courier type`
  In examples or program listings, things that you must enter at the command line or in a file are displayed in `courier` type. Results or data that IDL displays on your computer screen are also shown in courier type. An example might direct you to enter the following at the IDL command prompt:

  ```
  array = INDGEN(5)
  PRINT, array
  ```

  In this case, the results are shown like this:

  ```
     0       1       2       3       4
  ```

Quitting IDL

To quit IDL, do one of the following:

- Enter the `EXIT` command at the IDL command prompt.
- If you are running the IDL Development Environment (IDLDE), select the **Exit** option from the **File** menu.
- Under Microsoft Windows, press **Alt+F4**.
- Under UNIX or MacOS X, if you use IDL's command-line mode, press **Ctrl+D** as the first character in command-line mode causes IDL to exit back to the operating system. The `EXIT` procedure has the same function. If **Ctrl+D** is not the first character, it simply ends the input line as if a return had been entered.

Note

When using IDL's command-line mode under UNIX or MacOS X, you can normally press **Ctrl+Z** to suspend IDL and return you to the shell process without exiting IDL. After completing any shell commands, type `fg` to return IDL to the foreground. Although the UNIX suspend character can be changed by the user outside of IDL, this is rarely done. For the purposes of this manual, we assume the default convention.

Reporting Problems

We strive to make IDL as reliable and bug free as possible. However, no program with the size and complexity of IDL is perfect, and problems do surface. When you encounter a problem with IDL, the manner in which you report it has a large bearing on how well and quickly we can fix it.

The relnotes.txt file accompanying each release includes information about new features in that release, bug fixes, and known problems which may be of help.

This section is intended to help you report problems in a way which helps us to address the problem rapidly.

Background Information

Sometimes, a problem only occurs when running on a certain machine, operating system, or graphics device. For these reasons, we need to know the following facts when you report a problem:

- Your IDL installation number.
- The version of IDL you are running.
- The type of machine on which it is running.
- The operating system version it is running under.
- The type and version of your windowing system if you are on UNIX.
- The graphics device, if the problem involves graphics and you know what graphics device is on your system.

The installation number is assigned by us when you purchase IDL and is included in the license information that we sent you. The IDL version, site number, and type of machine are printed when IDL is started.

For example, the following startup announcement appears indicating you are running IDL version 6.2 under Sun Solaris using installation number xxxxx-x, under a floating license located on a particular license manager.

```
IDL Version 6.2, Solaris (sunos sparc m64).
(c) 2004, Research Systems, Inc.
Installation number: xxxxx-x.
Licensed for use by: RSI IDL floating licenses
```

Under UNIX, the version of the operating system can usually be found in the file `/etc/motd`. It is also printed when the machine boots. In any event, your system administrator should know this information.

Under Windows, select **About** from the **Help** menu in the Windows Explorer.

Double Check

Before reporting a problem double check with the manual or a local expert if one is available. Sometimes, it is a simple matter of misinterpreting what is supposed to happen.

If you cannot determine what should happen in a given situation by consulting the reference manual, the manual needs to be improved on that topic. Please let us know if you feel that the manual was vague or unclear on a subject.

Another question to ask is whether the problem lies within IDL, or with the system running IDL. Is your system properly configured with enough virtual memory and sufficient operating system quotas? Does the system seem stable and is everything else working normally?

Describing The Problem

When describing the problem, it is important to use precise language. Terms like crashes, blows up, and fails are vague and open to interpretation. Does it really crash IDL and leave you looking at an operating system prompt? This is how RSI technical support personnel interpret a problem report of a crash. If the behavior being reported refers to an unexpected error message being issued before returning another prompt, then describing it as a crash becomes misleading. What is really meant by a term like "fails?"

It is also important to separate concrete facts from conjecture about underlying causes. For example, a statement such as "IDL dumps core when allocating dynamic memory" is not nearly as useful as this statement, "IDL dumps core when I execute the following statements... "

Reproducibility

Intermittent problems are by far the hardest kind to fix. In general, if we can't make it happen on our machine, we can't fix it. It is far more likely that we can help you if you can tell us a sequence of IDL statements that cause the problem to happen. Naturally, there are degrees of reproducibility. Situations where a certain sequence of statements causes the problem 1 time in 3 tries are fairly likely to be fixable. Situations where the problem happens once every few months and no one is sure what triggered it are nearly impossible to identify and correct.

Simplify the Problem

In accordance with RSI Technical Support policy, when reporting a problem, it is important to give us the shortest possible series of IDL statements that cause it. Here are some suggestions for simplifying your problem:

Copy the procedure and function files that are involved to a scratch second copy. Never modify your only copy!

Eliminate everything not involved in demonstrating the problem. Don't do this all at once. Instead, do it in a series of slow careful steps. Between each step, stop and run IDL on the result to ensure that the problem still appears.

If a simplification causes the problem to disappear, then slowly restore the statements involved until you can identify the source of the problem. The end result of such simplification should be a small number of IDL statements that demonstrate the problem.

If the problem does not involve file Input/Output, strive to eliminate all file I/O statements. Use IDL routines to generate a dummy data set, rather than including your own data if at all possible. If your problem report does not involve your data, it will be much easier for us to reproduce.

On the other hand, if the problem involves file Input/Output, and the problem only happens with a certain data file or type of data, we will need to look at your data or a sample of your data.

If it is necessary to send us your data, use one of the following methods:

- If the data set is small, please send it as an attachment in your email to us: support@RSInc.com.
- If the data set is large, please place it on our ftp site at: ftp.RSInc.com/incoming.

Be sure to include the commands that reproduce your problem in your message to use. If you have placed your data on the ftp site, include the name of the data set and when it was uploaded.

Problems with Dynamic Loading

Under some operating systems, the CALL_EXTERNAL and LINKIMAGE system routines allow you to dynamically load routines written in other languages into IDL. This is a very powerful technique for extending IDL, but it is considerably more difficult than simply writing IDL statements. At this level, the programmer is outside the user level shell of IDL and is not protected from programming errors. These errors could give incorrect results or crash IDL. In such situations, the burden of

proving that a problem is within IDL and not the dynamically loaded code is entirely the programmer's.

Although it is certainly true that a problem in this situation can be within IDL, it is very important that you exhaust all other possibilities before reporting the problem. If you decide that you need to report the problem, the comments above on simplifying things are even more important than usual. If you send us a small example that exhibits the problem, we may be able to respond with a correction or advice.

Contact Us

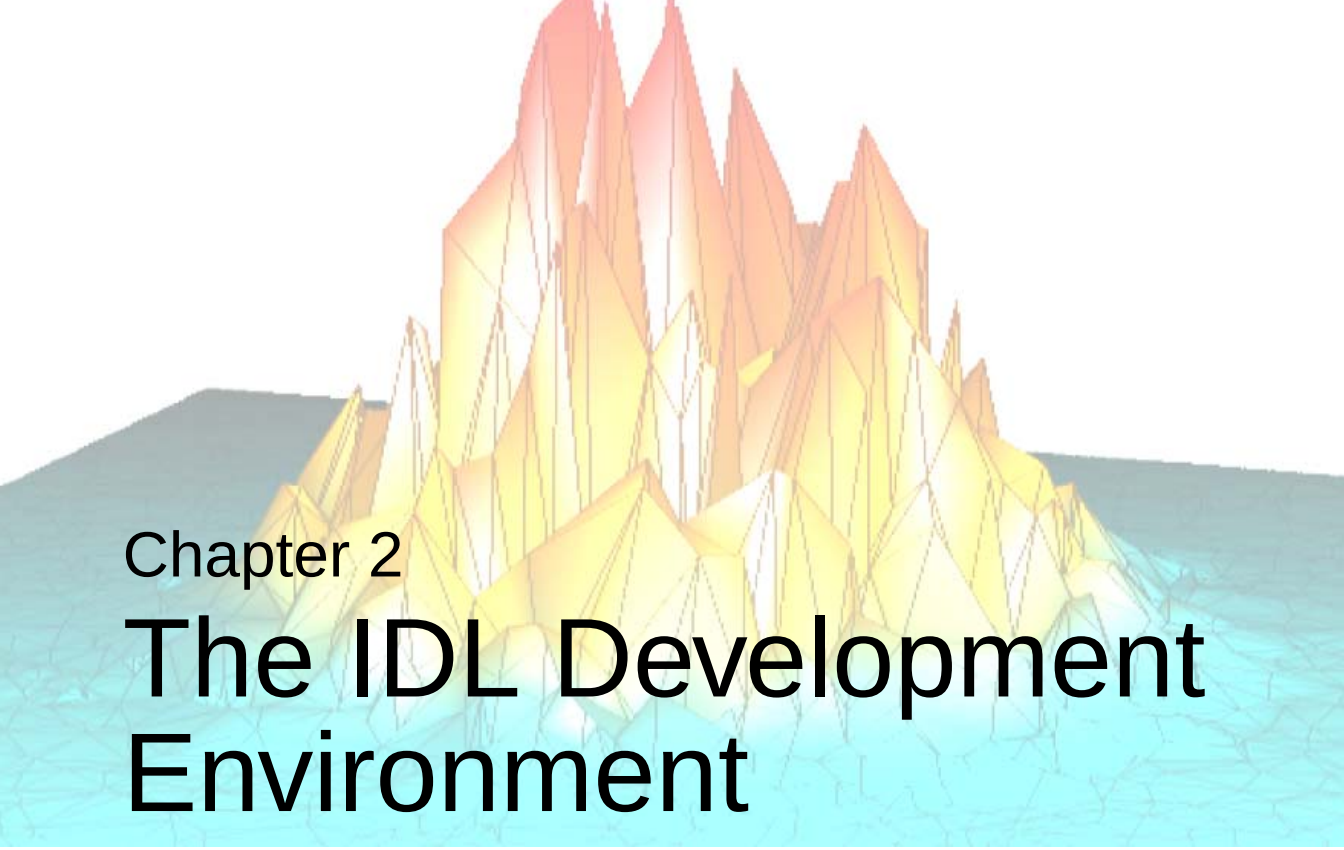
To report a problem, contact us at the following addresses:

Electronic Mail
support@RSInc.com

Telephone
(303) 786-9900
(303) 786-9909 (Fax)
(303) 413-3920 (IDL technical support direct line)

Mail
Research Systems, Inc.
4990 Pearl East Circle
Boulder, CO 80301

Web Site
http://www.RSInc.com



Chapter 2

The IDL Development Environment

This chapter describes the IDL Development Environment.

Components of the IDLDE 52
File Menu 59
Edit Menu 63
Search Menu 65
Run Menu 67
Project Menu 73
Macros Menu 74
Window Menu 76
Help Menu 79
Printing in IDL 80
IDL Printer Setup in UNIX or Mac OS X . 81

Components of the IDLDE

The IDL Development Environment (IDLDE) is a convenient multiple-document graphical user interface that includes built-in editing and debugging tools. This section describes briefly the components of the IDLDE. The Windows version is shown on the left and the UNIX version is shown on the right within the following figure.

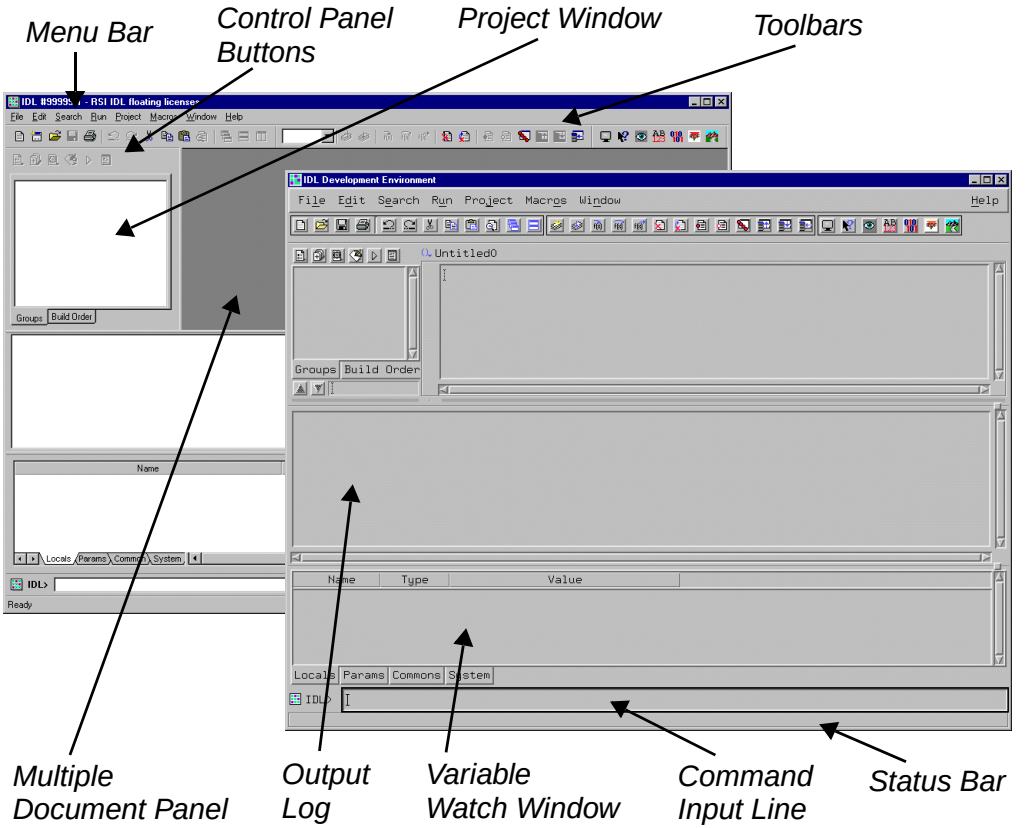


Figure 2-1: The IDL Development Environment for Windows (left) and UNIX (right).

Note

Individual components are similar across platforms.

Menu Bar

The menu bar, located at the top of the main IDLDE window, allows you to control various IDLDE features. When you select an option from a menu item in the IDLDE, the Status Bar displays a brief description.

You can display menu commands for each menu using the following methods:

- Clicking the menu on the Menu bar.
- Pressing the **Alt** key plus the underlined letter in the menu's title. For example, to display the File menu, press **Alt+F**.

You can select or execute a menu command using the following methods:

- Clicking the item in the menu.
- Pressing the **Alt** key plus the underlined letter in the menu's title, and then pressing the letter underlined in the menu item. For example, to select the menu item **File → Open**, press **Alt**+**F**+**O**.
- Using the cursor and the arrow keys to highlight a menu item, and then pressing the **Enter** key.

Note

Many items (on each platform) have keyboard shortcuts displayed to the right of the corresponding menu option.

The menu bar consists of the following menu items:

Menu Item	Description of Functions
File Menu	The File Menu gives you options such as opening, closing and creating new Editor windows and Projects and other options such as printing, printer setup, preferences and exiting IDL.
Edit Menu	The Edit Menu provides edit-related options such as undo, redo, cut, copy, paste, delete, select all, clear all and clear log.
Search Menu	The Search Menu allows you to find text in currently active Editor windows as well as other options such as find again, find selection, enter selection, replace, replace & find, go to line and go to definition.

Table 2-1: The IDLDE Menus

Menu Item	Description of Functions
Run Menu	Run Menu items are enabled when an IDL program is loaded into an IDL Editor window. The run menu allows you program related functionality such as compiling, resolving dependencies, resetting, and editing programs among other things. For more information on running programs in IDL, see Chapter 2, "Creating and Running Programs in IDL" in the *Building IDL Applications* manual.
Project Menu	The Project Menu provides project-related functionality such as adding/removing files, grouping and moving files, building, running and exporting projects and so on. For more information on working with IDL projects, see Chapter 22, "Creating IDL Projects" in the *Building IDL Applications* manual.
Macros Menu	The Macro Menu provides functionality for creating new macros and using existing macros in IDL. Fore more about working with macros in IDL, see Chapter 4, "Creating Development Environment Macros".
Window Menu	The Window Menu gives functionality related to Multiple Document Panel windows.
Help Menu	The Help Menu allows you to call IDL Online Help. You can call the entire Online Help system in the IDL Online Help Viewer or find help by topic. For more information on the IDL Help System, see "Getting Help with IDL" on page 35.

Table 2-1: The IDLDE Menus (Continued)

Toolbars

There are three toolbars in the IDLDE: **Standard**, **Run & Debug**, and **Macros**. In addition, when you open an IDL GUIBuilder window (Windows only), its associated toolbar is displayed. When you position the mouse pointer over a toolbar button, the Status Bar displays a brief description. If you click on a toolbar button which represents an IDL command, the IDL command issued is displayed in the Output Log. Display or hide toolbars by making selections among the **Windows** → **Toolbars** items.

Project Window

The Project Window displays information about the current Project you have open in the IDLDE. IDL Projects allow you to easily develop applications in IDL. Through a Project, you can compile, run, and create distributions of your IDL application. The IDL Project Window allows you to access and manage all of the files required for your application. This makes it easier to create a distribution for other developers, colleagues, or users.

For further information on the IDL Projects, refer to Chapter 22, "Creating IDL Projects" in the *Building IDL Applications* manual.

Multiple Document Panel

The section of the main IDL window where IDL Editor windows and GUIBuilder windows are displayed is known as the multiple document panel. Any number of files may be open at a single time. You can access different files from the **Windows** menu by clicking on the appropriate file.

Editor Windows

IDL Editor windows allow you to write and edit IDL programs (and other text files) from within IDL. Any number of Editor windows can exist simultaneously. No Editor windows are open when IDL is first started. Editor windows can be created by selecting **File** → **New** or **File** → **Open**. See "Maximizing the Editor's Capabilities" in Chapter 2 of the *Building IDL Applications* manual for more information on the IDL Editor.

To see the Multiple Document Panel at work, open the file `examples.pro` by typing `.COMPILE examples.pro` at the IDL command line. (See "Command Line" on page 57 for details.)

The following figure shows the IDL program file opened in the Windows IDLDE.

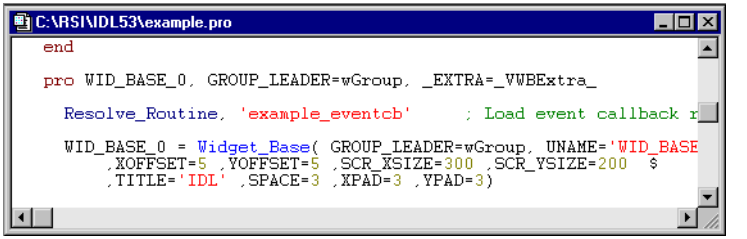


Figure 2-2: Editor Window showing example.pro

GUIBuilder Windows

Under Microsoft Windows, IDL GUIBuilder windows allow you to interactively create user interfaces. Then, you can generate the IDL code that defines the interface and the code to contain the event-handling routines. You can modify the code, compile, and run the application in the IDLDE. To open a GUIBuilder window, you can select **File** → **New** → **GUI** or you can select **File** → **Open**. See Chapter 29, "Using the IDL GUIBuilder" in the *Building IDL Applications* manual. for more information on the GUIBuilder.

Graphics Windows

IDL Graphics windows are not displayed in the Multiple Document Panel, but do appear when you use IDL to plot or display data. You can copy the contents of a Graphics window—iTool, Object or Direct—directly to the operating system clipboard in a bitmap format using **CTRL**+**C**.

When an IDL Graphics window is minimized (iconized), the icon displays the name of the IDL window. This icon appears on the desktop, not in the Multiple Document Panel, as with an iconized Editor window.

Warning

If the backing store is not set when a window is iconized, it will not be refreshed upon return. For more information about setting the backing store for graphics windows, see "Graphics Preferences" on page 104.

Command Line

The Command Line is an IDL prompt where you can enter IDL commands. The text output by IDL commands is displayed in the Output Log window. IDL is an interpreted language and commands entered at the Command Line are executed immediately. To see the IDL Command Line in action, enter the following in the Command Line at the IDL prompt and press Enter:

```
print, 'Hello World!'
```

Figure 2-3: IDLDE Command Line

If you click the right mouse button while positioned over the Command Input Line, a popup menu appears displaying the command history, with a default buffer of 20 entries and a maximum of 100 entries. Select an entry to reissue the command. See "Recalling Commands" in Chapter 2 of the *Building IDL Applications* manual for additional information about the command recall buffer.

Output Log

Output from IDL is displayed in the Output Log window, which appears by default when the IDLDE is first started. Notice the result of our print command in the Output Log in the following figure.

Figure 2-4: The IDL Output Log

If you click the right mouse button while positioned over the Output Log, a context menu appears allowing you to move to a specified error or clear the contents of the Output Log. An additional Windows-only context menu option allows you to copy selected contents.

Variable Watch Window

The Variable Watch window appears by default when you start the IDLDE. It keeps track of variables as they appear and change during program execution (tabs exist for viewing variables by type; Locals, Params, Common and System). For more information about the Variable Watch window, see "The Variable Watch Window" in Chapter 8 of the *Building IDL Applications* manual.

Status Bar

When you position the mouse pointer over a **C**ontrol Panel or Toolbar button, or select an option from a menu in IDLDE, the Status Bar displays a brief description.

Docking/Undocking

In IDL for Windows, four sections of the IDLDE can be moved within and unanchored from the main IDLDE window: the Toolbars, Output Log, Variable Watch Window, and Command Line. Click on the border and drag the left mouse button. You will notice the outline of the chosen section moving with your mouse. When a location is chosen, release the mouse button to dock the window. If you move this outline so that it overlaps an edge of the window space being used by the IDLDE, the section will be docked to the nearest available side of the main IDLDE window. The Toolbars, Output Log, Variable Watch window, and Command Line will remain between the Menu Bar and the Status Bar when docked. They can be docked in any order to an edge. If the outline doesn't overlap an edge, the section will float on the desktop. If you hold down the [**Ctrl**] key, the sections will float instead of docking to the nearest available side of the IDLDE.

Control Panel Buttons

In IDL for UNIX, the Control Panel buttons issue IDL commands for the currently-selected Editor window when pressed. The IDL command issued is displayed in the Output Log. By default, there are three different toolbars and the buttons displayed as well as the commands they issue are completely configurable (see Chapter 3, "Setting IDL Preferences" for more on these toolbars). When you position the mouse pointer over a Control Panel Button, the Status Bar displays a brief description.

File Menu

The following options are available in the **File** menu.

Note

See “Using Keyboard Accelerators” on page 33 for information about using IDL’s keyboard shortcuts on a Macintosh.

Menu Item	Description
New	Select from the following sub-menu items: • **Editor [Ctrl+N]**: Opens a new IDL Editor window. • **GUI** (Microsoft Windows Only): opens a new IDL GUIBuilder file. See Chapter 29, “Using the IDL GUIBuilder” in the *Building IDL Applications* manual for details. • **Project...**: opens the **New Project** dialog. • **Visualization**: Launches an iTool. See “Introducing the iTools” in Chapter 1 of the *iTool User’s Guide* manual. New windows are `Untitled`*n* or `UntitledPrc`*n* (where *n* is the numerical index of the new file) until saved with another name.

Table 2-2: IDLDE File Menu Items

Menu Item	Description
Open... **Ctrl+O**	Select this option to open single or multiple text files for editing. (On Microsoft Windows platforms, you can also select an IDL GUIBuilder `*.prc` portable resource file.) In the **Open** dialog, you can select a continuous range of files by holding down the **Shift** key after selecting the first file, or select multiple, separate files by selecting each file while holding down the **Control** key. A new IDL Editor window is created to contain each text file. **Note -** On Motif platforms, if the **Multiple Windows** option is selected, a new IDL Editor window is created outside the main window to contain each text file. See "Layout Preferences" on page 100 for details. **Note -** You can also open text files from the Command Line. Enter the following at the IDL prompt: `.EDIT file`$_1$ `[file`$_2$ `... file`$_n$`]` where file is the name of the text file you want to open. If the file is not in a directory included in the !PATH system variable, you must enter the full path for file. See ".EDIT" in the *IDL Reference Guide* manual for more information.
Close	Select this option to close the currently-selected IDL Editor window. If you have made changes in an IDL Editor window, you are asked if you want to save the changes before closing the window.
Open Project...	Select this option to open a new IDL Project. The **Open** dialog appears. Select the project you want to open and click **Open**.
Save Project	Select this option to save the current IDL Project. If the Project has not yet been saved, you are prompted for a filename with the **Save As** dialog.
Save Project As...	Select this option to save the current IDL Project to a specified filename. The **Save As** dialog appears.
Close Project	Select this option to close the current IDL Project. If you have made changes in to the project, you are asked if you want to save the changes before closing the window.

Table 2-2: IDLDE File Menu Items (Continued)

Menu Item	Description
Save **Ctrl+S**	Select this option to save the contents of an IDL Editor window. If the file has not yet been saved, you are prompted for a filename with the **Save As** dialog. **Note -** Changes made to a previously-compiled routine are not available to IDL until that routine is re-compiled. Executing the routine without first saving and re-compiling simply re-runs the previously-compiled version, without incorporating recent changes. Select the **Compile** option in the **Run** menu to return to the main program level and re-compile the routine. Select **Compile from Memory** in the **Run** menu to save and compile recent changes to a temporary file.
Save As... **Ctrl+W** (Motif)	Select this option to save the contents of an IDL Editor window to a specified filename. The **Save As** dialog appears. On Windows, when the **File** → **Save As**... option is selected, the default file name is the name of the last procedure or function in the file. On UNIX, the default file name is `*.pro`. For portability between platforms, the filename is lowercase letters.
Revert to Saved	Select this option to reload the last saved version of the document. **Warning -** Unsaved changes are lost without warning.
Generate .pro	Microsoft Windows Only On a Microsoft Windows system, select this option to generate source code files from GUIBuilder interface definitions. When you generate code for the first time, all options open the **Save As** dialog so that you can select a location and specify a filename. The following are generated: • The widget definition code to a `*.pro` file. • The event-handler callback code to a `*_eventcb.pro` file. For information about the IDL GUIBuilder generated code, see "Generating Files" in Chapter 29 of the *Building IDL Applications* manual.

Table 2-2: IDLDE File Menu Items (Continued)

Menu Item	Description
Print... **Ctrl+P**	On Microsoft Windows systems, select this option to immediately print the contents of the currently-selected window to the default printer. On Motif systems, the **Print** dialog appears. Select **Numbered Lines** to include line numbers in the printout. Select **Wrapped Lines** to cause lines longer than the width of the printed page to wrap to a new line. Select **Two Pages** to print two pages per sheet of paper (each logical page is printed at half normal size). Select **Header** to include file information at the top of each page.
Print Setup...	Select this option to change the printer and printing options. The Print (Windows) or Printer Setup (Motif) dialog appears. For further information on setting up a printer, see "Printing in IDL" on page 80.
Recent Files	Select this option to open recently opened or created files. This menu item lists the last ten opened or created files. (On Microsoft Windows systems, it includes both text and GUIBuilder files.) To open a file on this list, select it. On Motif systems, to change the maximum number of files displayed from ten to another number, modify the `idlde.numRecentFiles` resource in your `.idlde` resource file. See Chapter 5, "Customizing IDL on Motif Systems", for details.
Recent Projects	Select this option to open recently opened project files.
Preferences...	Select this option to display the tabbed Preferences dialog, which allows you to customize your interaction with the IDLDE. The options available via the Preferences dialog are described in detail in Chapter 3, "Setting IDL Preferences".
Exit **Ctrl+Q**	Select this option to exit IDL.

Table 2-2: IDLDE File Menu Items (Continued)

Edit Menu

The following options are available in the **Edit** menu.

Note

See "Using Keyboard Accelerators" on page 33 for information about using IDL's keyboard shortcuts on a Macintosh.

Menu Item	Description
Undo **Ctrl+Z** (Windows) **Alt+Z** (Motif)	Select this option to undo previous editing actions. Multiple undo operations are supported; the first reverses the most recent operation, the next reverses the second most recent operation, etc. If the most recent action is irreversible, this option will not be accessible.
Redo **Ctrl+Y** (Windows) **Alt+Y** (Motif)	Select this option to redo previously undone editing actions. Successive redo operations are supported; the first redo reverses the most recent undo, etc.
Cut **Ctrl+X** (Windows) **Alt+X** (Motif)	Select this option to remove currently-selected text from an IDL Editor window or the Command Line to the Windows clipboard.
Copy **Ctrl+C** (Windows) **Alt+C** (Motif)	Select this option to copy the currently-selected text in an IDL Editor window, Output Log window, or Command Line to the clipboard. **Copy** also allows you to copy graphics from an IDL graphics window or draw widget to the clipboard.
Paste **Ctrl+V** (Windows) **Alt+V** (Motif)	Select this option to paste the contents of the Windows clipboard at the current insertion point. The insertion point can only be placed in an IDL Editor window.
Comment	Add the comment character (;) to a line or selected block of text in the Editor window.
Uncomment	Remove the comment character (;) from a line or selected block of text in the Editor window.

Table 2-3: IDLDE Edit Menu Items

Menu Item	Description
Delete **Del**	Select this option to delete the currently-selected text. The deleted text is not placed on the clipboard.
Select All	Use this option to highlight the entire contents of an IDL Editor window.
Clear All **Ctrl+Del** (Windows)	Use this option to clear the entire contents of the current IDL Editor window.
Clear Log **Ctrl+Y** (Motif)	Use this option to clear the entire contents of the Output Log.
Properties	Microsoft Windows Only Select this option to open the GUIBuilder Properties dialog, which you can use to set the attribute and event properties for a widget. For information on the Properties dialog, see "Using the Properties Dialog" in Chapter 29 of the *Building IDL Applications* manual.
Menu	Microsoft Windows Only Select this option to open the GUIBuilder Menu Editor, which you can use to define menus for top-level base widgets and button widgets. For information on the Menu Editor, see "Using the Menu Editor" in Chapter 29 of the *Building IDL Applications* manual.

Table 2-3: IDLDE Edit Menu Items (Continued)

Search Menu

The following options are available in the **Search** menu.

Note

See "Using Keyboard Accelerators" on page 33 for information about using IDL's keyboard shortcuts on a Macintosh.

Menu Item	Description
Find... **Ctrl+F** (Windows) **Alt+F** (Motif)	Select this option to find text in an IDL Editor window or windows. The Search or Find/Replace dialog appears. Enter the text to find in the field marked **Search for** or **Find**; click **Find next** to highlight the search text in the currently active file. **Platform Differences** • On Windows platforms, you can also choose an entry from the pulldown list of recent search terms rather than entering a new term in the **Search for** field. • On Windows platforms, you can specify replacement text by checking the **Replace with** checkbox and entering a replacement term. Click **Replace** to replace the selected text. Check the **Case sensitive** checkbox to match the case of the text you enter. Check **Whole words only** to match only entire words (the default is to match sub-strings). To replace all instances of the search text, check the **Replace all** checkbox and click **Replace**. Select **Forward from cursor** or **Backward from cursor** to specify the direction in which you would like to begin the search, or **Entire file** to search from the beginning of the file. By default, the search will take place in the currently-selected window. Choose a different file or **All Windows** from the pulldown list marked **Search in file** to search other windows.

Table 2-4: IDLDE Search Menu Items

Menu Item	Description
Find Again **F3** (Windows) **Alt+G** (Motif)	Select this option to repeat the previous Find operation.
Find Selection **Ctrl+E** (Windows) **Alt+I** (Motif)	Select this option to find the next occurrence of the selected text in an IDL Editor window.
Enter Selection **Alt+T** (Motif)	Motif Only Select this option to enter selected text in the **Find** field of the Find/Replace dialog.
Replace... **Ctrl+H** (Windows) **Alt+R** (Motif)	Select this option to find text in an IDL Editor window and replace it with new text. The Replace dialog box appears. The Replace dialog has the same controls as the Search dialog, described above in the Find item. By default, the **Replace with** checkbox is checked.
Replace & Find **Alt+P**	Motif Only Select this option to repeat the most recent search-and-replace operation.
Replace Again **Shift+F3**	Select this option to repeat the previous Replace operation.
Go To Line... **Ctrl+G**	Select this option to jump directly to the specified line number in an IDL Editor window. The Go To Line dialog appears.
Go To Definition **Ctrl+D** (Windows) **Ctrl+T** (Motif)	Use this option to go to and mark with a current line indicator (blue arrow) the procedure or function call of the item next to which the cursor is positioned. The item must be either user-defined or a procedure or function written in IDL, and must have been compiled during the current IDLDE session.

Table 2-4: IDLDE Search Menu Items (Continued)

Run Menu

Run menu items are enabled when an IDL program is loaded into an IDL Editor window and compiled. If you click the right mouse button while positioned over an editor window, a popup menu appears allowing you to quickly access several of the most convenient commands. The popup menu changes to display common debugging commands if IDL is running a program. See Chapter 8, "Debugging and Error-Handling" in the *Building IDL Applications* manual for more information.

Note

See "Using Keyboard Accelerators" on page 33 for information about using IDL's keyboard shortcuts on a Macintosh.

Menu Item	Description
Compile *filename*.pro **Ctrl+F5**	Select this option to compile a `.pro` file. The currently-selected file is only recognized as an IDL procedure or function if suffixed with `.pro`. Selecting this option is the same as entering `.COMPILE` at the Command Line, with the appropriate Editor window selected in the Multiple Document Panel. You can also compile files from the Command Line. Enter the following at the IDL prompt: `.COMPILE file1 [file2 ... filen]` where `file` is the name of the file you want to open. IDL opens your files in editor windows and compiles the procedures and functions contained therein. If the path is not specified in the **Path Preferences** from the **File** menu, you must enter the full path for file. See ".COMPILE" in the *IDL Reference Guide* manual for a more detailed explanation.

Table 2-5: IDLDE Run Menu Items

Menu Item	Description
Compile *filename*.pro from Memory **Ctrl+F6**	Select this option to save and compile changes to the current editor window without affecting the last-saved version of the file. The temporary file created allows you to experiment without committing changes to the permanent file. Selecting this option is the same as entering `.COMPILE -f` at the Command Line. See ".COMPILE" in the *IDL Reference Guide* manual for a more detailed explanation.
Compile All	Select this option to compile all currently open `*.pro` files.
Run *filename* **F5**	Select this option to execute the file called `filename` contained in the currently-active editor window. Selecting this option is the same as entering the procedure name at the Command Line or using the .GO executive command at the Command Line. If the file is interrupted while running, selecting this option resumes execution; it is the same as entering .CONTINUE at the Command Line. For more information, see .CONTINUE and .GO in the *IDL Reference Guide*. **Warning -** In order for the **Run** option to reflect the correct procedure name in the **Run** menu, the `.pro` filename must be the same as the main procedure name. For example, the file named `squish.pro` must include: `pro squish`
Resolve Dependencies **Alt+F5** (Motif)	Select this option to iteratively compile all un-compiled IDL routines that are referenced in any open and compiled files. Selecting this option is the same as entering `RESOLVE_ALL, /QUIET` at the Command Line. The `QUIET` keyword suppresses informational messages. See "RESOLVE_ALL" in the *IDL Reference Guide* manual for a more detailed explanation.

Table 2-5: IDLDE Run Menu Items (Continued)

Menu Item	Description
Profile	Select this option to access the Profile dialog. The IDL Code Profiler allows you to analyze the performance of your applications. You can identify which modules are used most frequently, and which modules take up the greatest amount of time.For more information about the IDL Code Profiler, see "The IDL Code Profiler" in Chapter 10 of the *Building IDL Applications* manual.
Test GUI **Ctrl+T**	Microsoft Windows Only Select this option to test the GUI interface in a GUIBuilder window. This option allows you to see how the interface you have designed will look when it is running. To exit test mode: Press the **Esc** key. or Click the **X** in the upper-right corner of the application window of the running test application. **Note -** This option is not available if a blocking widget is currently active.
Break **Ctrl+Break** (Windows) **Ctrl+C** (Motif)	Select this option to interrupt program execution. IDL inserts a marker to the left of the line at which program execution was interrupted.

Stop
Ctrl+R
	Select this option to stop program execution and return to the main program level. Selecting this item is the same as entering the following at the Command Line:
	`RETALL`
	`WIDGET_CONTROL,/RESET`
	`CLOSE, /ALL`
	`HEAP_GC, /VERBOSE`
	See RETALL, WIDGET_CONTROL, CLOSE, or HEAP_GC in the *IDL Reference Guide* for details.

Table 2-5: IDLDE Run Menu Items (Continued)

Menu Item	Description
Reset	Select this option to completely reset the IDL environment. This option executes .RESET_SESSION. See ".RESET_SESSION" in the *IDL Reference Guide* manual for details.
Step Into **F8**	Select this option to execute a single statement in the current program. The current-line indicator advances one statement. If the statement being stepped into calls another IDL procedure or function, statements from that procedure or function are executed in order by successive Step commands. Selecting this item is the same as entering `.STEP` at the IDL Command Line. See ".STEP" in the *IDL Reference Guide* manual for a more detailed explanation.
Step Over **F10**	Select this option to execute a single statement in the current program. The current-line indicator advances one statement. If the statement which is stepped over calls another IDL procedure or function, statements from that procedure or function are executed to the end without interactive capability. Selecting this item is the same as entering `.STEPOVER` at the IDL Command Line. See ".STEPOVER" in the *IDL Reference Guide* manual for details.
Step Out **Ctrl+F8**	Select this option to continue processing until the current program returns. Selecting this item is the same as entering `.OUT` at the IDL Command Line. See ".OUT" in the *IDL Reference Guide* manual for a more detailed explanation.

Table 2-5: IDLDE Run Menu Items (Continued)

Menu Item	Description
Trace...	Select this option to access the Trace Execution dialog. You can modify the interval between successive .STEP or .STEPOVER commands, depending on whether **Step into routines** or **Step over routines** is checked. The current-line indicator points to program lines as they are executed. Selecting this item at full speed is the same as entering `.TRACE` at the IDL command prompt. See ".TRACE" in the *IDL Reference Guide* manual for a more detailed explanation.
Run to Cursor **F7**	Select this option to execute statements in the current program up to the line where the cursor is positioned. Selecting this item is the same as setting a one-time breakpoint at a specific line. See "BREAKPOINT" in the *IDL Reference Guide* manual for details.
Run to Return **Ctrl+F7**	Select this option to execute statements in the current procedure or function up to the line where the return is positioned. Selecting this item is the same as setting a one-time breakpoint at a specific line. See ".RETURN" in the *IDL Reference Guide* manual for details.
Set Breakpoint **Clear Breakpoint** **F9**	Select this option to set or clear a breakpoint on the current line. See Chapter 8, "Debugging and Error-Handling" in the *Building IDL Applications* manual for details.
Disable Breakpoint **Ctrl+F12** (Motif)	Select this option to access disable a breakpoint in the current line. See Chapter 8, "Debugging and Error-Handling" in the *Building IDL Applications* manual for details.
Edit Breakpoint...	Select this option to access the **Edit Breakpoint** dialog. See Chapter 8, "Debugging and Error-Handling" in the *Building IDL Applications* manual for details.
Up Stack **Ctrl+Up Arrow**	Select this option to move up the call stack by one.

Table 2-5: IDLDE Run Menu Items (Continued)

Menu Item	Description
Down Stack **Ctrl+Down Arrow**	Select this option to move down the call stack by one.
List Call Stack	Select this option to display the current nesting of procedures and functions. Selecting this item is the same as entering `HELP, /TRACEBACK` at the IDL Command Line. See "HELP" in the *IDL Reference Guide* manual for details.

Table 2-5: IDLDE Run Menu Items (Continued)

Project Menu

For more information on the following **Project** menu items, see Chapter 22, "Creating IDL Projects" in the *Building IDL Applications* manual.

Note

See "Using Keyboard Accelerators" on page 33 for information about using IDL's keyboard shortcuts on a Macintosh.

Menu Item	Description
Add/Remove Files...	Select this option to add or remove files from the current project.
Remove Selected Ctrl+H	Motif Only Select this option to remove the currently selected file from your IDL Project.
Move To	Motif Only Select this option to move the currently selected file to the indicated project directory.
Groups...	Selecting this option displays the **Project Groups** dialog from which you can create a new group or rename, remove, move up or down, or set to filter specific file types for the default groups within an IDL Project.
Options...	Select this option to change the options for a project. The **Project Options** dialog is displayed.
Compile	Select this option to compile files in a project. You can choose either **All Files** to compile all the source files in a project or **Modified Files** to compile only the files that have been modified since the last compile.
Build	Select this option to build your project.
Run	Select this option to run the project application.
Export	Select this option to export your project.

Table 2-6: IDLDE Project Menu Items

Macros Menu

The following options are available in the **Macros** menu.

Menu Item	Description
Edit...	Select this item to access the **Edit Macros** dialog. Macros which have already been defined are listed in the **Macros:** field. To edit a macro, click on the macro to access its characteristics and click **OK** when your adjustments are complete. To add a macro, click **Add...**, which will access the **Add Macro** dialog. Enter the name of the new macro in the given field and click **OK**. Enter the IDL command to be executed by the new macro in the **IDL Command:** field. Enter the menu item name, the full path to the toolbar bitmap file, the tooltip text, and the status bar text in the appropriate fields. Select the accelerator by specifying the key in the **Key:** field and then optionally clicking on any combination of **Ctrl**, **Alt** and **Shift**. **Note -** Bitmap files for toolbar buttons must be 16 pixels by 16 pixels, and must contain 256 colors or fewer. To remove a macro, click **Remove**. To change the position of a macro in the **Macro** menu and on the **Macro Toolbar**, click on the macro to highlight it and click on either **Move Up** or **Move Down**.
Import...	Microsoft Windows Only Use this menu selection to display the **Import Macros** dialog box. Use this dialog to select the previous IDL installation from which you want macros to be imported.

Table 2-7: IDLDE Macros Menu Items

Menu Item	Description
Print Var (Windows) **Print Variable** (Motif)	Select this option to print the selected variable. Selecting this item is the same as entering `PRINT, x` at the IDL Command Line, where `x` is the selected variable.
Help On Var (Windows) **Help On Variable** (Motif)	Select this option to list attributes of the selected variable. Selecting this item is the same as entering `HELP, x, /STRUCTURE` at the IDL Command Line, where `x` is the selected variable.
Import Image	Select this option to import an image file into IDL. For more information, see "Using Macros to Import Image Files" on page 165.
Import ASCII	Select this option to import an ASCII file into IDL. For more information, see "Using Macros to Import ASCII Files" on page 167.
Import Binary	Select this option to import a binary file into IDL. For more information, see "Using Macros to Import Binary Files" on page 169.
Import HDF	Select this option to import an HDF file into IDL. For more information, see "Using Macros to Import HDF Files" on page 170.
Demo	Select this option to access IDL's Demo application.

Table 2-7: IDLDE Macros Menu Items (Continued)

Window Menu

The following options are available in the **Window** menu.

Note

See "Using Keyboard Accelerators" on page 33 for information about using IDL's keyboard shortcuts on a Macintosh.

Menu Item	Description
Read Only	Motif Only Select this option to enable or disable editing of the currently selected window. A filled square next to the item indicates Read-Only status.
Next **F6** (Windows) **F11** (Motif)	Select this option to shift IDL's focus to the next numbered editor window.
Previous **Shift+F6** (Windows) **Alt+F11** (Motif)	Select this option to shift IDL's focus to the previous numbered editor window.
Cascade	Select this option to cascade all the IDL **Editor** windows within the main window.
Tile Horizontally	Microsoft Windows Only Select this option to tile all the IDL **Editor** windows on top of one another within the main window.
Tile Vertically	Microsoft Windows Only Select this option to tile all the IDL **Editor** windows side-by-side within the main window.
Tile	Motif Only Select this option to arrange all open windows in a non-overlapping fashion.

Table 2-8: IDLDE Window Menu Items

Menu Item	Description
Arrange Icons	Select this option to arrange all minimized **Editor** or **Graphics** windows.
Close All	Select this option to close all IDL **Editor** windows. If the text within an IDL **Editor** window has changed, you are asked if you want to save the file before closing.
Configure	Motif Only Select this option to access a pulldown menu which alters the appearance of the IDLDE. Select each toggle option to hide or show each component. For more information about each component, see "Components of the IDLDE" on page 52. • **Hide Control** (Show Control) • **Hide View** (Show View) • **Hide Log** (Show Log) • **Hide Variable Watch** (Show Variable Watch) • **Hide Command** (Show Command) • **Hide Status** (Show Status) • **Hide Project** (Show Project)
Command Input **Ctrl+I**	Microsoft Windows Only If this menu item has a check mark by it, the IDL **Command Line** is visible in the main IDL window. If this item does not have a check mark next to it, the IDL command line is not visible. Use this menu item to toggle between the two states.
Output Log **Ctrl+L**	Microsoft Windows Only If this menu item has a check mark by it, the **Output Log** is visible in the main IDL window. If this item does not have a check mark next to it, the **Multiple Document Panel** is maximized in the main IDL window. Use this menu item to toggle between the two states.

Table 2-8: IDLDE Window Menu Items (Continued)

Menu Item	Description
Variable Watch **Ctrl+A**	Microsoft Windows Only If this menu item has a check mark by it, the **Variable Watch Window** is visible in the main IDL window. If this item does not have a check mark net to it, the **Variable Watch Window** is not visible. Use this menu item to toggle between the two states.
Project	Microsoft Windows Only If this menu item has a check mark by it, the **Project Window** is visible in the main IDL window. If this item does not have a check mark net to it, the **Project Window** is not visible. Use this menu item to toggle between the two states.
Toolbars	Select this option to access a pulldown menu with the three Windows toolbars: **Standard**, **Run & Debug**, and **Macros**. If a toolbar has a check mark by it, it is visible below the menu bar items.
Status Bar	Microsoft Windows Only If this menu item has a check mark by it, the **Status** bar is visible at the very bottom of the Main IDL window.
Numbered Windows	The numbered menu items at the bottom of the **Window** menu display open files. Select any of these menu items to make that window the current window.

Table 2-8: IDLDE Window Menu Items (Continued)

Help Menu

The following options are available in the **Help** menu.

Note

See "Using Keyboard Accelerators" on page 33 for information about using IDL's keyboard shortcuts on a Macintosh.

Menu Item	Description
Contents **Ctrl+F1**	Select this menu item to display the IDL **Online Help** Viewer.
Find Topic... **F1**	Select this menu item to display the **Search** dialog for IDL **Online Help**.
About IDL...	Select this option to display information on the IDL version in use.

Table 2-9: IDLDE Help Menu Items

Printing in IDL

IDL allows you two ways to print:

- Printing graphics from the IDL language
- Printing IDL source code from the **File** menu of the IDLDE.

While these sources are fundamentally different, the methods used to specify and configure a print device according to your operating system are the same. This topic is covered in the following sections. See "Printing Graphics" on page 222 for information on how to print from an IDL program.

Printer setup in Windows is relatively straightforward, and is described in the following section. UNIX printer setup is slightly more involved and is covered in "IDL Printer Setup in UNIX or Mac OS X" on page 81.

IDL Printer Setup in Windows

Setting up a printer in IDL for Windows uses the common Windows Printer Setup dialog. For more information on setting up a Printer on Windows, see your Windows operating system documentation or support.

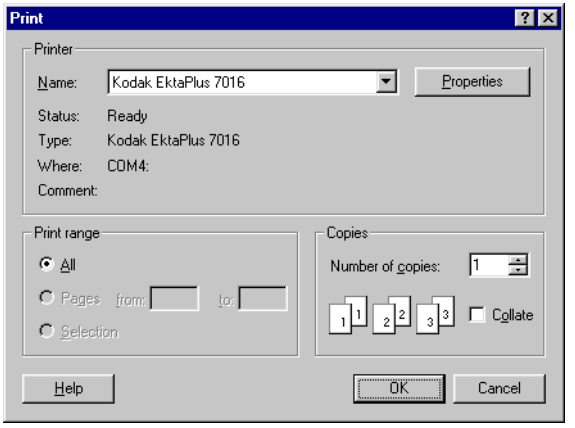


Figure 2-5: Common Printer Setup Dialog in Windows

IDL Printer Setup in UNIX or Mac OS X

IDL for UNIX uses the Xprinter print technology from Bristol Technology to create and output information to a wide variety of printers. This section describes the Xprinter setup dialogs.

The Xprinter Setup Dialog

The Xprinter Setup dialog allows you to select model-specific printer options such as paper trays, paper size, page orientation, and the UNIX print spooler command. Printer options are saved in the `$HOME/.XprinterDefaults` file. Once configured, the desired information is saved to the file system and used in future IDL sessions.

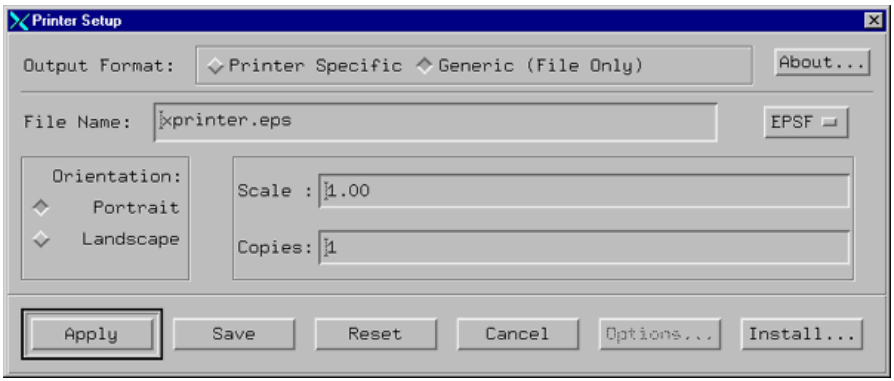


Figure 2-6: The Printer Setup Dialog

Printer Setup Dialog Buttons

The action area of the Printer Setup dialog contains six buttons:

Button	Description
OK	Writes current configuration information to your default printer information file `$HOME/.XprinterDefaults`. This button also dismisses the dialog.

Table 2-10: Printer Setup Dialog Buttons

Button	Description
Save	Writes current configuration information to your default printer information file `$HOME/.XprinterDefaults`.
Reset	Reloads default configuration from `$HOME/.XprinterDefaults`.
Cancel	Closes dialog and cancels all configuration changes.
Options	Displays the options dialog box that lets you select an alternate printer setup. This button is disabled if output is configured to be sent to a file instead of a printer.
Install	Displays the installation dialog box that allows you to add or remove printer devices and printer ports from the `$HOME/.XprinterDefaults` file.

Table 2-10: Printer Setup Dialog Buttons (Continued)

Configuring Printer Setup Options

Specify the following options on the initial Printer Setup dialog:

Option	Description
Output Format:	Specify whether to send output to a file or a printer. If you choose Printer Specific, you can send output to any printer type/port combination configured in your `$HOME/.XprinterDefaults` file. If the port is FILE:, Xprinter creates an output file for the specified printer type. If you choose Generic (File Only), print output is sent to an Encapsulated PostScript or generic PCL file.
Printer:	This field appears only if you select Output Format: Printer Specific. It specifies the name of the default printer type/port to which to send print output. Click the Options button to specify a different printer type/port combination.

Table 2-11: Specifying Printer Setup Options

Option	Description
File Name:	This field appears only if you choose Output Format: Generic (File Only). Type the name of the print file you wish to create. To pipe print output to a command, enter a `!` character as the first character and then specify the command to which to send output. For example, to send output to the lp command, enter the following: `!lp`
EPSF **PCL4** **PCL5**	This field only appears if you select Output Format: File. Click this button to display a list of output file types and select the desired type. Available types are EPSF (Encapsulated PostScript), PCL4, and PCL5.
Orientation	Specify portrait or landscape.
Scale	To increase the size of the output, specify a value greater than 1.00. To reduce the size, specify a value less than 1.00. For example, a value of 2.00 would double the size of the output; a value of 0.50 would reduce it by half.
Copies	Specify the number of copies to print.

Table 2-11: Specifying Printer Setup Options (Continued)

To set additional options, such as selecting a different printer or changing the page size, click the **Options** button. The Options dialog appears.

The Options dialog is only available when sending output to a printer.

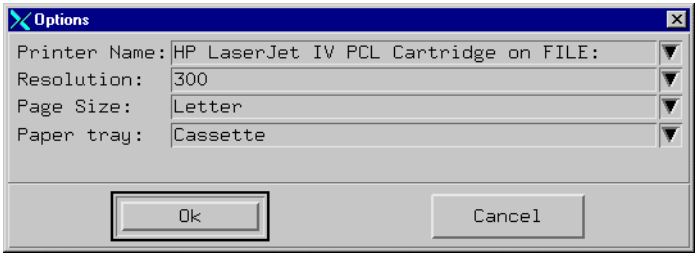


Figure 2-7: The Options Dialog

Use this dialog to set the Printer Setup options:

Option	Description
Printer Name	Use this field to select the current printer. Click the down arrow to display a list of configured printers.
Resolution	Specify printer resolution with this field. Values vary depending on printer.
Page Size	Specify paper size with this field. Values vary depending on printer.
Paper tray	Specify paper tray with this field. Values vary depending on printer.
Duplex	Specify duplex options (if the selected printer supports duplex printing). Valid values include None (no duplex printing), Duplex Tumble (flips over the short edge), and Duplex No Tumble (flips over the long edge). If the selected printer does not support duplexing, this field is disabled.

Table 2-12: The Printer Setup Options

Adding a New Printer to the List of Printer Choices

To add a new printer to your list of available printers:

- Define a port, which is an alias for the print command.
- Associate the port with the printer's PPD file.

Defining a New Port

To define a new port using the Printer Setup dialog:

1. Display the Ports dialog. From the Printer Setup dialog, select **Install**, **Add Printer**, and **Define New Port**.

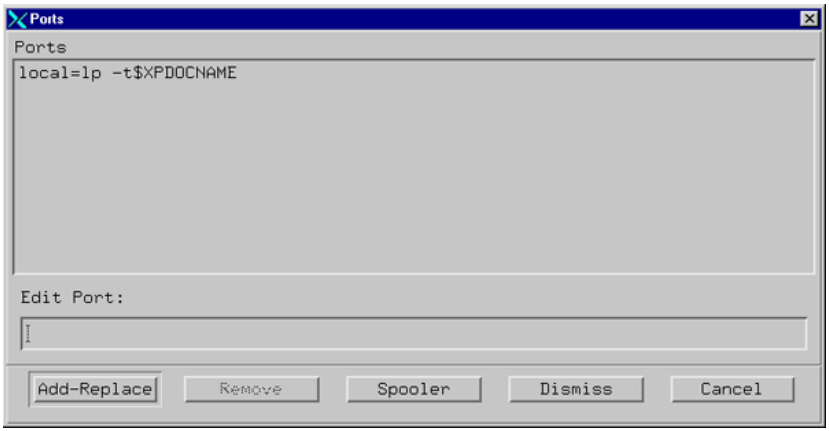


Figure 2-8: Defining a New Port

2. Type the port definition in the **Edit Port** edit box. Port definitions have the following format:

   ```
   port=print_command
   ```

   The print_command is the command for sending output to the printer port. If you were to have two printers named ORION and SIRIUS for example, the definitions would appear as follows:

   ```
   ORION=rsh bandit "lp -d ps"
   SIRIUS=rsh bandit "lp -d ps -T pcl5"
   ```

   Both printers here are connected to the system bandit, so the print command is a remote shell command executed on bandit. ORION is a PostScript printer, so the command `lp -d ps` is executed on bandit to print to ORION. SIRIUS though is a PCL5 printer, so the print command executed on bandit to print to SIRIUS is `lp -d ps -T pcl5`.

3. Click **Add/Replace** and the new port is now included in the list of current port definitions.

4. Repeat the above step for each printer to which you wish to send output.

Note

To create a printer port for each available queue on hp700 systems, click the Spooler button on the Ports dialog. This command creates a default printer port for each available printer queue returned by the `lpstat -a` command.

Modifying an Existing Port

In order to modify an existing port using the Printer Setup dialog:

1. Display the Ports dialog. From the Printer Setup dialog, click **Install**, **Add Printer**, and **Define New Port**.
2. Select the port you wish to modify and edit the port information in the **Edit Port** edit box.
3. Click **Add/Replace**. The modified port is now included in the list of current port definitions.

Matching a Printer Device to a Port

In order to match a printer device to a port using the Printer Setup dialog:

1. Display the Add Printer dialog. From the Printer Setup dialog, click **Install** and **Add Printer**.
2. In the **Printer Devices** field, select the description that matches the printer you are to install. If no description matches this printer, contact your printer vendor for a printer description (PPD) file.
3. Select the desired port in the Current Port Definitions list box and click Add Selected. The new printer is now included in the list of currently installed printers.

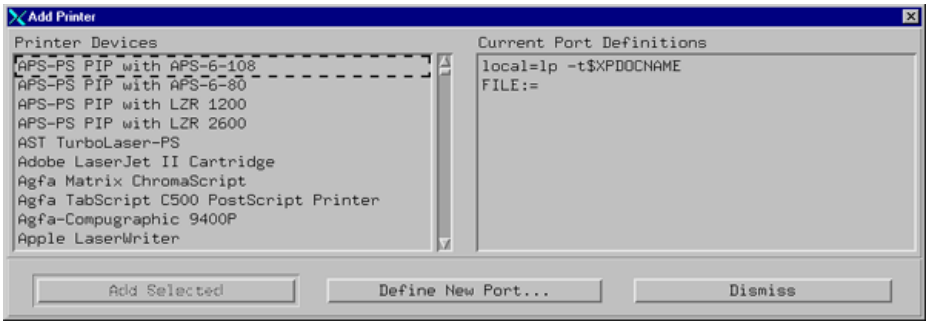


Figure 2-9: Adding a Printer

Removing an Installed Printer

In order to remove a printer device/port combination using the Printer Setup dialog:

1. Display the Printer Installation dialog. From the Printer Setup dialog, click **Install**.
2. In the **Currently Installed Printers** list box, select the printer you wish to remove and click on **Remove Selected**.

Manually Modifying Default Printer Setup Values

Xprinter retrieves default printer setup information from the file `.XprinterDefaults` in your home directory. If this file does not exist, Xprinter reads the information from the file `$XPHOME/xprinter/XprinterDefaults` or `$XPPATH/XprinterDefaults`.

Note

For IDL, `$XPATH` is set to `$IDL_DIR/resource/xprinter`.

The Xprinter Printer Setup dialog writes modifications to the default information in `$HOME/.XprinterDefaults`. However, it never modifies the default information in the file `$XPHOME/XprinterDefaults` or `$XPPATH/XprinterDefaults`. If the file `$HOME/.XprinterDefaults` does not already exist, the Xprinter Printer Setup dialog creates it.

Although the most common way to modify the default Printer Setup is using the Printer Setup dialog, which updates `$HOME/.XprinterDefaults` automatically, you may also edit this file with any text editor and make changes directly.

You may also set up the `$HOME/.XprinterDefaults` file to do the following:

- Define printer ports.
- Match printer types to defined ports.
- Specify the default printer.
- Specify printer-specific options.

Defining a Port

A printer port is an alias for the print command. It is defined in the [ports] section of `$HOME/.Xpdfaults` and appears as part of the Printer Name in the Printer Setup dialog. For instance, the following is the first Printer Name in the Printer Setup dialog before you make any changes to `$HOME/.XprinterDefaults`:

```
AppleLaserWriter v23.0 PostScript on FILE:
```

For this Printer Name, FILE: is the port name. To send output to a printer instead of a file, you first must define a port for each printer to which you wish to direct output. Port entries in the [ports] section have this format:

```
port=print_command
```

The print_command is the command for sending output to the printer port. For instance, if you have two printers (ORION and SIRIUS), your [ports] section may appear as follows:

```
[ports]
ORION=rsh bandit "lp -d ps"
SIRIUS=rsh bandit "lp -d ps -T pcl5"
```

In the above, both printers are connected to the system bandit, so the print command is a remote shell command executed on bandit. ORION is a PostScript printer, so the command `lp -d ps` is executed on bandit to print to ORION. SIRIUS, though, is a PCL5 printer, and thus the print command executed on bandit to print to SIRIUS is `lp -d ps -T pcl5`.

If a printer is connected to your local system, you will need to add an entry for that printer as well. For the local printer, your entry should be like the following:

```
[ports]
ORION=rsh bandit "lp -d ps"
SIRIUS=rsh bandit "lp -d ps -T pcl15"
LOCAL=lp -d ps
```

Your printer port can be any name you choose except FILE:, which is the only reserved port name. It causes Xprinter to create a print file formatted specifically for the specified printer type.

An entry must be created in the [ports] section for every printer to which you wish to be able to print.

Matching a Printer Type to a Defined Port

After you have defined a port for each printer, you must tell Xprinter what type of printer is associated with each port. List device types in the [devices] section of the `.XprinterDefaults` file. Each entry in the [devices] section has the following format:

```
alias=PPD_file driver,port
```

Note

There must be a space between the PPD_file and driver and a comma between the driver and the port. The following table describes each part of this entry.

Field	Description
alias	The alias is a descriptive name used to identify the printer. It can be anything you choose. The alias is the name which appears in the Printer Setup dialog (such as `HP LaserJet III SI PostScript`).
PPD_file	The PPD_file is the name of the printer description (PPD) file used by the printer, without a `.PPD` extension. Search in the directory `$XPHOME/xprinter/ppds/` to find the PPD file for your printer.
driver	The driver is the type of driver your printer uses. Value values are `PostScript`, `PCL4`, and `PCL5`.
port	The port is the printer port as listed in the [ports] section of the `.XprinterDefaults` file (ORION, SIRIUS, and LOCAL in the example [ports] section).

Table 2-13: Associating a Printer with a Port

Here's an example configuring three printers:

Port	Printer Type	Output Type
ORION	HP LaserJet IIISi PostScript v52.3	PostScript
SIRIUS	HP LaserJet 4M PCL Cartridge	PCL
LOCAL	QMS-PS 2200 v52.3	PostScript

Table 2-14: Example Configuration

First, be sure to choose an alias for each printer. In order to make it simpler to identify the printer from the Printer Setup dialog you wish to use, you may use the following aliases:

```
HP LaserJet PS
HP LasterJet PCL
QMS PS
```

It is important to note that if you utilize the Printer Setup dialog to associate ports and PPD files, you cannot specify a printer alias. You must instead choose an alias from the predefined listing that appears in the Printer Devices list box in the Add Printer dialog. The corresponding PPD file is already associated with the printer aliases in this list box.

Now, identify the PPD file associated with each of these printers.

Thus the [devices] section of the `.XprinterDefaults` file would be as follows:

```
[devices]
HP LaserJet PS=HP3SI523 PostScript,ORION
HP LaserJet PCL=HP4M PCL,SIRIUS
QMS PS=Q2200523 PostScript,LOCAL
```

After these entries have been added to your `.XprinterDefaults` file, the following printer choices are available from the Printer Setup dialog:

```
HP LaserJet PS on ORION
HP LaserJet PCL on SIRIUS
QMS PS on LOCAL
```

Specifying a Default Printer

After you have configured all available printers, you may select one of them as the default printer. To make a specific printer the default printer on the Printer Setup dialog, add an entry (in the following format) to the [windows] section of the `.XprinterDefaults` file:

```
[windows]
device=PPD_file,driver,port
```

Simply provide the same information that you used in the [devices] section. Only the format of the entry is different; there is a comma between the PPD_file and the driver instead of a space.

For example, suppose you wish the default printer to be the printer at port ORION. The [windows] section would appear as follows:

```
[windows]
device=HP3SI523,PostScript,ORION
[ports]
ORION=rsh bandit "lp -d ps"
SIRIUS=rsh bandit "lp -d ps -T pcl5"
LOCAL=lp -d ps
[devices]
HP LaserJet PS=HP3SI523 PostScript,ORION
HP LaserJet PCL=HP4M PCL,SIRIUS
QMS PS=Q2200523 PostScript,LOCAL
```

In your default `.XprinterDefaults` file, the [windows] entry appears:

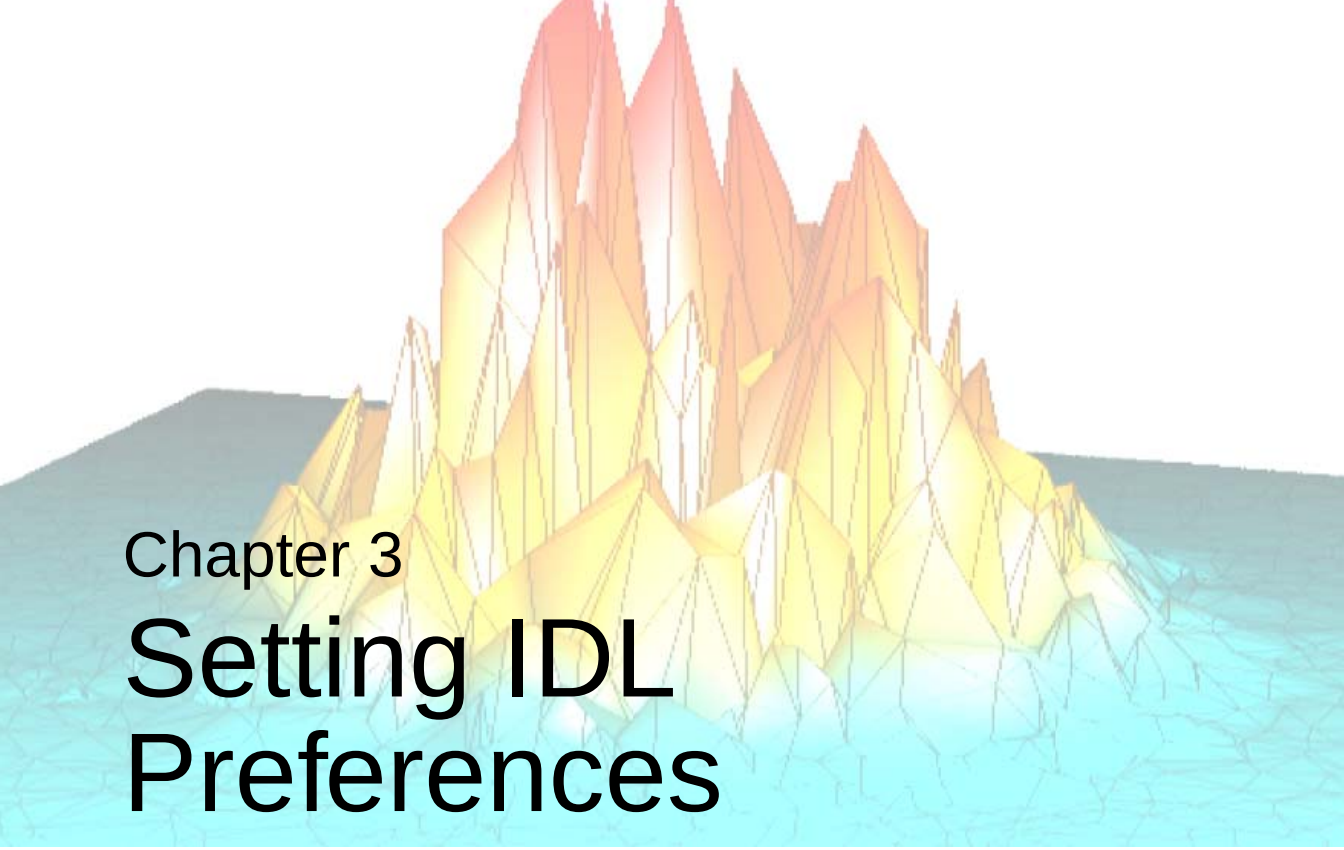
```
[windows]
device=NULL,PostScript,FILE:
```

Since no PPD file is listed (NULL), the default on the Printer Setup dialog is to print generic PostScript to a file. You may specify the filename and change the type of output to PCL on the Printer Setup dialog.

Specifying Printer-Specific Options

You may include a section that lists the default printer-specific options for each printer defined in the devices section. The options available vary between differing printers, but typical options include number of copies, page size, paper tray, and orientation. An example follows of a printer-specific section for a default printer in the example `.XprinterDefaults` file:

```
[HP3SI523,PostScript]
Scale=0.80
Copies=1
PaperTray=Lower
PageSize=Letter
Orientation=Portrait
DPI=300
```



Chapter 3

Setting IDL Preferences

The IDL Development Environment can be customized by setting *preferences*. This chapter describes the sections of the **Preferences** dialog:

About IDL Preferences 94
Customizing IDL 95
General Preferences 97
Layout Preferences 100
Graphics Preferences 104
Editor Preferences 107
Startup Preferences 110
Font Preferences 112
Path Preferences 115

About IDL Preferences

Preferences are internal values that control various aspects of the environment that IDL presents to its users. Preferences supply initial values for many system variables and control the layout of the IDL development environment (IDLDE) and a variety of other aspects of IDL's behavior. Preferences can be specified from a variety of sources. They persist between IDL sessions, meaning that once you get them set in a way that satisfies your needs, you can forget them, and IDL will behave in the way you have specified every time you run it.

You can specify values for many of the IDL preferences through the IDLDE's Preferences dialog. For more information, see "Customizing IDL" on page 95.

Some preferences are not visible in the Preferences dialog. To customize them, use the IDL PREF_* routines, environment variables, or user preference files to specify preference/value pairs. You can also use these mechanisms to modify preference values visible in the Preferences dialog. For more information, see Appendix E, "IDL Preferences" in the *IDL Reference Guide* manual.

Unavailable Preferences

The value of a preference can come from a variety of sources. There is a hierarchy to these sources, and IDL will use the value from the source with the highest priority.

Preferences specified at the command line when launching IDL have the highest priority, followed by preferences specified in environment variables. If a preference takes its value from either of these sources, you will not be able to change the preference's value during the course of the IDL session, and the value will be desensitized in the **Preferences** dialog.

See "Understanding Preference Sources" in Appendix E of the *IDL Reference Guide* manual for additional information about the hierarchy of preference sources.

Customizing IDL

Various settings for the IDL Development Environment can be customized using the Preferences dialog. To open the **Preferences** dialog, select **Preferences** from the IDL Development Environment **File** menu.

Note

On UNIX platforms, including Macintosh OS X, some settings can also be customized by editing IDL's resource files. For further information about editing resource files on UNIX and Macintosh OS X, see Chapter 5, "Customizing IDL on Motif Systems".

The **Preferences** dialog contains tabbed sections that allow you to customize your interaction with the IDLDE. The tabs and their uses are described below.

Note

The terminology used on the Preferences dialogs differs between Microsoft Windows and Motif systems. In this documentation, if the wording is significantly different between the two platforms, the wording used in the Windows dialogs is listed first, followed by the wording used in the Motif dialogs.

Tab	Description
General Preferences	This tab allows you to specify how the IDLDE session begins and ends, to control the number of lines in the recall buffer and the Output Log, and to designate how the files should be opened and read.
Layout Preferences	This tab allows you to specify the location and size of the main IDLDE window on the screen. You can also designate which components of the IDLDE will be visible.
Graphics Preferences	This tab allows you to set the layout of windows that contain IDL graphics, and to specify the backing store, the size of the TrueType font cache, and the object graphic rendering preference.
Editor Preferences	This tab allows you to customize the IDL's built-in editor and also offers several compiling options.

Table 3-1: Preference Dialog Tabs

Tab	Description
Startup Preferences	This tab allows you to specify the locations of the working directory and a startup file.
Font Preferences	This tab allows you to specify different fonts, styles, and sizes for the Editor, Command Line and Output Log.
Path Preferences	This tab allows you to specify the IDL Files Search Path and path cache settings.

Table 3-1: Preference Dialog Tabs (Continued)

Platform Differences

Microsoft Windows and UNIX platforms (including Macintosh OS X) implement the **Preferences** dialog using different dialog application buttons. The following table lists the buttons, the platforms on which they are found in the **Preferences** dialog, and the action performed when the button is used.

Platform	Button	Result
Windows, UNIX	OK	Changes are saved and applied to the current session, and the **Preferences** dialog is dismissed.
	Cancel	Any changes that were not applied are ignored, and the **Preferences** dialog is dismissed.
	Apply	Changes are applied to the current session, but not saved. (On UNIX, changes to items marked on the dialog with an asterisk take effect in the next session. To make the changes for the current session, use **OK**.) The **Preferences** dialog remains visible.
	Help	Displays IDL online help.
Windows only	Reset	Restores the preferences on the dialog to the preference values from the start of the current IDL session.

Table 3-2: Preferences Dialog Button Descriptions

General Preferences

The **General** tab of the Preferences dialog has three sections: **Program**, **Log and Command Window**, and **Files**.

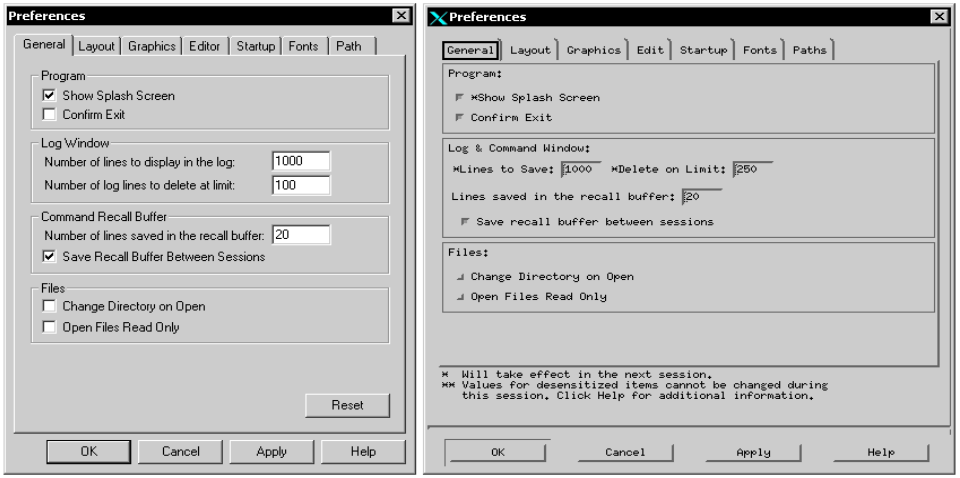


Figure 3-1: General Preferences Dialog

Note

Some preference settings may be desensitized. See "Unavailable Preferences" on page 94 for details.

Program Section

You can specify how IDL handles starting up and exiting. Click on the following check boxes to apply or disable the options:

- **Show Splash Screen** — Select this option to show the IDL splash screen on startup. This selection takes effect the next time an IDL session is started.

  This control sets the value of the IDL_WDE_SPLASHSCREEN preference (Windows) and the IDL_MDE_SPLASHSCREEN preference (UNIX). For more information, see Appendix E, "IDL Preferences" in the *IDL Reference Guide* manual.

- **Confirm Exit** — Select this option to display a warning dialog when you exit IDL.

This control sets the value of the IDL_WDE_EXIT_CONFIRM preference (Windows) and the IDL_MDE_EXIT_CONFIRM preference (UNIX). For more information, see Appendix E, "IDL Preferences" in the *IDL Reference Guide* manual.

Log and Command Window Section

Note

On Microsoft Windows systems, these preferences are divided between the Log Window and Command Recall Buffer sections of the dialog.

The number of lines saved in the recall buffer for the Command Line has an impact on the performance of IDL. The amount of memory required for greater numbers of saved lines in the buffer affects the speed at which IDL runs. Click in the field next to each description and enter your adjusted value to change the settings.

- **Number of lines to display in the log / Lines to Save** — This field controls the maximum number of lines retained by the **Output Log** window. The default is 1000 lines.

  This control sets the value of the IDL_WDE_LOG_LINES preference (Windows) and the IDL_MDE_LOG_LINES preference (UNIX). For more information, see Appendix E, "IDL Preferences" in the *IDL Reference Guide* manual.

- **Number of log lines to delete at limit / Delete on Limit** — This field controls the number of lines that will be deleted from the Output Log window when the maximum number of lines is reached. The earliest lines in the log are deleted. The default is 100 for Microsoft Windows systems and 250 for UNIX systems.

  This control sets the value of the IDL_WDE_LOG_TRIM preference (Windows) and the IDL_MDE_LOG_TRIM preference (UNIX). For more information, see Appendix E, "IDL Preferences" in the *IDL Reference Guide* manual.

- **Number of lines saved in the recall buffer** — This field controls the maximum number of lines saved in the recall buffer. (See "Recalling Commands" in Chapter 2 of the *Building IDL Applications* manual for information on using the recall buffer.) The default is 20 lines.

  This control sets the value of the IDL_RBUF_SIZE preference. For more information, see Appendix E, "IDL Preferences" in the *IDL Reference Guide* manual.

- **Save Recall Buffer Between Sessions** — Select this option to have the recall buffer persist between IDL sessions.

  This control sets the value of the IDL_RBUF_PERSIST preference. For more information, see Appendix E, "IDL Preferences" in the *IDL Reference Guide* manual.

Files Section

You can change the way in which IDL handles opening files. Select or clear the following check boxes to apply or disable the options:

- **Change Directory on Open** — Select this option to cause IDL to change the current working directory when you open a file. The new current working directory will be the directory that contains the opened file.

  This control sets the value of the IDL_WDE_EDIT_CWD preference (Windows) and the IDL_MDE_EDIT_CWD preference (UNIX). For more information, see Appendix E, "IDL Preferences" in the *IDL Reference Guide* manual.

- **Open Files Read Only** — Select this option to open files so that they can be viewed, but not changed.

  This control sets the value of the IDL_WDE_EDIT_READONLY preference (Windows) and the IDL_MDE_EDIT_READONLY preference (UNIX). For more information, see Appendix E, "IDL Preferences" in the *IDL Reference Guide* manual.

Layout Preferences

This tab allows you to control the appearance and placement of the IDLDE.

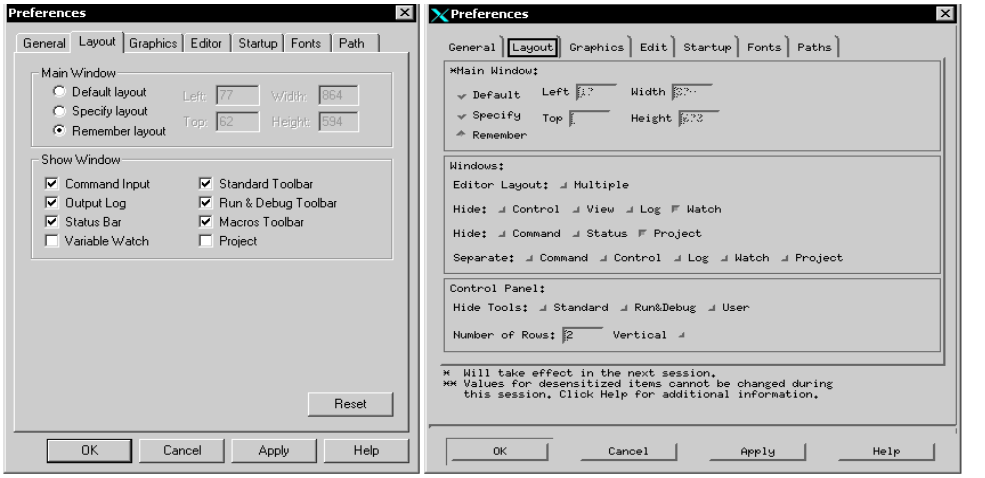


Figure 3-2: Layout Preferences Dialog

Note

Some preference settings may be desensitized. See "Unavailable Preferences" on page 94 for details.

Main Window Section

Use the fields in this section to specify the default size and placement of the IDL Development Environment's main window. (See "Components of the IDLDE" in Chapter 2 for descriptions of the components of the IDLDE.)

- Select the **Default Layout** radio button to use the IDLDE's default layout, which depends on the size and resolution of your computer screen. If you select this radio button, all of the IDLDE's windows and toolbars will be displayed in their standard locations.
- Select the **Specify Layout** radio button to manually specify the layout of the IDLDE:
  - Enter the number of pixels from the left-hand edge of the screen the IDLDE window should be displayed in the **Left** field

- Enter the number of pixels from the top edge of the screen the IDLDE window should be displayed in the **Top** field
- Enter the width of the IDLDE window in pixels in the **Width** field
- Enter the height of the IDLDE window in pixels in the **Height** field

Note that if you select the **Default Layout** radio button after specifying values in these fields, your values will be replaced with "-1" to indicate that the default values will be used the next time IDL starts.

- Select the windows and toolbars to be displayed from the **Show Window** section (Windows) or **Windows** and **Control Panel** sections (Motif)

Click **Apply** to apply your changes to the current IDLDE window without saving the values. (This allows you to use the **Layout** tab to control the appearance of the IDLDE for the current session without making your changes permanent.) Click **OK** to apply your changes and save the values; they will be used the next time IDL starts.

- Select the **Remember Layout** radio button and click **OK** to save the current layout of the IDLDE windows for use the next time IDL starts. This options is useful if you have configured the windows manually and wish to save your changes.

Undocking IDLDE windows

Some of the elements of the IDLDE can be "undocked" from the interface and appear as separate, free-floating windows. On Microsoft Windows systems, use the mouse to select an element and drag it away from the main IDLDE window to undock the element. On UNIX systems, you can use the checkboxes in the **Windows** section to undock elements. For more information, see "Docking/Undocking" in Chapter 2.

The following elements can be undocked:

- Command Line
- Toolbars
- Output Log
- Variable Watch Window
- Project Window

Show Window Section (Windows Only)

By default, all the listed options are checked, signifying that they are all visible in the IDLDE main window. Click on the check boxes to show or hide the following windows:

- Command line
- Output Log window
- Status Bar
- Variable Watch window
- Standard Toolbar
- Run & Debug Toolbar
- Macros Toolbar
- Project window

Click **Apply** to apply your changes to the current IDL session. (This is the same as selecting the corresponding options in the **Window** menu.)

Windows Section (Motif Only)

Use the options in this section to control the appearance of the window elements of the IDLDE.

- **Editor Layout** — Click **Multiple** to display open Editor and Project windows separately from the main IDLDE window. Note that if the **Multiple Windows** option is enabled, the choice to hide or view the Editor windows is not available.
- **Hide** — Select the check box for elements of the IDLDE you wish to hide from view. By default, none of the sections are hidden.
  - **Control** hides the toolbars;
  - **View** hides the Project window and the Editor window;
  - **Log** hides the Output Log window;
  - **Watch** hides the Variable Watch window;
  - **Command** hides the Input Command Line;
  - **Status** hides the fly over status line at the base of the Main IDL window;
  - **Project** hides the Project window and extends the Editor window to the full width of the IDLDE.

- **Separate** — Select the check box for the constituent window you want to separate from the IDLDE Main Window. When the **Separate** action is applied, the element is "undocked" from the interface and appears as separate, free-floating window.

Click **Apply** to apply your changes to the current IDL session. (This is the same as selecting the corresponding options in the Window menu.)

Control Panel Section (Motif Only)

You can specify how you would like to display the various toolbars on the Control Panel.

- **Hide Tools** — Select the check box for any of the available toolbars (**Standard**, **Run & Debug**, and **User)** to hide that toolbar.
- **Number of Rows** — Enter the number of rows to use in displaying any visible toolbars. You can select from 1 to 3 rows.
- **Vertical** — Select this check box to cause the toolbars to be stacked vertically one on top of the other rather than horizontally next to each other.

Graphics Preferences

This tab allows you to control the layout and default size of IDL graphics windows. You can also control IDL's default use of backing store and the size of the TrueType font cache. Note that the values set here are defaults; the values can be overridden when a graphics window is created.

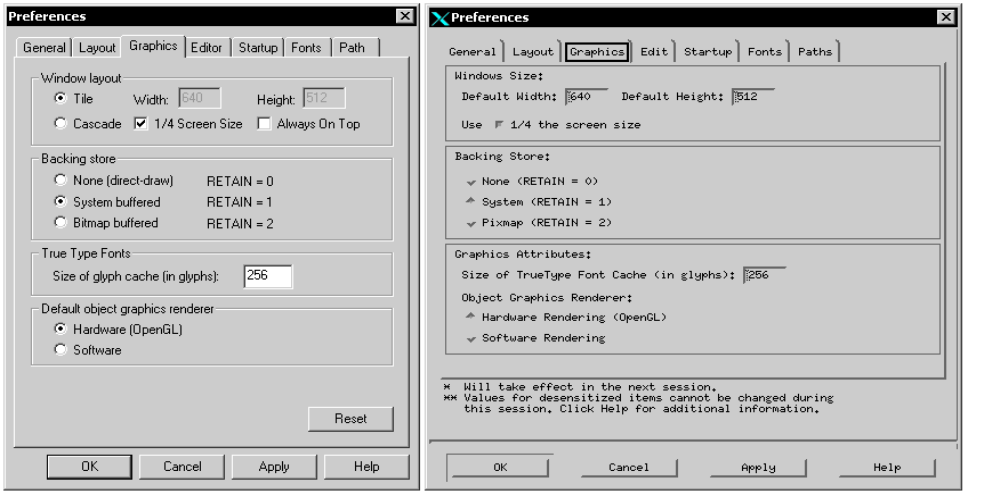


Figure 3-3: Graphics Preferences Dialog

Note

Some preference settings may be desensitized. See "Unavailable Preferences" on page 94 for details.

Window Layout / Windows Size Section

Specify the default width and height of IDL graphics windows in the **Width** and **Height** fields. These controls set the values of the IDL_GR_WIN_HEIGHT and IDL_GR_WIN_WIDTH preferences (Windows) and the IDL_GR_X_HEIGHT and IDL_GR_X_WIDTH preferences (UNIX). For more information, see Appendix E, "IDL Preferences" in the *IDL Reference Guide* manual.

Alternatively, you can specify that graphics windows have a default width and height of half the screen width and height by checking the **1/4 Screen Size** checkbox. This control sets the value of the IDL_GR_WIN_QSCREEN preference (Windows) and

the IDL_GR_X_QSCREEN preference (UNIX). For more information, see Appendix E, "IDL Preferences" in the *IDL Reference Guide* manual.

Platform Differences

On Windows systems, you can specify that graphics windows should be created side-by-side, with no overlap by selecting the **Tile** radio button, or that they should be created overlapping by selecting the **Cascade** radio button. This control sets the value of the IDL_GR_WIN_LAYOUT preference. For more information, see Appendix E, "IDL Preferences" in the *IDL Reference Guide* manual.

Select the **Always On Top** checkbox to ensure that graphics windows float above all other IDL windows. This control sets the value of the IDL_GR_WIN_ONTOP preference. For more information, see Appendix E, "IDL Preferences" in the *IDL Reference Guide* manual.

Backing Store Section

When backing store is enabled, a copy of each Graphics window is kept in memory; the copy is used to refresh the window when it has been covered and uncovered. IDL's performance may increase when no backing store is used, since the amount of memory required to save copies can affect the speed at which IDL will run. Settings in this section correspond to settings of the RETAIN keyword to the DEVICE procedure; see "Backing Store" in Appendix A of the *IDL Reference Guide* manual for more information.

- **None (**RETAIN = 0): Select this option to refrain from keeping a copy of the window. In some situations, disabling backing store may lead to an increase in IDL's performance.
- **System** (RETAIN = 1): Select this option to request backing store from the windowing system. This is the default.
- **Bitmap / Pixmap** (RETAIN = 2): Select this option to specify that IDL should maintain the backing store using its own memory.

Note

Backing Store preference changes do not take effect until the next IDL session.

This control sets the value of the IDL_GR_WIN_RETAIN preference (Windows) and the IDL_GR_X_RETAIN preference (UNIX). For more information, see Appendix E, "IDL Preferences" in the *IDL Reference Guide* manual.

True Type Fonts Section

Note

On UNIX systems, this preference is included in the Graphics Attributes section of the dialog, described below.

IDL saves TrueType fonts as a set of *glyphs*; each glyph represents the triangulation data for drawing one character. The **Size of TrueType Font Cache** (in glyphs) field allows you to set the number of glyphs to keep in cache memory; keeping glyphs in memory speeds drawing of fonts in IDL graphics windows. The default number of glyphs in cache memory is 256, roughly two TrueType font sets.

Enter the number of TrueType characters for which to save triangulation information. Saving the triangulation information for TrueType characters means that IDL will not have to calculate the polygons to draw the next time a character of the same font and size is rendered. Larger values will use more memory but can increase drawing speed if multiple fonts are used. The default is 256.

This control sets the value of the IDL_GR_TTCACHESIZE preference. For more information, see Appendix E, “IDL Preferences” in the *IDL Reference Guide* manual.

Default Object Graphics Renderer / Graphics Attributes Section

IDL supports two methods of rendering object graphics: via a hardware graphics accelerator or via a software rendering package. Select **Hardware** rendering if your system has OpenGL graphics accelerator hardware. Select **Software** rendering otherwise. This control sets the value of the IDL_GR_WIN_RENDERER preference (Windows) and the IDL_GR_X_RENDERER preference (UNIX). For more information, see Appendix E, “IDL Preferences” in the *IDL Reference Guide* manual.

See “Hardware vs. Software Rendering” in Chapter 12 of the *Object Programming* manual for information about the differences between the two rendering systems.

Editor Preferences

This tab allows you to specify settings for the built-in IDL Editor and control the way IDL compiles files loaded in editor windows. On Microsoft Windows systems, this tab also allows you to specify syntax-highlighting and other editor features.

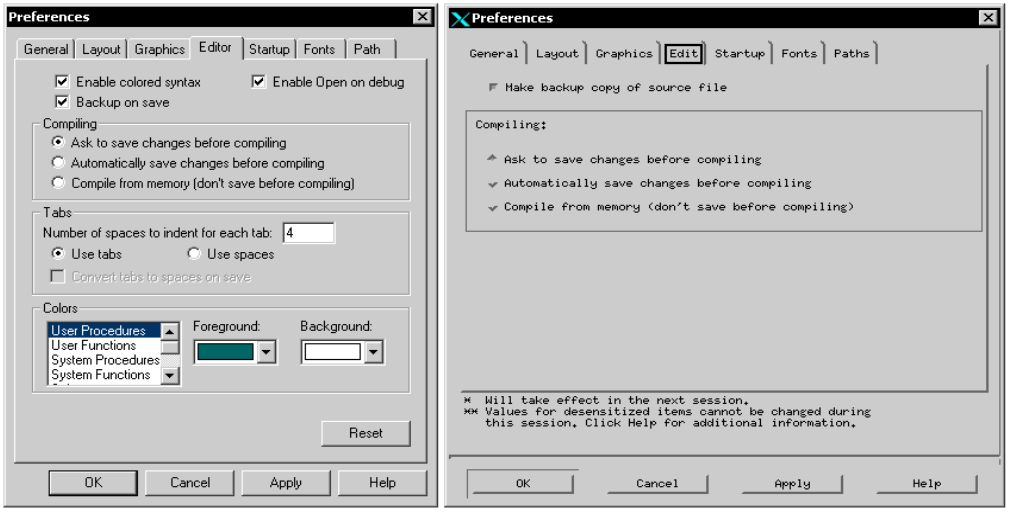


Figure 3-4: Editor Preferences Dialog

Note

Some preference settings may be desensitized. See "Unavailable Preferences" on page 94 for details.

Backup on Save

Select the **Backup on Save / Make backup copy of source file** check box to cause IDL to create a backup of the original file when saving a file in an IDL editor window.

This control sets the value of the IDL_WDE_EDIT_BACKUP preference (Windows) and the IDL_MDE_EDIT_BACKUP preference (UNIX). For more information, see Appendix E, "IDL Preferences" in the *IDL Reference Guide* manual.

Syntax Highlighting (Windows Only)

On Microsoft Windows systems, you can choose to use *syntax highlighting* in IDL editor windows. If syntax highlighting is turned on, IDL statements are displayed in different colors. Select the **Enable colored syntax** checkbox to enable syntax highlighting. This control sets the value of the IDL_WDE_EDIT_CHROMACODE preference. For more information, see Appendix E, "IDL Preferences" in the *IDL Reference Guide* manual.

Open on Debug (Windows Only)

If you want IDL to open the source file for a program that generates an error in an IDL editor window, select the **Enable Open on debug** checkbox. This control sets the value of the IDL_WDE_EDIT_OPEN_ON_DEBUG preference. For more information, see Appendix E, "IDL Preferences" in the *IDL Reference Guide* manual.

Compiling Section

Select the **Ask to save changes before compiling** radio button if you would like to save changes when you compile a program in an IDL editor window. This is the default.

Select the **Automatically save changes before compiling** radio button if you do not want to be prompted each time you compile, but do want to save the changes.

Select the **Compile from memory (don't save before compile)** radio button if you do not want to save files before compiling them.

Note

You can override your default selection by selecting the appropriate menu item from the **Run** menu.

This control sets the value of the IDL_WDE_EDIT_COMPILE_OPTION preference (Windows) and the IDL_MDE_EDIT_COMPILE_OPTION preference (UNIX). For more information, see Appendix E, "IDL Preferences" in the *IDL Reference Guide* manual.

Tabs Section (Windows Only)

You can specify the width of the white space to be used when you press the **Tab** key in an IDL editor window. Enter a number in the **Number of spaces to indent for each tab** field to specify the width of the indent to be used. This control sets the value of the IDL_WDE_EDIT_TAB_WIDTH preference. For more information, see Appendix E, "IDL Preferences" in the *IDL Reference Guide* manual.

If you want the IDL editor to insert a tab character (ASCII 9) when you press the **Tab** key, select the **Use tabs** radio button. If you want IDL to insert the specified number of space characters (ASCII 32) when you press the **Tab** key, select the **Use spaces** radio button. This control sets the value of the IDL_WDE_EDIT_TAB_ENABLE preference. For more information, see Appendix E, "IDL Preferences" in the *IDL Reference Guide* manual.

If you have selected the **Use spaces** radio button, you have the option to convert tab characters to spaces when the file is saved by selecting the **Convert tabs to spaces on save** checkbox. This control sets the value of the IDL_WDE_EDIT_TAB_SP_ON_SAVE preference. For more information, see Appendix E, "IDL Preferences" in the *IDL Reference Guide* manual.

Colors Section (Windows Only)

Use this section to select the colors that will be used in the IDL editor when syntax highlighting is enabled. To set colors, select a type of IDL statement from the scrolling listbox at left, then select the foreground and background colors for that type of statement.

These controls set the values of the IDL_WDE_EDIT_BCOLOR_* and IDL_WDE_EDIT_FCOLOR_* preferences. For more information, see "IDL_WDE_EDIT_[B|F]COLOR_*" in Appendix E of the *IDL Reference Guide* manual.

Startup Preferences

This tab allows you to specify the locations of the default working directory and any startup file to be run.

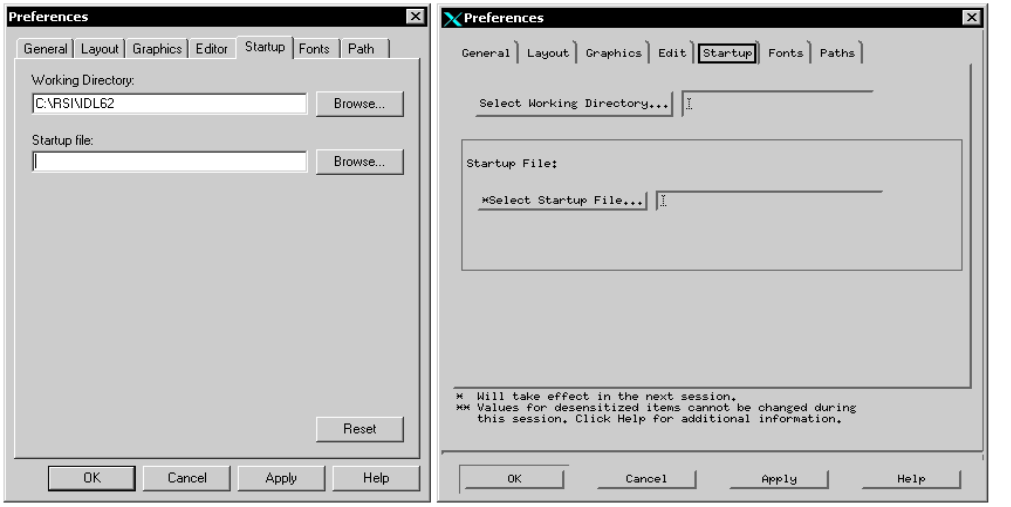


Figure 3-5: Startup Preferences Dialog

Note

Some preference settings may be desensitized. See “Unavailable Preferences” on page 94 for details.

Working Directory

This field allows you to set the initial working directory for future IDL sessions. The General Preferences tab contains a “Change Directory on Open” option, which also affects the working directory.

This control sets the value of the IDL_WDE_START_DIR preference (Windows) and the IDL_MDE_START_DIR preference (UNIX). For more information, see Appendix E, “IDL Preferences” in the *IDL Reference Guide* manual.

Startup File

Use this field to specify the name of an IDL batch file to be executed automatically each time IDL is run. See “Startup Files” on page 30 for additional details.

This control sets the value of the IDL_STARTUP preference. For more information, see Appendix E, “IDL Preferences” in the *IDL Reference Guide* manual.

Font Preferences

This tab allows you to specify fonts to be used in various sections of the IDLDE interface.

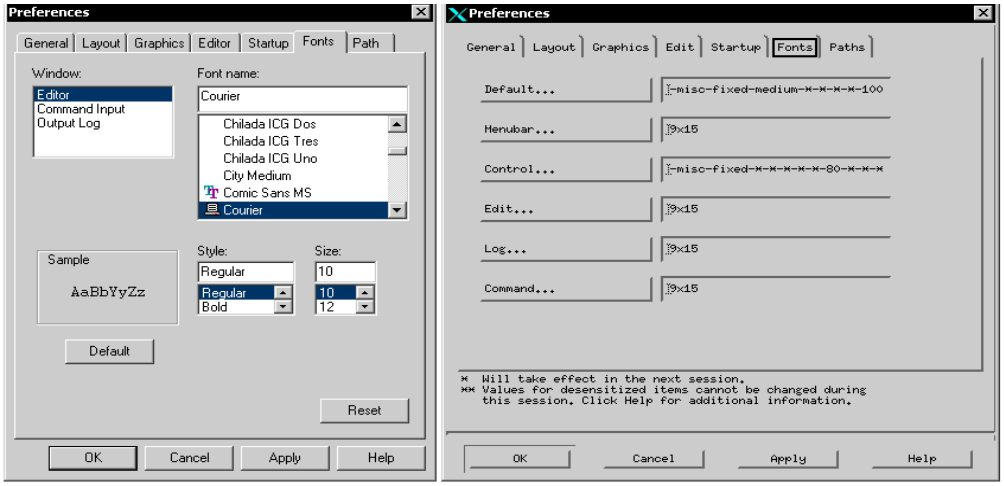


Figure 3-6: Font Preferences Dialog

Note

Some preference settings may be desensitized. See “Unavailable Preferences” on page 94 for details.

Microsoft Windows

Under Microsoft Windows, IDL uses a standard Windows font-selection dialog. You can select different fonts for IDL Editor windows, the Command Line, and the Output Log. Click on one of these areas in the **Window** list, then select the font, style, and size using the appropriate lists. Click **Use Default Fonts** to change to the IDL default font selections for all three areas.

These controls set the values of the IDL_WDE_EDIT_FONT, IDL_WDE_INPUT_FONT, and IDL_WDE_LOG_FONT preferences. For more information, see Appendix E, “IDL Preferences” in the *IDL Reference Guide* manual.

UNIX

This tab allows you to control which fonts are to be used for the main IDL window. Click on any of the following buttons to specify the relevant font:

- **Default** — dialog boxes
- **Menubar** — menu items
- **Control** — the Control Panel
- **Edit** — editor windows
- **Log** — the Output Log
- **Command** — the Command Line

Selecting a Font

Clicking any of the buttons on the **Fonts** tab of the Preferences dialog brings up the **Select Font** dialog. This dialog allows you to select fonts from the X Windows Server font database, based on the attributes *Foundry*, *Family*, *Weight*, *Slant*, *SetWidth*, and *Size*. Using this dialog is similar to using the `xfontsel` X Window

utility. See your X Window system font documentation for additional details. Once you have selected a font, click **OK** to accept your selection or **Cancel** to abandon it.

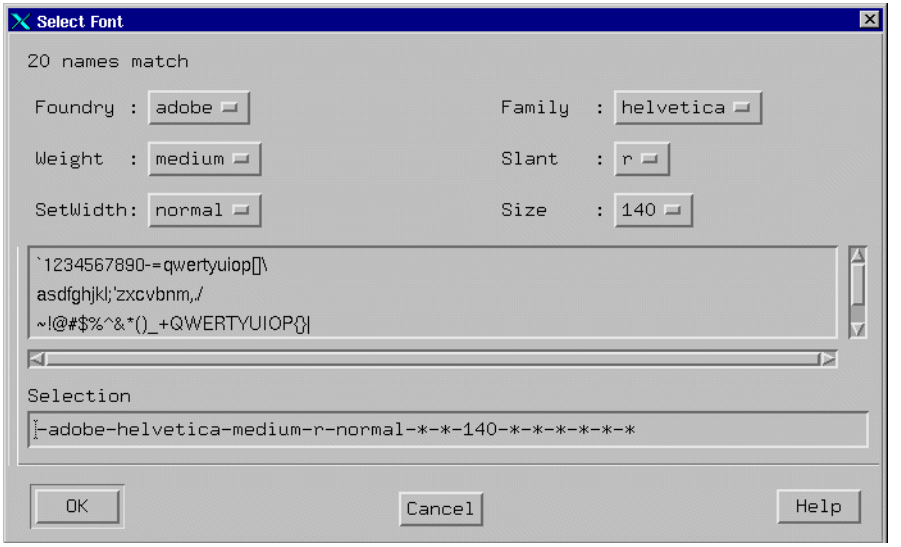


Figure 3-7: Motif Select Font Dialog.

Note

The UNIX IDLDE stores font preference information in the `~/.idlde` X Resource file. See Chapter 5, “Customizing IDL on Motif Systems” for details.

Path Preferences

This tab allows you to control where IDL looks for procedures and functions. The path elements specified in the **Search Path / IDL Files Search Path** are used to set the IDL_PATH preference and the !PATH system variable.

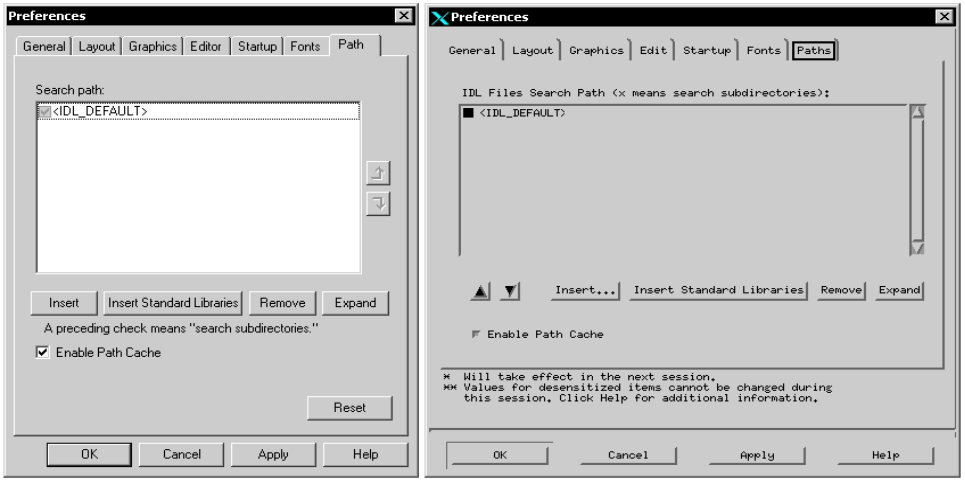


Figure 3-8: Path Preferences Dialog

Note

Some preference settings may be desensitized. See "Unavailable Preferences" on page 94 for details.

Search Path / IDL Files Search Path

The IDLDE Path Preferences dialog uses the same mechanism to expand the elements of the **Search Path** field as is used by the EXPAND_PATH function. By default, this field is populated with the current value of the IDL_PATH preference. For more information, see Appendix E, "IDL Preferences" in the *IDL Reference Guide* manual.

Note

If you have not changed the value of the IDL_PATH preference, it contains a single entry (`<IDL_DEFAULT>`) indicating that the default IDL path will be used. See "The

Path Definition String" under "EXPAND_PATH" in the *IDL Reference Guide* manual for complete details on how this token is expanded.

If the box to the left of a path element is checked, all directories below the listed directory that contain at least one `.pro` or `.sav` file will be included in !PATH. (This mechanism is analogous to the use of a "+" symbol in an EXPAND_PATH path definition string.)

Note

If the `<IDL_DEFAULT>` entry is present, the box to its left is both checked and greyed out (Windows) or completely blacked out (Motif), indicating that the token will always be expanded.

You can modify the value of the !PATH system variable in the following ways using this dialog:

- **Change the order of the path elements** — using the up- and down-arrows, you can reorder the path elements. When searching the directories in the !PATH system variable for files, IDL will use the first matching file it finds. If you have multiple files with the same name in different directories within !PATH, you may need to adjust the order in which the directories are scanned.
- **Insert...** — To add a path to the **Search Path** list, click **Insert...** to display the Select Directory dialog. The new path is inserted before the first selected path. If none of the paths are selected, the new path is appended to the end of the list.
- **Insert Standard Libraries** — Click **Insert Standard Libraries** to insert the `<IDL_DEFAULT>` path element into the list.
- **Remove** — Click on **Remove** to delete the selected path.
- **Expand** — Click on **Expand** to include the individual subdirectories of the selected path element in the **Search Path** list. When you click **Expand**, the checkmark is removed from the original path element, since the subdirectories are now explicitly included in the path search list.

See "Automatic Compilation" in Chapter 2 of the *Building IDL Applications* manual for more information on how !PATH is used by IDL when compiling and running programs.

Enable Path Cache

Select **Enable Path Cache** to enable IDL's path caching mechanism. Path caching is enabled by default, and in almost all cases should be left enabled. See

"PATH_CACHE" in the *IDL Reference Guide* manual for more information about IDL's path cache.

This control sets the value of the IDL_PATH_CACHE_DISABLE preference. For more information, see Appendix E, "IDL Preferences" in the *IDL Reference Guide* manual.

Chapter 4

Creating Development Environment Macros

This chapter discusses the following topics:

What Are Macros? 120
Creating UNIX Macros 121
Creating Windows Macros 124
Command Stream Substitutions 126
Building IDL Example Macros 127

What Are Macros?

A macro allows you to execute commonly-used IDL tasks with the press of a mouse button or through a single keystroke (“hot key”) combination. In IDL you can create your very own macros using the following items:

- routines
- procedures
- statements
- command stream substitutions

For example, you may customize and extend the functionality of the IDL Development Environment (such as writing a procedural macro to change IDL’s working directory, which we will see later in this section).

Predefined IDL Macros

IDL offers several existing macro options on its Macro Toolbar. These macros allow you quick access to commonly used IDL functionality such as printing a variable, importing various file types, and running the IDL Demos.

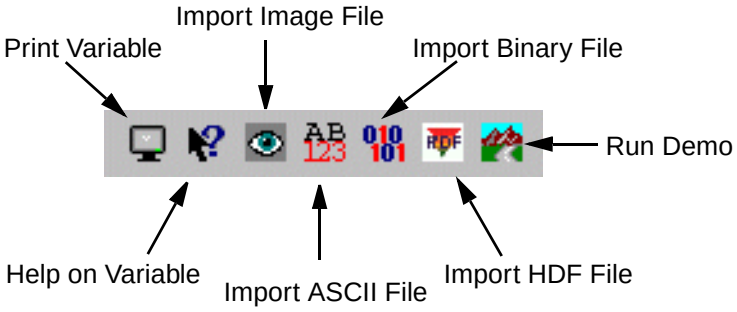


Figure 4-1: The IDLDE’s Macro Toolbar

See “Using IDL Macros” on page 164 for more information.

Creating UNIX Macros

You can modify the contents of the **Macros** menu and macros toolbar, either using the **E**dit Macros dialog (displayed by selecting **Edit...** from the **Macros** menu) or by manually editing the user resource (`.idlde`) file.

Using the Edit Macros Dialog

The Edit Macros dialog allows you to add, remove, or modify macros that appear either in the **Macros** menu or the **Macros** toolbar.

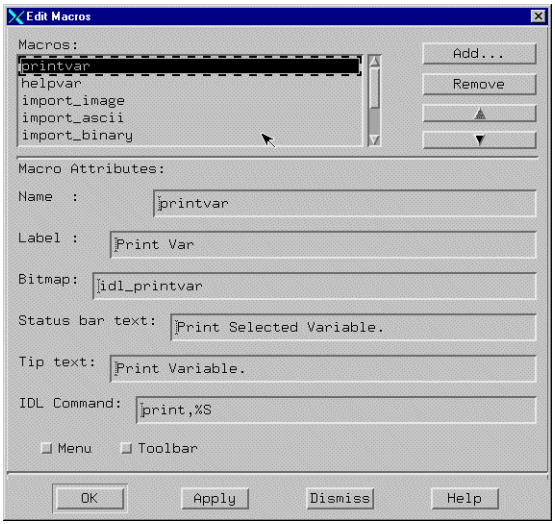


Figure 4-2: The Edit Macros Dialog.

To add a new macro, do the following:

1. Enter a name for your macro in the **Name** field. The Name appears only in the Edit Macros dialog.
2. Enter a label for your macro in the **Label** field. The label will be used in the **Macros** menu (if selected).
3. Enter the name of the bitmap (`.xbm` or `.xpm`) file associated with the macro in the **Bitmap** field. The bitmap will be used on the Macros toolbar (if selected). See “Bitmaps for Control Panel Buttons” on page 122 for details.

4. Enter text to be displayed on the IDLDE status bar in the **Status bar text** field.
5. Enter text to be displayed as a tooltip when the mouse cursor is positioned over the toolbar button in the **Tip text** field.
6. Enter the IDL command to be executed in the **IDL Command** field. See "Command Stream Substitutions" on page 126 for information on the types of dynamic information that can be included in the command.

   In addition to IDL-language commands, you can attach IDL Motif Action Routines to a macro. See "Action Routines" on page 141 for details.
7. Select the **Menu** checkbox if you want the macro to appear on the **Macros** menu.
8. Select the **Toolbar** checkbox if you want the macro to appear on the Macros toolbar.
9. Click **Add** to add the new macro, then click **OK**.

To Remove an existing macro, select it from the list and click **Remove**. To rearrange macros in the list, use the up- and down-arrow buttons.

Bitmaps for Control Panel Buttons

It is recommended that bitmaps for control panel buttons:

1. Be in either XBM (X11 bitmap file) or XPM (X11 system pixmap file) format, with the file extension `.xbm` or `.xpm`.
2. Supply the full path name to the bitmap file. Alternatively, if the bitmap is located in one of the following directories, you can supply only the base file name:
   - `$IDL_DIR/resource/X11/lib/app_defaults`
   - `$IDL_DIR/resource/X11/lib/app_defaults/bitmaps`
   - `$HOME`
   - `$HOME/bitmaps`

Note

The above directories show the default search path for a bitmap file if nothing other than the root file name is specified in the .idlde file.

Manually Editing the Resource File

Although there is little advantage in doing so, you can also modify the **Macros** menu or toolbar by manually editing either your own local IDL resource file or the system-wide resource file. For details, see "Modifying the Control Panel" on page 138.

Creating Windows Macros

You can modify the contents of the **Macros** menu and macros toolbar using the **Edit Macros** dialog (displayed by selecting **Edit...** from the **Macros** menu). The **Edit Macros** dialog allows you to add, remove, or modify macros that appear either in the **Macros** menu or the **Macros** toolbar.

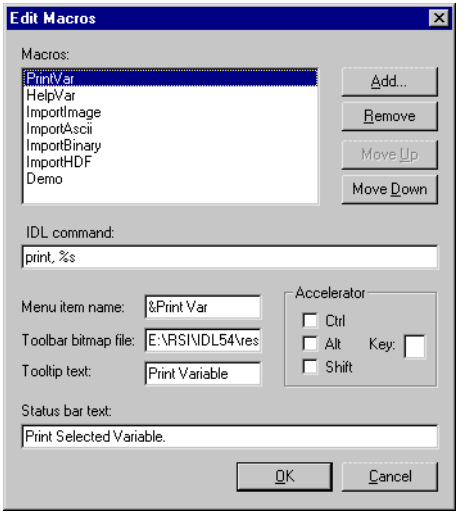


Figure 4-3: The Edit Macros Dialog.

To add a new macro, do the following:

1. Click **Add** and enter a name for your new macro. The name you specify appears only in the Edit Macros dialog.
2. Enter the IDL command to be executed in the **IDL Command** field. See "Command Stream Substitutions" on page 126 for information on the types of dynamic information that can be included in the command.
3. If you want your macro to be included in the **Macros** menu, enter a label for your macro in the **Menu Item Name** field.
4. If you want your macro to be included in the Macros toolbar, enter the full path name of the bitmap button file in the **Toolbar bitmap file** field. Bitmaps used as macro buttons in IDL must be 16 by 16 pixel `.bmp` files. IDL's default

bitmaps are stored in the `resources/bitmaps` subdirectory of the IDL distribution.

5. Enter text to be displayed as a tooltip when the mouse cursor is positioned over the toolbar button in the **Tooltip text** field. This value is ignored if no bitmap file is specified.
6. Enter text to be displayed on the IDLDE status bar in the **Status bar text** field.
7. Optionally, in the **Accelerator** field, enter a keystroke shortcut combination for your new macro. Note that you can create a macro that is available only by pressing the keystroke combination if you supply neither a label for the **Macros** menu nor a bitmap for the Macros toolbar.

To Remove an existing macro, select it from the list and click **Remove**. To rearrange macros in the list, use the up- and down-arrow buttons.

Click **OK** to accept your changes or **Cancel** to abandon them.

Command Stream Substitutions

You can use command stream (%) substitutions as shortcuts to incorporate certain types of information into the IDL command for your macro.

Command Stream Substitution	Result
%F	The filename associated with the currently active editor window.
%P	The full path filename associated with the currently active editor window.
%N	The base name of the filename without its path or suffix.
%B	The base name of the filename without its path, but with its suffix.
%S	The currently selected text.
%L	The line number with the current insertion point.
%%	Inserts the “%” character.

Table 4-1: Listing of Useful Command Stream Substitutions

Note

When creating a new macro, you may store the macro in the folder (directory) which IDL has already provided for the existing IDLDE macros. This folder exists in the `lib\macros` directory of your installation directory. If you wish to create a unique folder for the storage of only macros which you have created you may do so.

Building IDL Example Macros

Below are two examples that illustrate how a macro is created in IDL. The first example below is a UNIX-only example; the second example will work on either Microsoft Windows or UNIX.

Creating a Macro to Call a Text Editor in IDL for UNIX

On UNIX platforms, you can create a macro to open a file that is currently open in the IDL Editor in another editor, such as `emacs` or `vi`. Use the following procedure to create the macro:

1. Select **Macros** → **Edit** menu to bring up the Edit Macros dialog box. You can use this dialog to create, edit, or remove macros.
2. Complete the fields in the Edit Macros dialog:
   - **Name**: The name that you wish to appear in the **Macros** list in the Edit Macros dialog. For example, enter Edit in `emacs`.
   - **Label**: The name that you wish to appear on the **Macros** menu. For example, enter `emacs`.
   - **Bitmap**: The bitmap to use as the toolbar button label. Use the file paths and file name extensions discussed in "Bitmaps for Control Panel Buttons" in Chapter 4.
   - **Status bar text**: The text that appears in the status bar when the mouse is help over the menu item or toolbar button.
   - **Tip text**: The text for the tool tip that appears when the mouse is held over the toolbar button.
   - **IDL command**: The IDL command to execute when the macro is selected. To create a macro for editing in Emacs, enter the following:

     ```
     SPAWN, 'emacs +%L %P &'
     ```

   - Select the **Menu** and/or **Toolbar** checkbox to specify whether the macro will appear in the **Macros** menu and/or the toolbar.
3. Create the new macro by pressing the **Add** button. If you entered `emacs` in the **Label** field, a new "emacs" macro is added to the Macros list.
4. To add a macro for editing in vi, repeat the above steps, but enter the following in the "IDL command" field:

   ```
   SPAWN, 'xterm -e vi +%L %P &'
   ```

Note

The IDLDE always checks to determine whether the current file has been externally modified before using it. If a file was modified with an external editor, IDLDE notifies you, and asks you to reload the file before using it. You can also use the **Revert to Saved** option from the **File** menu to reload the file.

Creating a Macro to Change the Working Directory

The following macro will select and change your current working directory. The steps below describe the fields of the Macros dialog on a Microsoft Windows system, but the macro will work equally well on a UNIX system.

First we will create a `.pro` file in IDL which will display a platform-specific directory-selection dialog.

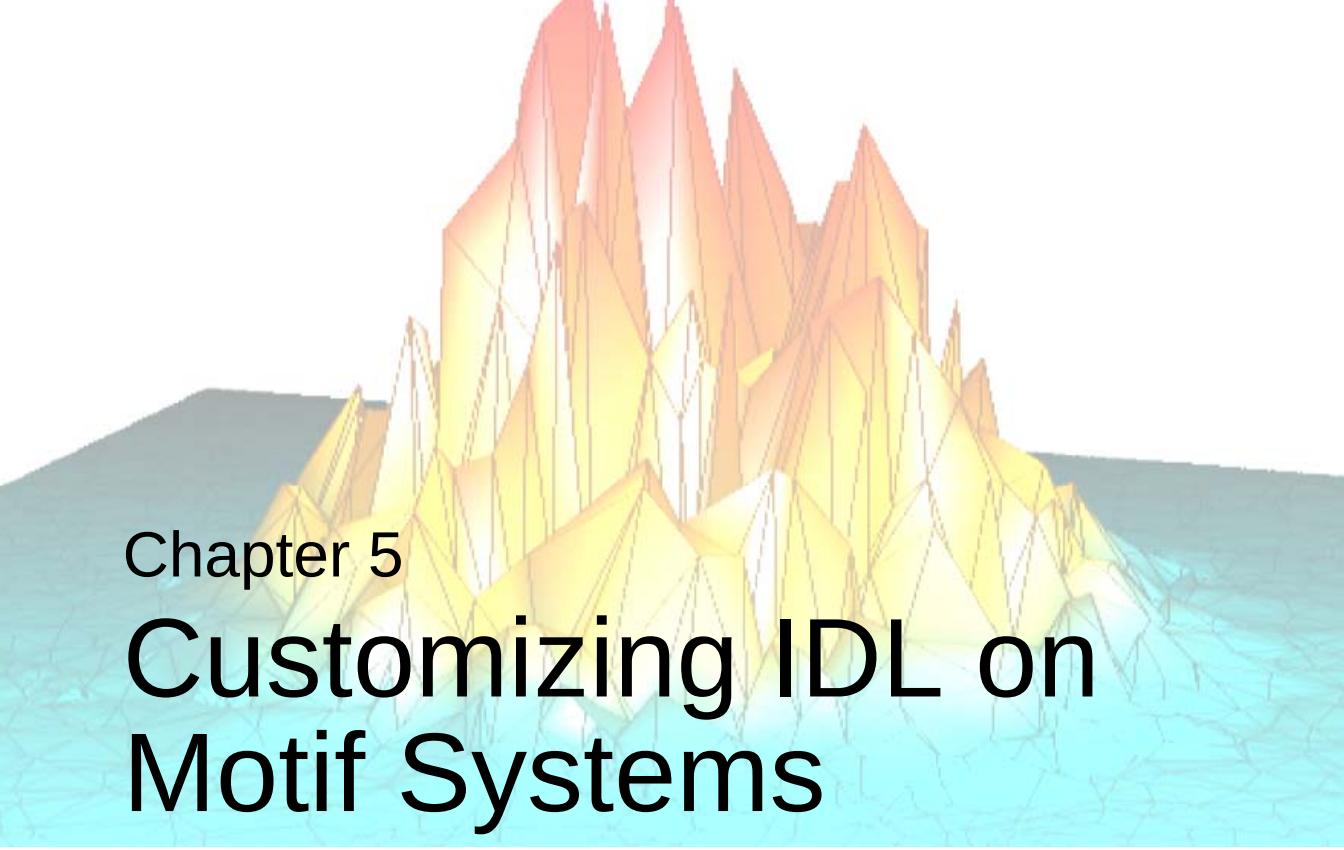
1. From the IDLDE, open a new IDL Editor window by selecting **File** → **New** → **Editor**.
2. Type (or copy) the following lines of code into the new Editor window to form a program:

```
PRO cd_test
dir = DIALOG_PICKFILE(/DIRECTORY)
  IF (dir) THEN BEGIN
    PRINT, 'Changing to: ', dir
    CD, dir
  ENDIF
END
```

3. Save the file as `cd_test.pro` in a directory included in IDL's path. (The file must be in IDL's path so that IDL will find it automatically when the command cd_test is executed by the macro we will create.)
4. Select **Macros** → **Edit** menu to bring up the Edit Macros dialog box.
5. Click **Add** to create a new macro. Enter "Change Directories" as the macro name.

6. Complete the following fields in the Edit Macros dialog:
   - Enter “cd_test” in the **IDL command** field.
   - Enter “Change Directories” in the **Menu item name** field.
   - Leave the **Toolbar bitmap file** field blank. This macro will appear only in the **Macros** menu.
   - Leave the **Tooltip text** field blank. This value is used only when a toolbar button is present.
   - Leave the **Status bar text** field blank. This value is used only when a toolbar button is present.

To use the new macro, select “Change Directories” from the **Macros** menu.



Chapter 5

Customizing IDL on Motif Systems

This chapter describes techniques for customizing versions of IDL running under the X Window System (Motif) graphical user interface.

Using X Resources to Customize IDL . . . 132
X Resources at the Command Line 136
Modifying the Control Panel 138
Action Routines 141

Using X Resources to Customize IDL

IDL on UNIX platforms respects the values of a number of X Window System (Motif) resources.

X Resources and IDL Preferences

Beginning with IDL 6.2, many values used to customize the appearance and behavior of IDL on UNIX platforms are stored in IDL *preferences* rather than in X resources. See Appendix E, "IDL Preferences" in the *IDL Reference Guide* manual for a detailed description of IDL's preferences system.

To provide backwards compatibility with older versions, current versions of IDL are still able to check the values of X resources set in the user's `.idlde` or `.Xdefaults` files and transfer them, if found, to the corresponding IDL preference setting. The mechanism used is described in detail in "Support for Obsolete Preference Mechanisms" in Appendix E of the *IDL Reference Guide* manual.

Not all X resources have corresponding preference values. Generally speaking, the X resource values that have *not* been implemented as preferences either control aspects of the appearance of the IDL Development Environment or define user macros. These values may become IDL preferences in a future version of IDL.

X Resources in Brief

The component widgets of an X Window System application each have two names, a class name that identifies its type (e.g., XmText for the Motif text widget) and an instance name (e.g., command, the name of the IDLDE command input text widget). The class name can be used to set resources for an entire class of widgets (e.g., to make all text widgets have a black background) while the instance name is used for control of individual widgets (e.g., set the IDLDE command input window font without affecting other widgets).

Applications consist of a tree of widgets, each having a class name and an instance name. To specify a resource for a given widget, list the names of the widgets lying between the top widget and the target widget from left to right, separated by periods. In a moderately complicated widget hierarchy, only some of the widgets are of interest; there are intervening widgets that serve uninteresting purposes (such as a base that holds other widgets). A star (*) character can be used as a wildcard to skip such widgets. Another fact to keep in mind is that a given resource specification is interpreted as broadly as possible to apply to any widget matching that description.

This allows a very small set of resource specifications to affect a large number of widgets.

Resource Files

There are two resource files used to customize the IDL Development Environment. An installation-wide resource file called `Idl` is located in

```
$IDL_DIR/resource/X11/lib/app-defaults
```

and a user resource file called `.idlde` is located in your home directory.

Modifying the global `Idl` resource file effects an installation-wide customization. Changes to the `Idl` file are not migrated when a new version of IDL is installed.

The user resource file, `.idlde`, customizes individual versions of IDLDE and is divided into two sections. The first section contains user-defined customization resources. You can place comments starting with "!" or "!!" in the first section of `.idlde`. When newer versions of `.idlde` are written, system comments are prefixed with "!!!". The second section of `.idlde` is used to store *IDLDE preferences*; it is modified when IDLDE preferences are modified via the **Preferences** tab of the Motif IDLDE, and should not be modified manually.

Note

IDLDE preferences saved in the `.idlde` file should not be confused with the IDL preference system included in IDL versions 6.2 and later. In some cases, values of the IDLDE preferences from the `.idlde` file are migrated to the newer preferences system; see "Support for Obsolete Preference Mechanisms" in Appendix E of the *IDL Reference Guide* manual for details.

If you use IDL in command-line mode rather than via the IDL Development Environment, you can include resources in the `.Xdefaults` file located in your home directory.

Format of IDL Resources

IDL resource strings begin with the characters "`Idl`". Most of these resources have been superseded by preferences in the IDL preference system.

Resource strings that apply only to the IDL Development Environment begin with the characters "`Idlde`" or "`idlde`". For example, the resource

```
idlde*hideCommand
```

controls whether the IDLDE Command Line is visible when IDL starts up.

Resources that include the string "idlde" must be included either in the system-wide Idl resource file, or in a .idlde file in your home directory. Resources that apply to IDL whether it is running in command-line mode or via the IDLDE can be included in either the system-wide Idl resource file or in a .Xdefaults file in your home directory.

To specify a value for an X resource, append a colon character and the value after the resource string. Whitespace is ignored. For example:

```
idlde*hideCommand:False
```

is the same as

```
idlde*hideCommand:        False
```

X Resources Used by IDL

IDL uses a large number of resources to control the behavior and appearance of the IDL Development Environment and any graphical application written in IDL. To learn more about the specific resources used, or to modify individual values, inspect the installation-wide resource file Idl, located in

```
$IDL_DIR/resource/X11/lib/app-defaults
```

Note

In order to maintain backward compatibility with previous versions of IDL, the Idl resource file contains values for some resources that correspond to IDL preferences. *X resources that have been superseded by preferences are ignored by IDL.* See "Support for Obsolete Preference Mechanisms" in Appendix E of the *IDL Reference Guide* manual for details.

Tip

RSI suggests that you use preferences rather than X resources when possible. If you must make changes to X resources, make the changes in a user-specific .idlde file or .Xdefaults file.

Reserving Colors

If you use a PseudoColor display device, when IDL starts, it attempts to secure entries in the shared system color map for use when drawing graphics. If the entry Idl.colors exists in one of the X resource files inspected by IDL at startup, IDL will first migrate the specified value to the value of the IDL_GR_X_COLORS preference, and then attempt to allocate the number of colors specified from the shared colormap. If for some reason it cannot allocate the requested number of colors

from the shared colormap, IDL will create a private colormap. Using a private colormap ensures that IDL has the number of colormap entries necessary, but can lead to colormap flashing when the cursor or window focus moves between IDL and other applications.

One way to avoid creating a private colormap for IDL is to set the IDL_GR_X_COLORS preference equal to a negative number. This causes IDL to try to use the shared colormap, allocating all but the specified number of colors. For example, setting the preference value to `-10` instructs IDL to allocate all but 10 of the currently available colors for its use. Thus, if there are a total of 220 colors not yet reserved by other applications (such as the windowing system), IDL will allocate 210 colors from the shared colormap.

The IDLDE application itself uses between 10-15 colors. On startup, the IDLDE will attempt to use colors in the shared colormap, but will reserve colors for itself if appropriate matching colors in the shared colormap are not found. As a result, running IDL with the IDLDE may use more colors than running IDL with the tty (plain command line) interface.

Note

If you use a TrueColor display device, IDL does not rely on the system's shared color map when drawing graphics. There is no need to either reserve colors from the shared color map or create a private color map.

X Resources at the Command Line

The appearance of the UNIX IDLDE can also be customized from the command line using the command line flags described below. Command line flags are given precedence over global resource files (`Idl`) and user resource files (`.idlde`). For more information about resources, see "Using X Resources to Customize IDL" on page 132.

X Resource Command Line Switches

The following command line switches can be used to control the values of X resources when invoking IDL on UNIX platforms. Unless otherwise noted, switches can be combined, and can be specified in any order.

-nocommand

Hides the Output Log window and Command Line at startup. The related resource is `Idl*idlde*hideCommand: True`.

-command

Displays Log window and Command Input window at startup. The related resource is `Idl*idlde*hideCommand: False`.

-nocontrol

Hides the Control panel buttons at startup. The related resource is `Idl*idlde*hideControl: True`.

-control

Displays the Control Panel buttons at startup. The related resource is `Idl*idlde*hideControl: False`.

-nolog

Hides the Output Log at startup. The related resource is `Idl*idlde*hideLog: True`.

-log

Displays the Output Log at startup. The related resource is `Idl*idlde*hideLog: False`.

-nostatus

Hides the Status Bar at startup. The related resource is `Idl*idlde*hideStatus: True`.

-status

Displays the Status Bar at startup. The related resource is `Idl*idlde*hideStatus: False`.

-single

Displays files in a single window, which is a child of the main IDLDE window. The related resource is `Idl*idlde*multiWindowEdit: False`.

-multi

Displays files in multiple windows, each one in a separate main level window. The related resource is `Idl*idlde*multiWindowEdit: True`.

-view

Displays the Multiple Document Panel in single window mode at startup. The related resource is `Idl*idlde*hideView: False`.

-noview

Hides the Multiple Document Panel at startup. The related resource is `Idl*idlde*hideView: True`.

-title "Title"

Use **Title** as the title of the main IDLDE window. The related resource is `idlde.title`.

Modifying the Control Panel

The **Control Panel**, with the resource name `control`, is located below the IDL Development Environment **Menu** bar. The Control Panel bar is a RowColumn widget containing buttons which serve as shortcuts for common commands.

You can modify the existing Control Panel settings by editing the `idlde*control` values in the system-wide `Idl` resource file or overriding those settings in your local `.idlde` file. In addition, you can add buttons to the **Macros** toolbar or menu by adding resources to your `.idlde` file.

Note

If you wish to add, modify, or remove the buttons on the **Macros** toolbar or menu, you can do so via the IDLDE interface using the **Edit Macros** dialog. See "Creating UNIX Macros" on page 121 for details. Whether you modify your macros using the dialog or by editing a resource file manually, the results are the same. There is little advantage to adding macros to the `.idlde` file manually.

Adding Macros Toolbar Buttons

The `idlButtonsUser` resource defines the *resource name* for each button on the **Macros** toolbar in the Control Panel. The resource name details button attributes, such as its label or pixmap, its associated IDL command, and its status bar message.

To add a button to the **Macros** toolbar, make the following modifications to the `.idlde` file:

- Add a new name to the `idlde*control*idlButtonsUser` list. The buttons are created in the order specified.
- Add `idlde*control*<new button>*labelString` or `labelPixmap` resources (or both). These resources define the button text or image. If you choose to use a pixmap label, be sure the file you specify abides by the restrictions described in "Bitmaps for Control Panel Buttons" on page 122.
- Add an `idlde*control*<new button>*idlCommand` resource. This is the text of the IDL command to execute. You can also include command stream substitutions; see "Command Stream Substitutions" on page 126 for details.

  Alternatively, you can add an `idlAction` resource. See "Action Routines" on page 141 for details.

- Add an `idlde*control*<new button>*hint` resource. This is the text that appears in the Status Bar when the cursor is positioned over the new button.
- Add an `idlde*control*<new button>*tip` resource. This is the text that appears as a “tooltip” when the cursor is positioned over the new button.

If you want your changes to be available to all users on the system, you can also modify the system-wide `Idl` resource file, located in the following directory:

```
$IDL_DIR/resource/X11/lib/app-defaults
```

Adding Macros Menu Entries

To add entries into the **Macros** menu, follow the same steps outlined above, modifying the `idlde*menubar*macrosMenu*macrosListUser` resource and substituting `idlde*menubar*macrosMenu*<new menu item>` for `idlde*control*<new button>` in the above steps.

Examples

To add a button called **Reset All** to the **Control Panel** with a color pixmap stored in the file `resetall.xpm` located in your home directory, add the following resources to the `.idlde` file in your $HOME directory:

```
idlde*control*idlButtonsUser: <exiting buttons> resetall
idlde*control*resetall*labelPixmap: resetall.xpm
idlde*control*resetall*labelString: Reset All
idlde*control*resetall*idlCommand:\
RETALL & WIDGET_CONTROL,/RESET
idlde*control*resetall*statusString:\
Stop execution of the current code and return to\
the main programming level
```

Note that in this example the new button is added at the end of the list of existing buttons. You can locate the new button anywhere in the list.

To specify a pixmap located in particular directory, specify the full file path of the pixmap file, for example:

```
idlde*control*resetall*labelPixmap:\
/home/user/bitmaps/resetall.xpm
```

To create two rows of the Control Panel from the default of one row, set the numColumns resource to 2:

```
idlde*control*numColumns: 2
```

To use label (text) buttons in the Control Panel set `labelType` to `XmSTRING`. To use icon (graphics) buttons set `labelType` to `XmPIXMAP`.

```
idlde*control*labelType: XmSTRING
or
idlde*control*labelType: XmPIXMAP
```

Action Routines

Most Motif widgets supply action routines which can be bound to events (such as keypress events). Action routines provided by IDL can be used to define commands for Control Panel buttons or menu items by using the `idlAction` resource.

The following action routines can be used in the same manner as the IDL commands specified in an `idlCommand` resource. The syntax to add an action routine to a control panel button is:

```
Idl*idlde*control*buttonName*idlAction: Action
```

or

```
Idl*idlde*control*buttonName*idlAction: Action(Arguments)
```

where *buttonName* is the name of the button and *Action* is the name of the action routine. Arguments to the action routine, if require, are enclosed in parentheses.

IdlBreakpoint

Use `IdlBreakpoint` to control the placement of breakpoints. If no parameter is specified, the breakpoint is set on the current line. At least one of the arguments from the following table must be set:

Argument	Action
SET	Set a breakpoint on the current line.
CLEAR	Clear the breakpoint on the current line.
TOGGLE	Toggle (SET or CLEAR) the state of the breakpoint on the current line.
COMPLEX	Display breakpoint dialog to set a complex breakpoint.
LIST	List all currently set breakpoints

Table 5-1: Breakpoint Arguments

For example, to use this action routine to clear a breakpoint, the *Action* specified would be:

```
IdlBreakpoint(CLEAR)
```

IdlClearLog

Use `IdlClearLog` to erase the contents of the Output Log.

IdlClearView

Use `IdlClearView` to clear the contents of the currently-active file in the Multiple Document Panel.

IdlCommandHide

Use `IdlCommandHide` to hide or expose the Command Area, which includes the Command Line and the Output Log. One of the following arguments must be set: **Show**, **Hide**, or **Toggle**.

IdlCompile

Use `IdlCompile` to compile the file in the currently-active editor window. One of the arguments from the following table must be set:

Argument	Action
FILE	Compiles the currently-active file.
TEMPORARY	Compiles the currently-active file into a temporary file
RESOLVE	Resolves all referenced and uncompiled IDL routines

Table 5-2: Compiling Arguments

IdlControlHide

Use `IdlControlHide` to hide or expose the Control Panel. One of the following arguments must be set: **Show**, **Hide**, or **Toggle**.

IdlEdit

Use `IdlEdit` to manipulate the contents of the currently-selected editor window. One of the arguments from the following table must be set:

Argument	Action
UNDO	Undo previous editing action.
REDO	Redo previously undone action.
CUT	Remove currently-selected text to UNIX clipboard.
COPY	Copy currently-selected text to UNIX clipboard.
PASTE	Paste contents of UNIX clipboard at current insertion point.
SELECTALL	Select all of the text in the currently-selected editor window.
GOTODEF	Display the definition of the currently-selected procedure or function.
GOTOLINE	Move directly to the specified line number.

Table 5-3: Editor Window Editing Arguments

IdlEditMacros

Use `IdlEditMacros` to display the **Edit Macros** dialog.

IdlExit

Use `IdlExit` to cause IDLDE to act as though the EXIT command has been entered. Note that this is usually tied to a menu accelerator (**Ctrl-Q** in this case), so this routine is rarely called directly.

IdlFile

Use `IdlFile` to manipulate the currently-selected editor window. One of the arguments in the following table must be set:

Argument	Action
NEW	Creates a new editor window.
OPEN	Opens an existing file.
SAVE	Saves the contents of the currently-selected editor window.
PRINT	Prints the contents of the currently-selected editor window.

Table 5-4: Editor Window Arguments

IdlFileReadOnly

Use `IdlFileReadOnly` to specify the read/write status of the currently-active editor window. One of the arguments from the following table must be set:

Argument	Action
READONLY	Disable editing of the currently-selected editor window.
READWRITE	Enables editing of the currently-selected window.

Table 5-5: Read/Write Arguments

IdlFunctionKey

Use `IdlFunctionKey` to allow entry of an IDL command into the input command stream. It is typically used to tie IDL commands to function keys. For example:

```
<Key>F5:IdlFunctionKey("print, 'F5 pressed'")\n
```

IdlInterrupt

Use `IdlInterrupt` to cause IDLDE to receive an interrupt. Note that this is usually tied to Ctrl-C as a menu accelerator.

IdlListStack

Use `IdlListStack` to display the current nesting of procedures and functions (calling stack).

IdlLogHide

Use `IdlLogHide` to hide or expose the Output Log. One of the following arguments must be set: **Show**, **Hide**, or **Toggle**.

IdlRecallCommand

Use `IdlRecallCommand` to recalls previously entered commands into the command widget. Either the **BACK** or the **FORWARD** argument must be specified to indicate the direction of the recall. For example:

```
<Key>osfUp:IdlRecallCommand(BACK)\n
```

IdlReset

Use `IdlReset` to reset the IDL environment.

IdlRun

Use `IdlRun` to execute the currently-active file.

IdlSearch

Use `IdlSearch` to call the **Find** dialog for a search of the current **Multiple Document Panel**. One of the optional arguments from the following table may be used:

Argument	Action
FIND	Displays a search dialog (default).
FINDAGAIN	Finds the next occurrence of the specified string.

Table 5-6: Find Dialog Arguments

Argument	Action
FINDSELECTION	Finds next occurrence of the current selection.
ENTERSELECTION	Enters the current selection as the search string in the Find dialog.
REPLACE	Replaces the search string, with a specified replacement string.
REPLACEFIND	Finds the next occurrence of the search string, and replaces it with the specified replacement string.

Table 5-6: Find Dialog Arguments

IdlStatusHide

Use `IdlStatusHide` to hide or expose the Status Bar. One of the following arguments must be set: **Show**, **Hide**, or **Toggle**.

IdlStep

Use `IdlStep` to control statement execution for debugging. At least one of the arguments from the following table must be set.

Argument	Action
INTO	Executes a single statement in the current program. If nested procedures or functions are encountered, they are also executed in single-statement mode.
OVER	Executes a single statement in the current program. If nested procedures or functions are encountered, they are run until completion, whereupon interactive control returns.
OUT	Continues execution until current routine returns.

Table 5-7: Debugging Arguments

Argument	Action
SKIP	Skips one statement and executes following statement.
CONTINUE	Continues execution of an interrupted program.
TOCURSOR	Executes file until encountering the cursor.
TORETURN	Executes file until encountering the return.

Table 5-7: Debugging Arguments

IdlTrace

Use `IdlTrace` to display a dialog box to control program tracing.

IdlViewHide

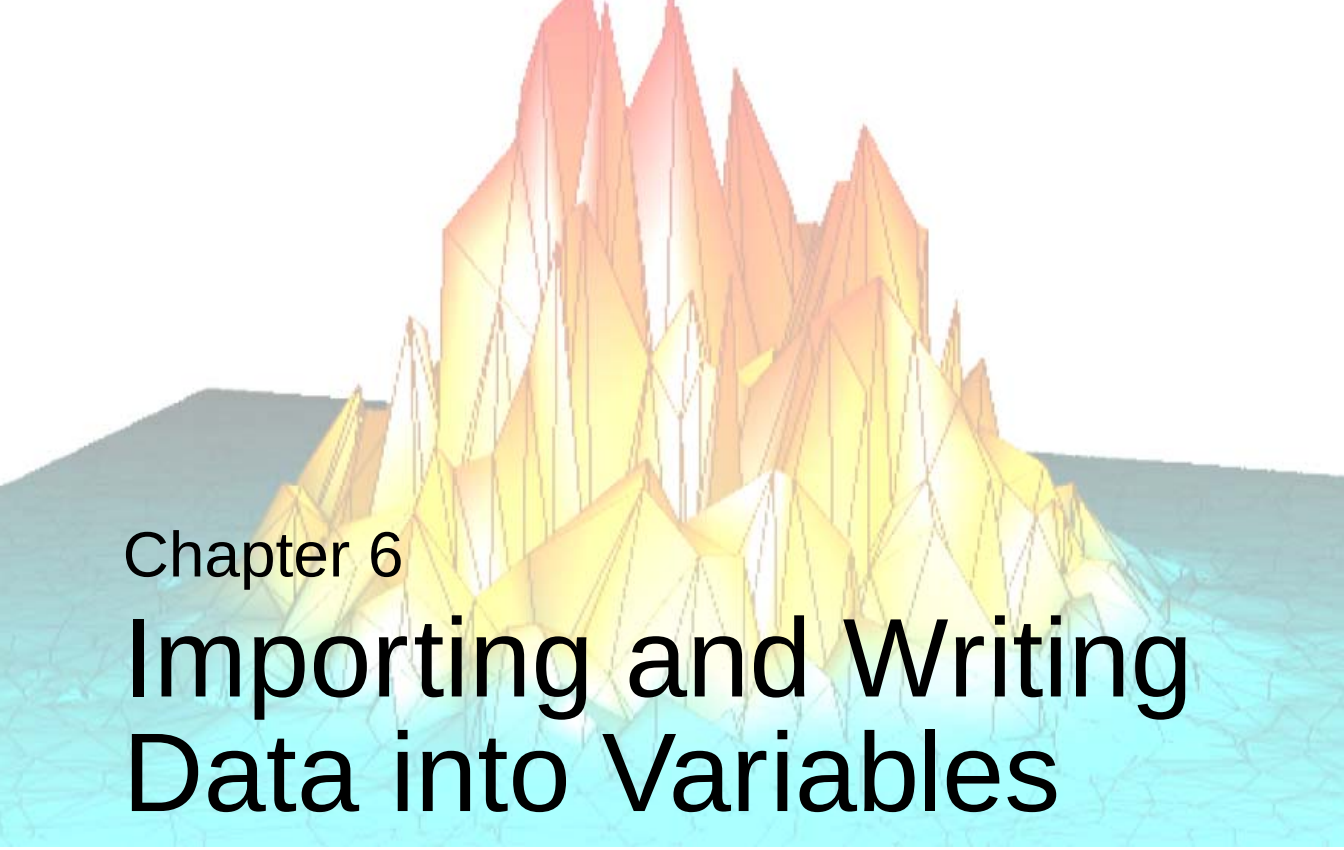
Use `IdlViewHide` to hide or expose the **Multiple Document Panel**. One of the following arguments must be set: **Show**, **Hide**, or **Toggle**.

IdlWindows

Use `IdlWindows` to manipulate the state of the Editor windows. One of the arguments from the following table must be set:

Argument	Action
CASCADE	Arrange open windows in a staggered, overlapping fashion.
TILE	Arrange all windows in a non-overlapping fashion.
MULTI	Open windows outside the IDLDE interface.
SINGLE	Display the most recent window on the Multiple Document Panel.

Table 5-8: Editor Window Display Arguments



Chapter 6

Importing and Writing Data into Variables

This chapter provides an introduction to accessing, reading and writing data using the dialogs, and routines found in IDL.

Overview of Data Access in IDL 150
Accessing Files Using Dialogs 151
Reading ASCII Data 153
Reading Binary Data 154
Accessing Files Programmatically 156
Accessing Image Data Programmatically 158
Accessing Non-Image Data Programmatically 162
Using IDL Macros 164
File Access Routines 171

Overview of Data Access in IDL

There are several ways to open files and access the data that they contain in IDL.You can open a file using interface elements, or using routines. In order of increasing complexity and flexibility, your options are:

- **Accessing data in iTools** — use **File → Open** from an iTool, and browse to select a file. This option automatically displays data (that is a supported type) in the iTool. See Chapter 2, "Importing and Exporting Data" in the *iTool User's Guide* manual for details.
- **Accessing files using dialogs** — launch an IDL dialog and browse to select or save a file. After accessing the file, use an IDL routine to access the data within the file. You can then preform additional data processing task or create a display. See "Accessing Files Using Dialogs" on page 151 for details.
- **Accessing files programmatically** — you can access data without requiring user interaction by using IDL statements in a program or at the command line. This give you the greatest control over the state of data at all times, but requires slightly more programming than the previous option. See "Accessing Files Programmatically" on page 156 for details.

There are advantages and disadvantages for each option. When you open a file using **File → Open** in the iTools, there is no opportunity to do pre-processing on the data. However, the display is created for you, and there are numerous interactive operations available.

You can combine the flexibility of accessing data using routines with the power of an iTool display by launching the iTool from the command line as described in "Parameter Data and the Command Line" in Chapter 2 of the *iTool User's Guide* manual. See "Accessing Image Data Programmatically" on page 158 and "Accessing Non-Image Data Programmatically" on page 162 for examples.

When you access data from the command line or in an IDL program, you have the greatest control over data modification. The iTools incorporate the functionality of many of the common data processing and manipulation routines. However, if you need greater control over data modification, want to create a custom display or object class, or need to use functionality that is not exposed through and iTool, you can import, export, and/or create your data programmatically.

Regardless of the method selected, it is important to note that only the options involving iTools will automatically display data for you. In other instances, you will need to configure a display yourself.

Accessing Files Using Dialogs

DIALOG_PICKFILE and DIALOG_READ_IMAGE are the two primary file access dialogs in IDL. Use DIALOG_PICKFILE to select any type of file. You can select multiple files, define the directory or define file filters using keywords. Use DIALOG_READ_IMAGE to access supported image formats (listed in "Image File Formats" in Chapter 1 of the *Using IDL* manual). This dialog offers preview capabilities and basic image information. The corollary DIALOG_WRITE_IMAGE allows you to write data to a select image file type.

See the following topics for more information:

- "Accessing Any File Type Using a Dialog" below
- "Importing an Image File Using a Dialog" on page 152
- "Saving an Image File Using a Dialog" on page 152

You can use other dialogs to access ASCII, binary and HDF data as described in:

- "Reading ASCII Data" on page 153
- "Reading Binary Data" on page 154

Also, several pre-defined IDL macros are provided to help you import data into the IDLDE. Each returns a structure, which you access programmatically in order to retrieve data. See "Using IDL Macros" on page 164 for details.

Note

Also see "CW_FILESEL" in the *IDL Reference Guide* manual for an example that configures a compound widget to open image files.

Accessing Any File Type Using a Dialog

The DIALOG_PICKFILE function lets you interactively pick a file using the platform's own native graphical file selection dialog. This function returns a string or an array of strings that contain the full path name of the selected file or files. The user can also enter the name of the file. The following statement opens the selection dialog and shows any `.pro` files in the current working directory. If you select a file and click **Open**, the `file` variable contains the full file path.

```
file = DIALOG_PICKFILE(/READ, FILTER = '*.pro')
```

Other keywords allow you to specify the initial directory, the dialog title, the filter list, and whether multiple file selection is permitted. See "DIALOG_PICKFILE" in the *IDL Reference Guide* manual for details.

After you select a file using DIALOG_PICKFILE, you can then use one of the file I/O routines to access the data within the file. See "Accessing Image Data Programmatically" on page 158 or "Accessing Non-Image Data Programmatically" on page 162 for more information.

Importing an Image File Using a Dialog

The DIALOG_READ_IMAGE function opens a graphical user interface which lets you read image files. This interface simplifies the use of IDL image file I/O. You can preview images with a quick and simple browsing mechanism which also reports important information about the image file. You can also control the preview mode.

The following statement opens the dialog so that you can select among `.gif`, `tiff`, `.dcm`, `.png` and `.jpg` files.

```
result = DIALOG_READ_IMAGE(FILE=selectedFile, IMAGE=image)
```

See "Using the Select Image File Dialog Interface" under "DIALOG_READ_IMAGE" in the *IDL Reference Guide* manual for additional information if desired. When you select a file and click **Open**, the file path is stored in `selectedFile` variable and the image data is stored in the `image` variable. Enter the following line to display image data in an iImage display.

```
IF result EQ 1 THEN iImage, image
```

Saving an Image File Using a Dialog

The DIALOG_WRITE_IMAGE function displays a graphical user interface that lets you write and save image files. This interface simplifies the use of IDL image file I/O. The following statements create and write a simple image to a `.tif` file name `myimage.tif`:

```
myimage = DIST(100)
result = DIALOG_WRITE_IMAGE(myimage, FILENAME='myimage.tif')
```

When you select **Save**, it creates a `.tif` file in your current working directory or the directory of your choice. See "DIALOG_WRITE_IMAGE" in the *IDL Reference Guide* manual for a complete list of keywords and a description of the dialog interface.

Reading ASCII Data

IDL recognizes two types of ASCII data files: free format files, and explicit format files. A free format file uses commas or tabs and spaces to distinguish each element in the file. An explicit format file distinguishes elements according to the commands specified in a format statement. Most ASCII files are free format files.

Note

If you prefer not to use an interactive dialog (described below), you can also use the READ/READF, or READS procedures to access ASCII data. The READ procedure reads free format data from standard input, READF reads free format data from a file, and READS reads free format data from a string variable.

Launching the ASCII Template Dialog

The ASCII_TEMPLATE function launches a dialog that you can use to configure the structure of data in an ASCII file. Access this feature in one of the following ways:

- From an iTool — select **File → Open** (or click the **Import File** button in the Data Manager or Insert Visualization dialog) and select a text file
- From the IDLDE — select **Macros → Import ASCII** and select a text file
- From the IDL command line — use the following syntax to call ASCII_TEMPLATE and select a text file:

```
sTemplate = ASCII_TEMPLATE()
```

Note

If you specify a *Filename* argument to ASCII_TEMPLATE, the dialog allowing you to browse to select a file will not appear. See "ASCII_TEMPLATE" in the *IDL Reference Guide* manual if you want specify a file and other parameters programmatically.

See "Using the ASCII Template Dialog" under "ASCII_TEMPLATE" in the *IDL Reference Guide* manual for instructions on how to use the dialog to define the structure of your ASCII data.

Reading Binary Data

Data is sometimes stored in files as arrays of bytes instead of a known format like JPEG or TIFF. These files are referred to as binary files. Binary data or binary data files are more compact than ASCII data files and are frequently used for large data files. Binary data files are stored as one long stream of bytes in a file. You will need to define the structure of the fields in the file in order to correctly read in the binary data.

The BINARY_TEMPLATE and READ_BINARY functions are designed to define and access binary data. The READ_BINARY function, which reads binary data, is either invoked internally (when you open a binary file from the iTools or use the **Import Binary** macro), or is explicitly called from the command line. This function is intended to read raw binary data that requires no special processing (except possibly byte-order swapping). This function is not designed to read commercial spreadsheet or word processing files.

Note

If you prefer not to use an interactive **Binary Template** dialog (described below) to define the structure of the data in the binary file, you can use the READU procedure. To read binary data files, define the variables, open the file for reading, and read the bytes into those variables. Each variable reads as many bytes out of the file as required by the specified data type and organizational structure.

If you need to open a single binary file, it may be easier to use READ_BINARY to directly define data characteristics using keywords instead of creating a template using the **Binary Template** dialog (described below). See "READ_BINARY" in the *IDL Reference Guide* manual for an example.

Launching the Binary Template Dialog

The BINARY_TEMPLATE function launches a dialog that you can use to define the structure of data in an binary file. Access this feature in one of the following ways:

- From an iTool — select **File** → **Open** (or click the **Import File** button in the Data Manager or Insert Visualization dialog) and select a binary file
- From the IDLDE — select **Macros** → **Import Binary** and select a binary file
- From the IDL command line — use the following syntax to call BINARY_TEMPLATE and select a text file:

```
sTemplate = BINARY_TEMPLATE()
```

Note

If you specify a *Filename* argument to BINARY_TEMPLATE, the dialog allowing you to browse to select a file will not appear. See "BINARY_TEMPLATE" in the *IDL Reference Guide* manual if you want specify a file and other parameters programmatically.

See "Using the BINARY_TEMPLATE Interface" under "BINARY_TEMPLATE" in the *IDL Reference Guide* manual for instructions on how to use the dialog to define the structure of your binary file.

Accessing Files Programmatically

Regardless of the data type, there are several routines that are commonly used to access files and data. To read data into an IDL variable, you must identify the file containing the data, and then extract the data from the file. This section discusses file access. Following sections (discuss data access.

File Access

One of the most common file access routines is FILEPATH. Use this to select a named file in a specified directory. For example, to select a file in the `examples/data` directory of the existing working directory, use the statement:

```
file = FILEPATH('mr_brain.dcm', SUBDIRECTORY=['examples', 'data'])
```

To access a file outside the existing working directory, use the ROOT_DIR keyword. The following statement opens a file named `testImg.tif` in the `C:\tempImages` directory.

```
file = FILEPATH('testImg.tif', ROOT_DIR='C:', $
   SUBDIRECTORY='tempImages')
```

Cross-platform File Access

If your application requires a cross-platform path, one that is not specific to UNIX or Windows, consider using the DIALOG_PICKFILE routine with the GET_PATH keyword. This lets you choose a file and store the operating system native path to the file in a variable. In the following example, you choose an image file and the full directory path to the selected image is stored in `path`:

```
sFile = DIALOG_PICKFILE(/MUST_EXIST, $
   TITLE = 'Select an Image File', $
   FILTER = ['*.bmp', '*.jpg', '*.png', '*.ppm', '*.tif'], $
   GET_PATH=path)
```

When you need to access a file in the directory stored in `path`, you can use the PATH_SEP function to return the correct path separation character for the operating system. Suppose you have a file called `myTestFile.jpg` that you want to delete before a program ends. FILE_DELETE requires a string *File* argument that is in the native syntax for the current operating system. To delete this file, you can use the directory information stored in `path`, plus the PATH_SEP function, plus the name of the file to delete as follows (the + operator concatenates strings):

```
FILE_DELETE, path+PATH_SEP()+'myTestFile.jpg', /ALLOW_NONEXISTENT
```

IDL also provides an extensive number of other file manipulation routines. See "General File Access" under the functional category "Input/Output" in the *IDL Quick Reference* manual for a list.

FILEPATH is often used in conjunction with routines that access the data from a file, as shown in the following section.

Accessing Image Data Programmatically

You can access image data using routines designed for general image file access, designed specifically for an image file format, or using unformatted data access routines. Which option you choose depends on the file type and the level of control you want over reading and writing the file. See the following topics for details:

- "Importing Formatted Image Data Programmatically" below
- "Importing Unformatted Image Files" on page 159
- "Exporting Formatted Image Files Programmatically" on page 160
- "Exporting Unformatted Image Files" on page 161

Note

These sections describe how to load data into a variable and includes examples of passing variable data to an iTool programmatically. See "Importing Data from the IDL Session" in Chapter 2 of the *iTool User's Guide* manual if you want information on how you can access variable data from the iTools Data Manager.

Importing Formatted Image Data Programmatically

The majority of IDL image data access routine require a file specification, indicating the file from which to access the data. The FILEPATH routine is often used within a data access routine as shown in the following example.

Note

To validate that an image file can be accessed using READ_* routines, you can query the image first. See "Returning Image File Information" on page 175 for details.

The following example opens a JPEG file from the `examples/data` directory, performs feature extraction, and displays both images using IIMAGE.

```
; Open a file and access the data.
file = FILEPATH('n_vasinfecta.jpg', $
   SUBDIRECTORY = ['examples', 'data'])
READ_JPEG, file, image, /GRAYSCALE

; Mask out pixel values greater than 120
; and create a distance map.
binaryImg = image LT 120
distanceImg = MORPH_DISTANCE(binaryImg, NEIGHBOR_SAMPLING = 1)
```

```
; Launch iImage, creating a 2 column, 1 row layout.
; Display the original and distanceImg in the two views.
IIMAGE, image, VIEW_GRID=[2,1]
IIMAGE, distanceImg, /VIEW_NEXT, /OVERPLOT
```

In the previous example, you could use the READ_IMAGE function instead of the READ_JPEG function by replacing the following statement:

```
READ_JPEG, file, image, /GRAYSCALE
```

with

```
image = READ_IMAGE(file)
```

In this instance, you do not have control over the color table associated with the image. It is often more useful to use a specific READ_* routine or object designed for the image file format to precisely control characteristics of the imported image.

For a list of available image access, import and export routines and objects, see "Image Data Formats" under the functional category "Input/Output" in the *IDL Quick Reference* manual.

Note

IDL can also import images stored in scientific data formats, such as HDF and netCDF. For more information on these formats, see the *Scientific Data Formats* manual.

Importing Unformatted Image Files

Images in unformatted binary files can be imported with the READ_BINARY function using the DATA_DIMS and DATA_TYPE keywords as follows:

- You must specify the size of the image within the file using the DATA_DIMS keyword. This is required because the READ_BINARY function assumes the data values are arranged in a single vector (a one-dimensional array). The DATA_DIMS keyword is used to specify the size of the two- or three-dimensional image array.
- You can set the DATA_TYPE keyword to the image's data type using the associated IDL type code (see "IDL Type Codes and Names" under the SIZE function in the *IDL Reference Guide* for a complete list of type code). Most images in binary files are of the byte data type, which is the default setting for the DATA_TYPE keyword.

No standard exists by which image parameters are provided in an unformatted binary file. Often, these parameters are not provided at all. In this case, you should already

be familiar with the size and type parameters of any images you need to access within binary files.

For example, the `worldelv.dat` file is a binary file that contains an image. You can only import this image by supplying the information that the data values of the image are byte and that the image has dimensions of 360 pixels by 360 pixels. Before using the READ_BINARY function to access this image, you must first determine the path to the file:

```
file = FILEPATH('worldelv.dat', $
   SUBDIRECTORY = ['examples', 'data'])
```

Define the size parameters of the image with a vector:

```
imageSize = [360, 360]
```

An image type parameter is not required because we know that the data values of image are byte, which is the default type for the READ_BINARY function.

The READ_BINARY function can now be used to import the image contained in the `worldelv.dat` file:

```
image = READ_BINARY(file, DATA_DIMS = imageSize)
IIMAGE, image
```

Exporting Formatted Image Files Programmatically

Images can be exported to common image file formats using the WRITE_IMAGE procedure. The WRITE_IMAGE procedure requires three inputs: the exported file's name, the image file type, and the image itself. You can also provide the red, green, and blue color components to an associated color table if these components exist.

For example, you can import the image from the `worldelv.dat` binary file:

```
file = FILEPATH('worldelv.dat', $
   SUBDIRECTORY = ['examples', 'data'])
imageSize = [360, 360]
image = READ_BINARY(file, DATA_DIMS = imageSize)
```

You can export this image to an image file (a JPEG file) with the WRITE_IMAGE procedure:

```
WRITE_IMAGE, 'worldelv.dat', 'JPEG', image
```

IDL also provides format-specific WRITE_* routines that are similar to the WRITE_IMAGE procedure, but provide more flexibility when exporting a specific image file type. See "Image Data Formats" under the functional category "Input/Output" in the *IDL Quick Reference* manual for a list of available image access, import and export routines and objects.

Note

IDL can also export images stored in scientific data formats, such as HDF and netCDF. For more information on these formats, see the *Scientific Data Formats* manual.

Exporting Unformatted Image Files

Images can be exported to an unformatted binary file with the WRITEU procedure. Before using the WRITEU procedure, you must open a file to which the data will be written using the OPENW procedure. Any file you open must be specifically closed using either the FREE_LUN or CLOSE procedure when you are done exporting the image.

For example, you can import the image from the `rose.jpg` image file:

```
file = FILEPATH('rose.jpg', $
   SUBDIRECTORY = ['examples', 'data'])
image = READ_IMAGE(file)
```

You can export this image to a binary file by first opening a new file:

```
OPENW, unit, 'rose.dat', /GET_LUN
```

Then, use the WRITEU procedure to write the image to the open file:

```
WRITEU, unit, image
```

You must remember to close the file once the data has been written to it:

```
FREE_LUN, unit
```

Note

For complete details about reading, writing and formatting unformatted data, see Chapter 18, “Files and Input/Output” in the *Building IDL Applications* manual.

Accessing Non-Image Data Programmatically

There are a number of options available for reading non-image data into IDL. Depending upon the file type, consider using one of the following:

- **Formatted data** — use a data-type-specific routine (such as READ_ASCII or READ_BINARY). See "Reading Binary Data in a Volume" below for more information.
- **Unformatted data** — use a general data access routines (such as OPEN or WRITE). For complete details about reading, writing and formatting unformatted data, see Chapter 18, "Files and Input/Output" in the *Building IDL Applications* manual.
- **SAVE file data** — use the RESTORE procedure to access variable data in a SAVE file. See "Reading Contour Data from a SAVE File" on page 163 for an example.

Note

These sections describe how to load data into a variable and includes examples of passing variable data to an iTool programmatically. See "Importing Data from the IDL Session" in Chapter 2 of the *iTool User's Guide* manual if you want information on how you can access variable data from the iTools Data Manager.

Reading Binary Data in a Volume

The following example uses READ_BINARY to access binary data (`head.dat`) consisting of a stack of 57 images slices of the human head. After reading the data, create a display using IVOLUME. Enter the following at the IDL command prompt:

```
file = FILEPATH('head.dat', $
   SUBDIRECTORY = ['examples', 'data'])
dataSize = [80,100,57]
volume= READ_BINARY(file, DATA_DIMS = dataSize)
iVolume, volume, /AUTO_RENDER
```

Note

You can also create a template for binary file access. See "Reading Binary Data" on page 154 for options.

Reading Contour Data from a SAVE File

You can also access information from a SAVE file. This example restores a SAVE file containing variable data (`marbells.dat`), configures the data, and displays it using ICONTOUR.

```
PRO maroonBellsContour_doc

; Restore Maroon Bells data into the IDL variable "elev".
RESTORE, FILEPATH('marbells.dat', SUBDIR=['examples','data'])

; Create x and y vectors giving the position of each
; column and row.
X = 326.850 + .030 * FINDGEN(72)
Y = 4318.500 + .030 * FINDGEN(92)

; Set missing data points to a large value. Reduce to a
; 72 x 92 matrix.
elev (WHERE (elev EQ 0)) = 1E6
new = REBIN(elev, 360/5, 460/5)

iContour, new, X, Y, C_VALUE = 2750 + FINDGEN(6) * 250.,$
 XSTYLE = 1, YSTYLE = 1, YMARGIN = 5, MAX_VALUE = 5000, $
   C_LINESTYLE = [1, 0], $
   C_THICK = [1, 1, 1, 1, 1, 3], $
   XTITLE = 'UTM Coordinates (KM)'

End
```

Note

See Chapter 4, “Creating SAVE Files of Programs and Data” in the *Building IDL Applications* manual for complete details on creating and restoring SAVE files.

Using IDL Macros

When you are working in the IDLDE, you can use a pre-defined macro to help you import image, ASCII, binary or HDF data. These macros call internal functions and return structures containing data. From the IDL command line, you can access and display data elements contained in the structures. These macros are available through the **Macros** menu and also through IDL toolbar buttons.

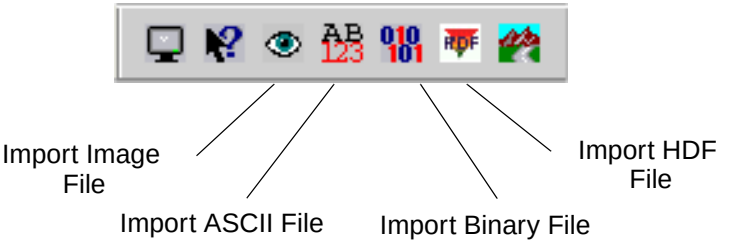


Figure 6-1: Macro Toolbar Buttons

See the follow sections for more information:

- “Using Macros to Import Image Files” on page 165
- “Using Macros to Import ASCII Files” on page 167
- “Using Macros to Import Binary Files” on page 169
- “Using Macros to Import HDF Files” on page 170

Using Macros to Import Image Files

To import an image file into IDL using a macro, complete the following steps:

1. Select the **Import Image** toolbar button. The **Select Image File** dialog is displayed.
2. Select a file to import. For example, select the `rsi-directory/examples/data/muscle.jpg` file where `rsi-directory` is the installation directory for IDL. See "Using the Select Image File Dialog Interface" under "DIALOG_READ_IMAGE" in the *IDL Reference Guide* manual for additional information if desired.
3. Click **Open**.

The `muscle.jpg` image data has been opened into a structure variable named MUSCLE_IMAGE. The **Import Image** macro opens and stores image data in a structure variable named *filename*_IMAGE where *filename* is the name of the file you opened without the extension.

Note

IDL variables must begin with a letter, and may contain only letters, digits, the underscore character, or the dollar sign. If the first character of *filename* is not a letter, the prefix "var" is added to the variable name. Any spaces within *filename* are converted to underscores. Any other illegal characters within *filename* are removed.

The MUSCLE_IMAGE structure contains the following fields:

- IMAGE — The actual image array.
- R — The red color table vectors.
- G — The green color table vectors.
- B — The blue color table vectors.
- QUERY — Contains information about the image.
  - CHANNELS — The number of channels in the image.
  - HAS_PALETTE — Specifies if the palette is present. 1 if the palette is present, else 0. If your image is *n*-by-*m* the palette is usually present and the R, G, and B color table vectors mentioned above will contain values. If your image is 3-by-*n*-by-*m*, the palette will not be present and the R,G, and B color table vectors will not contain any values.

- IMAGE_INDEX — The index of the image of the file. The default is 0, the first image in the file. If there are multiple images in the file that you read, this will be the number (or index) of the image.
- NUM_IMAGES — The number of images in the original file.
- PIXEL_TYPE — The IDL Type Code of the image pixel format. Valid types are described in "IDL Type Codes and Names" under "SIZE" in the *IDL Reference Guide* manual.
- TYPE — The image format type.

The structure can be viewed in the Variable Watch Window.

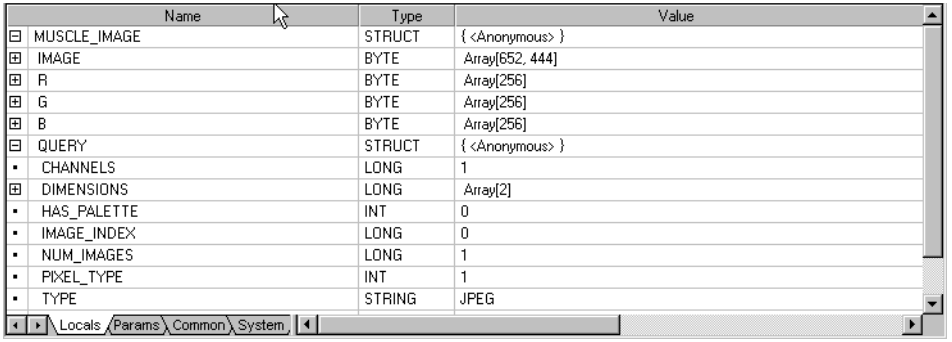


Figure 6-2: Variable Watch Window Showing MUSCLE_IMAGE Structure

You can specify which part of the structure variable you want to access by using the following syntax:

variable_name.element_name[.*element_name*]

For example, if you want to view the image, enter the following:

```
IIMAGE, MUSCLE_IMAGE.IMAGE
```

If you want to know the file type, enter the following:

```
PRINT, MUSCLE_IMAGE.QUERY.TYPE
```

IDL prints:

```
JPEG
```

Using Macros to Import ASCII Files

To import an ASCII file into IDL using a macro, complete the following steps:

1. Select the **Import ASCII** toolbar button. The **Select an ASCII file to read** dialog appears.
2. Select a file to import.
3. See "Using the ASCII Template Dialog" under "ASCII_TEMPLATE" in the *IDL Reference Guide* manual for instructions on how to use the dialog to define the structure of your ASCII data.

ASCII files opened with the **Import ASCII** macro are stored in structure variables which are named *filename*_ASCII where *filename* is the name of the file you opened without the extension.

Note

IDL variables must begin with a letter, and may contain only letters, digits, the underscore character, or the dollar sign. If the first character of *filename* is not a letter, the prefix "var" is added to the variable name. Any spaces within *filename* are converted to underscores. Any other illegal characters within *filename* are removed.

For example, if you opened `ascii.txt`, the data is now in the structure variable named ASCII_ASCII. Each field (named in the ASCII Template dialog) is an element of the structure.

The structure can be viewed in the Variable Watch Window.

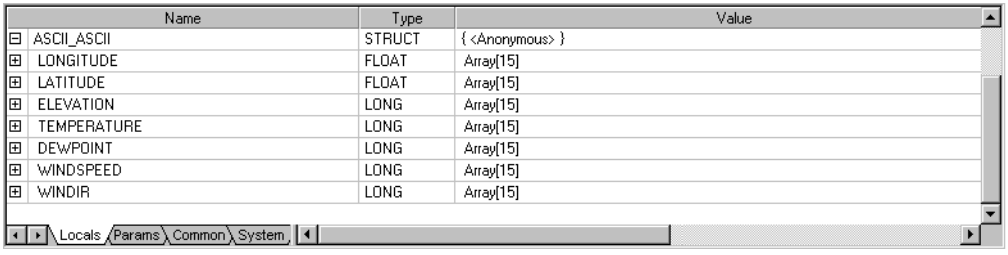


Figure 6-3: Variable Watch Window Showing ASCII_ASCII Structure

You can specify which part of the structure variable you want to access by using the following syntax:

variable_name.element_name

For example, if you want to view the Longitude field data, enter the following:

```
Print, ASCII_ASCII.LONGITUDE
```

If you want to plot the Temperature data, enter the following:

```
IPLOT, ASCII_ASCII.TEMPERATURE
```

The following figure results.

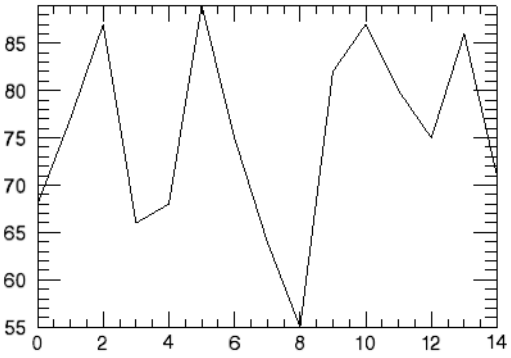


Figure 6-4: Plot of ASCII_ASCII.TEMPERATURE

Using Macros to Import Binary Files

To import a binary file into IDL using a macro, complete the following steps:

1. Select the **Import Binary** toolbar button. The **Select a binary file to read** dialog appears.
2. Select a file to import. For example, select the `surface.dat` from the `examples/data` directory in your IDL installation directory. Click **Open**.
3. See Using the BINARY_TEMPLATE Interface under "BINARY_TEMPLATE" in the *IDL Reference Guide* manual for instructions on how to use the dialog to define the structure of your binary data.

Binary files opened with the **Import Binary File** macro are stored in structure variables which are named *filename*_BINARY where *filename* is the name of the file you opened without the extension.

Note

IDL variables must begin with a letter, and may contain only letters, digits, the underscore character, or the dollar sign. If the first character of *filename* is not a letter, the prefix "var" is added to the variable name. Any spaces within *filename* are converted to underscores. Any other illegal characters within *filename* are removed.

So, the file we just opened (`surface.dat`) is now in the structure variable named SURFACE_BINARY. The variable is a structure, and contains elements that are the field names defined in the **Binary Template** dialog. In this case the single field is named `marbells`. The structure can be viewed in the Variable Watch Window.

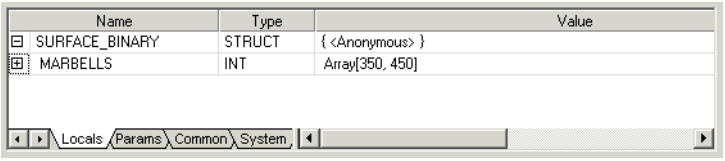


Figure 6-5: Variable Watch Window Showing MARBELLS_BINARY Structure

Access data from the structure variable using the following syntax:

variable_name.element_name

For example, display the surface by entering:

```
ISURFACE, SURFACE_BINARY.marbells
```

Using Macros to Import HDF Files

To import a Hierarchical Data Format (HDF), HDF-EOS, or NETCDF file into IDL, complete the following steps:

1. Select the **Import HDF File** toolbar button. The **Select a valid HDF, NETCDF or HDF-EOS file** dialog is displayed.
2. Select a file to import. Click **Open**.
3. See "Using the HDF Browser Interface" under "HDF_BROWSER" for instructions on how to use the dialog.

After selecting to import data and clicking **OK**, HDF, NETCDF, or HDF-EOS files read with the **Import HDF** macro are stored in structure variables which are named *filename*_DF where *filename* is the name of the file you opened without the extension.

Note

IDL variables must begin with a letter, and may contain only letters, digits, the underscore character, or the dollar sign. If the first character of *filename* is not a letter, the prefix "var" is added to the variable name. Any spaces within *filename* are converted to underscores. Any illegal characters within *filename* are removed.

The variable is a structure with each data or metadata name being an element of the structure. You can specify which part of the structure variable you want to access by using the following syntax:

variable_name.data_name

For example, if you imported two data elements out of a file named `hydrogen.hdf` and you named the elements `IMAGE1` and `IMAGE2`, you could access each individual data element using the following:

```
HYDROGEN_DF.IMAGE1
HYDROGEN_DF.IMAGE2
```

If you wanted to view `IMAGE1`, you would enter:

```
IIMAGE, HYDTROGEN_DF.IMAGE1
```

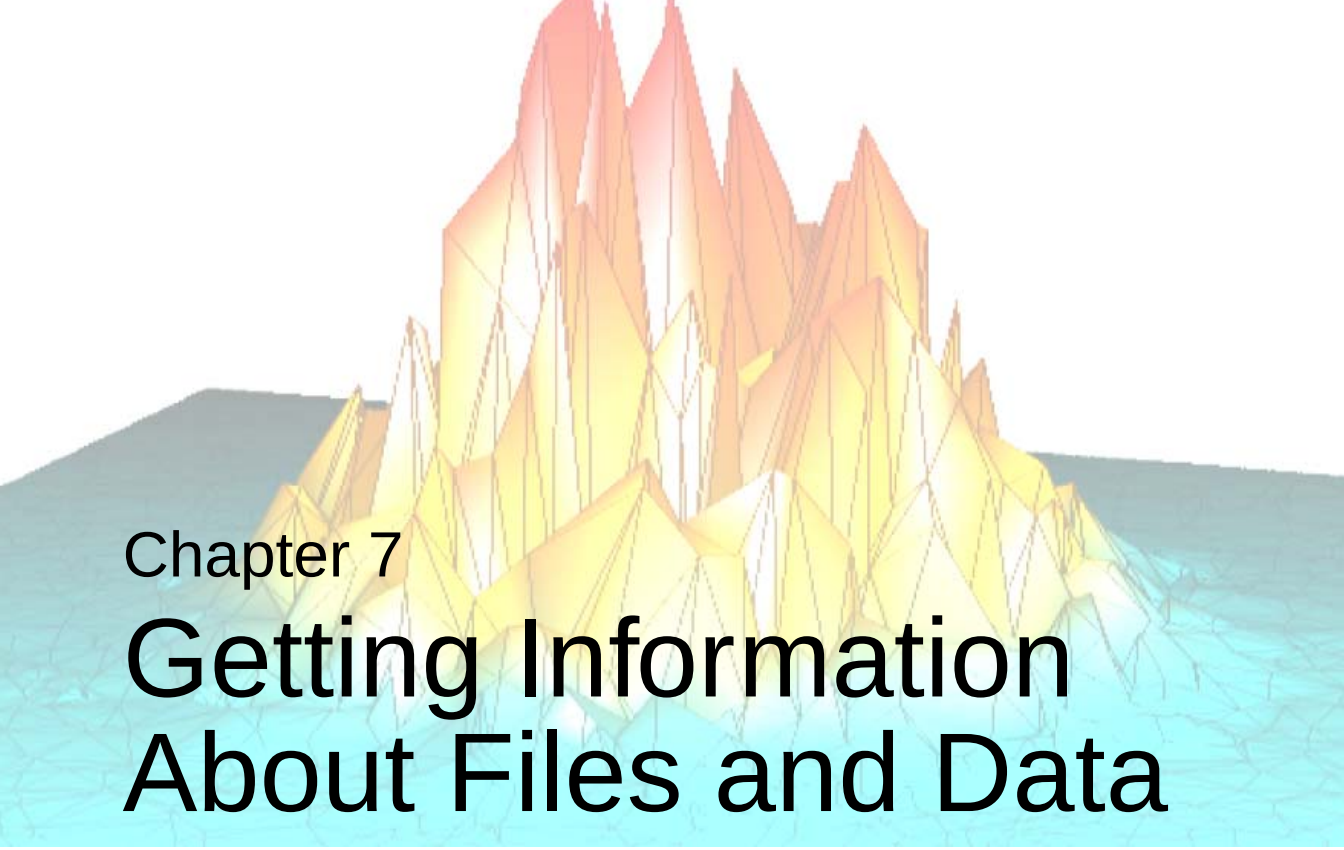
For more information on IDL support of HDF and other scientific data formats, see the *Scientific Data Formats* manual.

For information on importing HDF5 files using the **HDF5 Browser** dialog, see "H5_BROWSER" in the *IDL Reference Guide* manual

File Access Routines

See the following categories under "Input/Output" in the *IDL Quick Reference* manual for a list of available file and data access routines:

- "Image Data Formats" — includes read and write routines for supported image formats (such as JPEG, TIFF, DICOM, etc.), and routines that launch dialogs for image file access.
- "Scientific Data Formats" — includes CDF, EOS, NCDF, HDF, and HDF5 routines.
- "Other Data Formats" — includes routines that access ASCII, BINARY, XML, and other non-image data formats.
- "General Input/Output" — includes READ, WRITE and other routines commonly used when accessing unformatted data. Also see Chapter 18, "Files and Input/Output" for information on using these routines and formatting your data.



Chapter 7

Getting Information About Files and Data

The following topics are covered in this chapter:

Investigating Files and Data 174
Returning Image File Information 175
Returning Type and Size Information 179
Getting Information About SAVE Files . . 181
Returning Object Type and Validity 186
Returning Information About a File 188

Investigating Files and Data

There are a number of routines and functions in IDL that allow you to quickly access information about your data. While it is always a good idea to know your data before processing, the routines in this chapter can help you uncover details of arrays, expressions, SAVE files, objects, or specific images.

Accessing Information in iTools

When you are working in the iTools, there are a number of ways to get information about variable data, an object's properties, an image's statistical information, and the data hierarchy. For more information about these options, see the following topics:

- "About the Data Manager" in Chapter 2 of the *iTool User's Guide* manual provides information on data associated with a visualization
- "The Visualization Browser" in Chapter 6 of the *iTool User's Guide* manual provides information on the properties of a visualization
- "Additional Operations" in Chapter 7 of the *iTool User's Guide* manual describes the Histogram and Statistics windows available in iTools

Returning Image File Information

When accessing formatted image data (not contained in a binary file), there are a number of ways to get information about the data characteristics. The most flexible is the QUERY_IMAGE routine, which returns a structure that includes the number of image channels, pixel data type and palette information. If you need specific information from a formatted image file, you can use the QUERY* routine specifically designed for images of that format.

Note

You can also use the SIZE function to quickly return the size of an image array. See "Using SIZE to Return Image Dimensions" on page 180 for details.

Using the QUERY_IMAGE Info Structure

Common image file formats contain standardized header information that can be queried. IDL provides the QUERY_IMAGE function to return valuable information about images stored in supported image file formats.

For example, using the QUERY_IMAGE function, you can return information about the `mineral.png` file in the `examples/data` directory. First, access the file. Then use the QUERY_IMAGE function to return information about the file:

```
file = FILEPATH('mineral.png', $
   SUBDIRECTORY = ['examples', 'data'])
queryStatus = QUERY_IMAGE(file, info)
```

To determine the success of the QUERY_IMAGE function, print the value of the *query* variable:

```
PRINT, 'Status = ', queryStatus
```

IDL prints

```
queryStatus =           1
```

If *queryStatus* is zero, the file cannot be accessed with IDL. If *queryStatus* is one, the file can be accessed. Because the query was successful, the *info* variable is now an IDL structure containing image parameters. The tags associated with this structure variable are standard across image files. You can view the tags of this structure by setting the STRUCTURE keyword to the HELP command with the *info* variable as its argument:

```
HELP, info, /STRUCTURE
```

IDL displays the following text in the Output Log:

```
** Structure <1407e70>, 7 tags, length=36, refs=1:
   CHANNELS        LONG                 1
   DIMENSIONS      LONG         Array[2]
   HAS_PALETTE     INT                  1
   IMAGE_INDEX     LONG                 0
   NUM_IMAGES      LONG                 1
   PIXEL_TYPE      INT                  1
   TYPE            STRING           'PNG'
```

The structure tags provide the following information:

Tag	Description

CHANNELS	Provides the number of dimensions within the image array:
	• 1 – two-dimensional array
	• 3 – three-dimensional array
	Print the number of dimensions using:
	`PRINT, 'Number of Channels: ', info.channels`
	For the `mineral.png` file, IDL prints:
	`Number of Channels:  1`

DIMENSIONS	Contains image array information including the width and height. Print the image dimensions using:
	`PRINT, 'Size: ', info.dimensions`
	For the `mineral.png` file, IDL prints:
	`Size:  288  216`

HAS_PALETTE	Describes the presence or absence of a color palette:
	• 1 (True) – the image has an associated palette
	• 0 (False) – the image does not have an associated palette
	Print whether a palette is present or not using:
	`PRINT, 'Is Palette Available?: ', info.has_palette`
	For the `mineral.png` file, IDL prints:
	`Is Palette Available?:  1`

Table 7-1: Image Structure Tag Information

Tag	Description

IMAGE_INDEX	Gives the zero-based index number of the current image. Print the index of the image using:
	`PRINT, 'Image Index: ', info.image_index`
	For the `mineral.png` file, IDL prints:
	`Image Index:  0`

NUM_IMAGES	Provides the number of images in the file. Print the number of images in the file using:
	`PRINT, 'Number of Images: ', info.num_images`
	For the `mineral.png` file, IDL prints:
	`Number of Images:  1`

PIXEL_TYPE	Provides the IDL type code for the image pixel data type:
	• 0 – Undefined
	• 1 – Byte
	• 2 – Integer
	• 3 – Longword integer
	• 4 – Floating point
	• 5 – Double-precision floating
	• 6 – Complex floating
	• 9 – Double-precision complex
	• 12 – Unsigned Integer
	• 13 – Unsigned Longword Integer
	• 14 – 64-bit Integer
	• 15 – Unsigned 64-bit Integer
	See "IDL Type Codes and Names" under the SIZE function in the *IDL Reference Guide* for a complete list of type codes.
	Print the data type of the pixels in the image using:
	`PRINT, 'Data Type: ', info.pixel_type`
	For the `mineral.png` file, IDL displays the following text in the Output Log:
	`Data Type:  1`

Table 7-1: Image Structure Tag Information (Continued)

Tag	Description

TYPE	Identifies the image file format. Print the format of the file containing the image using:
	`PRINT, 'File Type: ' + info.type`
	For the `mineral.png` file, IDL prints:
	`File Type: PNG`

Table 7-1: Image Structure Tag Information (Continued)

From the contents of the *info* variable, it can be determined that the single image within the `mineral.png` file is an indexed image because it has only one channel (is a two-dimensional array) and it has a color palette. The image also has byte pixel data.

Note

When working with RBG images (with a CHANNELS value of 3) it is important to determine the interleaving (the arrangement of the red, green, and blue channels of data) in order to properly display these image. See "RGB Image Interleaving" in Chapter 8 of the *Using IDL* manual for an example that shows you how to determine the arrangement of these channels.

Using Specific QUERY_* Routines

All of the QUERY_* routines return a status, which determines if the file can be read using the corresponding READ_ routine. All of these routines also return the `Info` structure, (described in the previous section), which reports image dimensions, number of samples per pixel, pixel type, palette info, and the number of images in the file. However, some of the QUERY_* routines (such as QUERY_MRSID and QUERY_TIFF) return more detailed information particular to that specific image format. See "Query Routines" in the *IDL Quick Reference* manual for a complete list of the available QUERY_* routines.

Returning Type and Size Information

The SIZE function returns size and type information for a given expression. The returned vector is always of longword type.

- The first element is equal to the number of dimensions of the parameter and is zero if the parameter is a scalar.
- The next elements contain the size of each dimension.
- After the dimension sizes, the last two elements indicate the data type and the total number of elements, respectively.

See "IDL Type Codes and Names" under the SIZE function in the *IDL Reference Guide* for a complete list of type codes. See the following examples for more information on the SIZE function:

- "Determining if a Variable is a Scalar or an Array" below
- "Using SIZE to Return Image Dimensions" on page 180

In addition to the examples listed above, also see the following SIZE function examples in the *IDL Reference Guide*:

- "Example: Returning Array Dimension Information"
- "Example: Returning the IDL Type Code of an Expression"

Determining if a Variable is a Scalar or an Array

The SIZE function can be used to determine whether a variable holds a scalar value or an array. Setting the DIMENSIONS keyword causes the SIZE function to return a 0 if the variable is a scalar, or the dimensions if the variable is an array:

```
A = 1
B = [1]
C = [1,2,3]
D = [[1,2],[3,4]]

PRINT, SIZE(A, /DIMENSIONS)
PRINT, SIZE(B, /DIMENSIONS)
PRINT, SIZE(C, /DIMENSIONS)
PRINT, SIZE(D, /DIMENSIONS)
```

IDL Prints:

```
0
1
3
```

```
2     2
```

Using SIZE to Return Image Dimensions

The following example reads an image array and uses the SIZE function DIMENSIONS keyword to access the number of rows and columns in the image file. In this simple example, the information is used to create a display window of the correct size.

```
PRO ex_displayImage

; Select and read the image file.
earth = READ_PNG (FILEPATH ('avhrr.png', $
   SUBDIRECTORY = ['examples', 'data']), R, G, B)

; Load the color table and designate white to occupy the
; final position in the red, green and blue bands.
TVLCT, R, G, B
maxColor = !D.TABLE_SIZE - 1
TVLCT, 255, 255, 255, maxColor

; Prepare the display device.
DEVICE, DECOMPOSED = 0, RETAIN = 2

; Get the size of the original image array.
earthSize = SIZE(earth, /DIMENSIONS)

; Prepare a window and display the new image.
WINDOW, 0, XSIZE = earthSize[0], YSIZE = earthSize[1]
TV, earth

END
```

Getting Information About SAVE Files

The IDL_Savefile object provides an object-oriented interface that allows you to query a SAVE file for information and restore one or more individual items from the file. Using IDL_Savefile, you can retrieve information about the user, machine, and system that created the SAVE file, as well as the number and size of the various items contained in the file (variables, common blocks, routines, *etc*). Individual items can be selectively restored from the SAVE file.

Use IDL_Savefile in preference to 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.

Note

The IDL_Savefile object does not provide methods that allow you to modify an existing SAVE file. The only way to modify an existing SAVE file is to restore its contents into a fresh IDL session, modify the contained routines or variables as necessary, and use the SAVE procedure to create a new version of the file.

To use the IDL_Savefile object to restore items from an existing SAVE file, do the following:

- Create a Savefile Object
- Query the Savefile Object
- Restore Items from the Savefile Object
- Destroy the Savefile Object

The following sections describe each of these steps. For complete information on the IDL_Savefile object and its methods, see "IDL_Savefile" in Chapter 9 of the *IDL Reference Guide* manual.

Create a Savefile Object

When an IDL_Savefile object is instantiated, it opens the actual SAVE file for reading and creates an in-memory representation of its contents — without actually restoring the file. The savefile object persists until it is explicitly destroyed (or until the IDL session ends); the SAVE file itself is held open for reading as long as the savefile object exists.

To create a savefile object from the `draw_arrow.sav` file created in "Example: A SAVE File of a Simple Routine" in Chapter 4 of the *Building IDL Applications* manual, use the following command:

```
myRoutines = OBJ_NEW('IDL_Savefile', 'draw_arrow.sav')
```

Similarly, to create a savefile object from the saved image data, use the following command:

```
myImage = OBJ_NEW('IDL_Savefile', 'imagefile.sav')
```

Query the Savefile Object

Once you have created a savefile object, three methods allow you to retrieve information about its contents:

- The Contents method provides information about the SAVE file including the number and type of items contained therein.
- The Names method allows you to retrieve the names of routines and variables stored in the file.
- The Size method allows you to retrieve size and type information about the variables stored in the file.

Contents Method

The Contents method returns a structure variable that describes the SAVE file and its contents. The individual fields in the returned structure are described in detail in "IDL_Savefile::Contents" in Chapter 9 of the *IDL Reference Guide* manual.

In addition to providing information about the system that created the SAVE file, the Contents method allows you to determine the number of each type of saved item (variable, procedure, function, *etc*.) in the file. This information can be used to programmatically restore items from the SAVE file.

Assuming you have created the `myRoutines` savefile object, the data returned by the Contents method looks like this:

```
savefileInfo = myRoutines->Contents()
HELP, savefileInfo, /STRUCTURE
```

IDL Prints:

```
** Structure IDL_SAVEFILE_CONTENTS, 17 tags, length=176, data leng
th=172:
   FILENAME            STRING    '/rsi/test/draw_arrow.sav'
   DESCRIPTION         STRING    ''
   FILETYPE            STRING    'Portable (XDR)'
```

```
USER                STRING    'dquixote'
HOST                STRING    'DULCINEA'
DATE                STRING    'Thu May 08 12:04:46 2003'
ARCH                STRING    'x86'
OS                  STRING    'Win32'
RELEASE             STRING    '6.2'
N_COMMON            LONG64                    0
N_VAR               LONG64                    0
N_SYSVAR            LONG64                    0
N_PROCEDURE         LONG64                    2
N_FUNCTION          LONG64                    0
N_OBJECT_HEAPVAR    LONG64                    0
N_POINTER_HEAPVAR   LONG64                    0
N_STRUCTDEF         LONG64                    0
```

From this you can determine the name of the SAVE file from which the information was extracted, the names of the user and computer who created the file, the creation date, and information about the IDL system that created the file. You can also see that the SAVE file contains definitions for two procedures and nothing else.

Names Method

The Names method returns a string array containing the names of the variables, procedures, functions, or other items contained in the SAVE file. By default, the method returns the names of variables; keywords allow you to specify that names of other items should be retrieved. The available keyword options are described in "IDL_Savefile::Names" in Chapter 9 of the *IDL Reference Guide* manual.

The names of items retrieved using the Names method can be supplied to the Size method to retrieve size and type information about the specific items, or to the Restore method to restore individual items.

For example, calling the Names method with the PROCEDURE keyword on the `myRoutines` savefile object yields the names of the two procedures saved in the file:

```
PRINT, myRoutines->Names(/PROCEDURE)
```

IDL Prints:

```
ARROW  DRAW_ARROW
```

Similarly, to retrieve the name of the variable saved in `imagefile.sav`, which is referred to by the `myImage` savefile object:

```
PRINT, myImage->Names()
```

IDL Prints:

```
IMAGE
```

Size Method

The Size method returns the same information about a variable stored in a SAVE file as the SIZE function does about a regular IDL variable. It accepts the same keywords as the SIZE function, and returns the same information using the same formats. The Size method differs only in that the argument is a string or integer identifier string (returned by the Names method) that specifies an item within a SAVE file, rather than an in-memory expression. See "IDL_Savefile::Size" in Chapter 9 of the *IDL Reference Guide* manual for additional details.

For example, to determine the dimensions of the image stored in the `imagefile.sav` file, do the following:

```
imagesize = myImage->Size('image', /DIMENSIONS)
PRINT, 'Image X size:', imagesize[0]
PRINT, 'Image Y size:', imagesize[1]
```

IDL Prints:

```
Image X size:         256
Image Y size:         256
```

Restore Items from the Savefile Object

The Restore method allows you to selectively restore one or more items from the SAVE file associated with a savefile object. Items to be restored are specified using the item name strings returned by the Names method. In addition to functions, procedures, and variables, you can also restore COMMON block definitions, structure definitions, and heap variables. See "IDL_Savefile::Restore" in Chapter 9 of the *IDL Reference Guide* manual for additional details.

For example, to restore the DRAW_ARROW procedure without restoring the ARROW procedure, do the following:

```
myRoutines->Restore, 'draw_arrow'
```

Note on Restoring Objects and Pointers

Object references and pointers rely on special IDL variables called *heap variables*. When you restore a regular IDL variable that contains an object reference or a pointer, the associated heap variable is restored automatically; there is no need to restore the heap variables separately. It is, however, possible to restore the heap variables independently of any regular IDL variables; see "Restoring Heap Variables Directly" in Chapter 9 of the *IDL Reference Guide* manual for complete details.

Destroy the Savefile Object

To destroy a savefile object, use the OBJ_DESTROY procedure:

```
OBJ_DESTROY, myRoutines
OBJ_DESTROY, myImage
```

Destroying the savefile object will close the SAVE file with which the object is associated.

Returning Object Type and Validity

Three IDL routines allow you to obtain information about an existing object: OBJ_CLASS, OBJ_ISA, and OBJ_VALID.

OBJ_CLASS

Use the OBJ_CLASS function to obtain the class name of a specified object, or to obtain the names of a specified object's direct superclasses. For example, if we create the following class structures:

```
struct = {class1, data1:0.0 }
struct = {class2, data2a:0, data2b:0L, INHERITS class1 }
```

We can now create an object and use OBJ_CLASS to determine its class and superclass membership.

```
; Create an object.
A = OBJ_NEW('class2')

; Print A's class membership.
PRINT, OBJ_CLASS(A)
```

IDL prints:

```
CLASS2
```

Or you can print as superclasses:

```
; Print A's superclasses.
PRINT, OBJ_CLASS(A, /SUPERCLASS)
```

IDL prints:

```
CLASS1
```

See "OBJ_CLASS" in the *IDL Reference Guide* manual for further details.

OBJ_ISA

Use the OBJ_ISA function to determine whether a specified object is an instance or subclass of a specified object. For example, if we have defined the object A as above:

```
IF OBJ_ISA(A, 'class2') THEN $
    PRINT, 'A is an instance of class2.'
```

IDL prints:

```
A is an instance of class2.
```

See "OBJ_ISA" in the *IDL Reference Guide* manual for further details.

OBJ_VALID

Use the OBJ_VALID function to verify that one or more object references refer to valid and currently existing object heap variables. If supplied with a single object reference as its argument, OBJ_VALID returns TRUE (1) if the reference refers to a valid object heap variable, or FALSE (0) otherwise. If supplied with an array of object references, OBJ_VALID returns an array of TRUE and FALSE values corresponding to the input array. For example:

```
; Create a class structure.
struct = {cname, data:0.0}

; Create a new object.
A = OBJ_NEW('CNAME')

IF OBJ_VALID(A) PRINT, "A refers to a valid object." $
    ELSE PRINT, "A does not refer to a valid object."
```

IDL prints:

```
A refers to a valid object.
```

If we destroy the object:

```
; Destroy the object.
OBJ_DESTROY, A

IF OBJ_VALID(A) PRINT, "A refers to a valid object." $
    ELSE PRINT, "A does not refer to a valid object."
```

IDL prints:

```
A does not refer to a valid object.
```

See “OBJ_VALID” in the *IDL Reference Guide* manual for further details.

Returning Information About a File

You can use the FILE_INFO function to retrieve information about a file that is not currently open. To get information about an open file (for which there is an IDL Logical Unit Number), use the HELP procedure or the FSTAT function. See "Returning Information About a File Unit" in Chapter 18 of the *Building IDL Applications* manual.

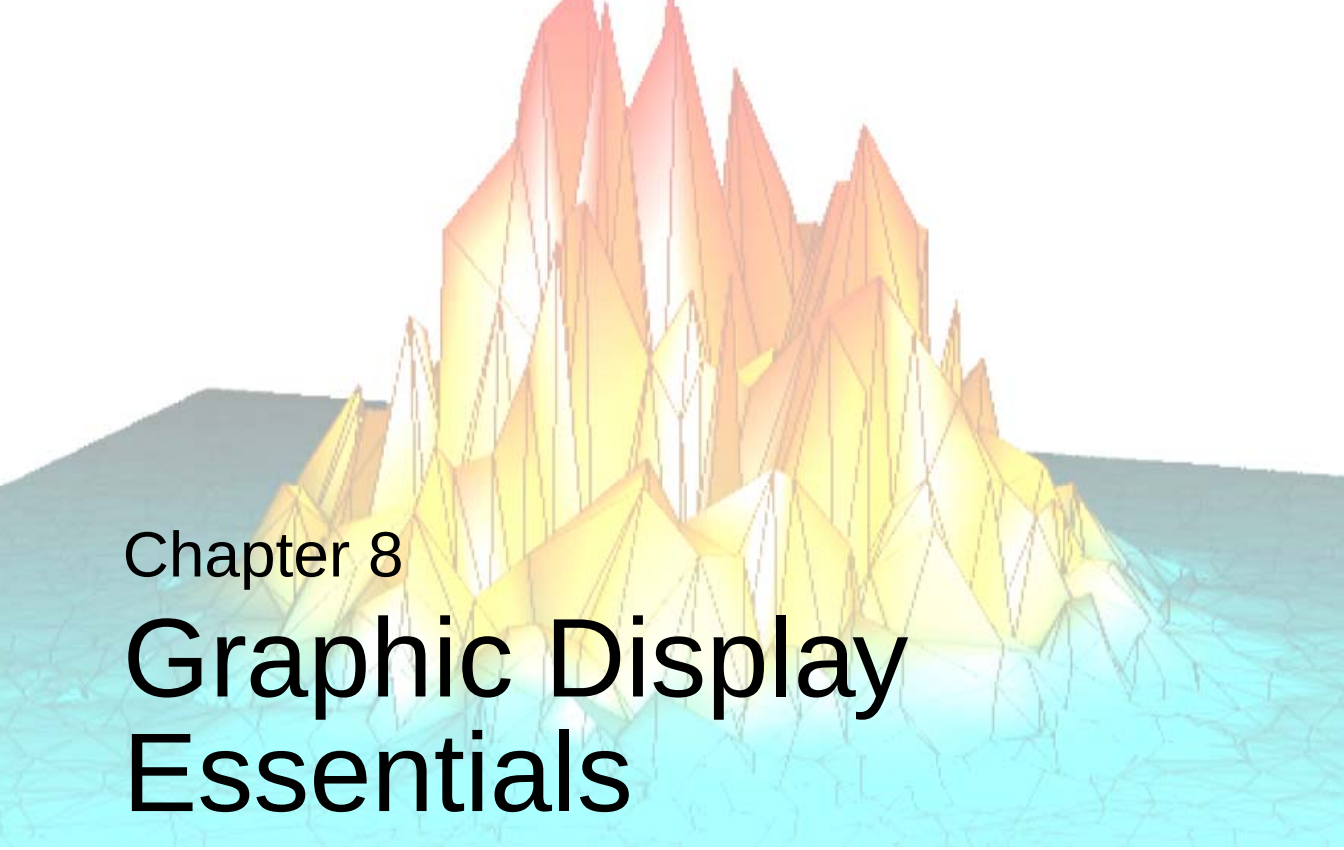
The FILE_INFO function returns a structure expression of type FILE_INFO containing information about the file. For example, get information on `dist.pro`:

```
HELP,/STRUCTURE, FILE_INFO(FILEPATH('dist.pro',
SUBDIRECTORY='lib'))
```

The above command will produce output similar to:

```
** Structure FILE_INFO, 21 tags, length=72:
        NAME            STRING    '/usr/local/rsi/idl/lib/dist.pro'
        EXISTS          BYTE         1
        READ            BYTE         1
        WRITE           BYTE         0
        EXECUTE         BYTE         0
        REGULAR         BYTE         1
        DIRECTORY       BYTE         0
        BLOCK_SPECIAL   BYTE         0
        CHARACTER_SPECIAL
                        BYTE         0
        NAMED_PIPE      BYTE         0
        SETGID          BYTE         0
        SETUID          BYTE         0
        SOCKET          BYTE         0
        STICKY_BIT      BYTE         0
        SYMLINK         BYTE         0
        DANGLING_SYMLINK
                        BYTE         0
        MODE            LONG               420
        ATIME           LONG64           970241431
        CTIME           LONG64           970241595
        MTIME           LONG64           969980845
        SIZE            LONG64                1717
```

The fields of the FILE_INFO structure provide various information about the file, such as the size of the file, and the dates of last access, creation, and last modification. For more information on the fields of the FILE_INFO structure, see "FILE_INFO" in the *IDL Reference Guide* manual. See "FILE_LINES" in the *IDL Reference Guide* manual for information on how to retrieve the number of lines of text in a file.



Chapter 8
Graphic Display Essentials

The following topics are covered in this chapter:

IDL Visual Display Systems 190
IDL Coordinate Systems 193
Coordinates of 3-D Graphics 195
Coordinate Conversions 198
Interpolation Methods 201
Polygon Shading Method 203
Color Systems 204
Display Device Color Schemes 207
Colors and IDL Graphic Systems 209
Indexed and RGB Image Organization . . 213
Loading a Default Color Table 218
Using Fonts in Graphic Displays 221
Printing Graphics 222

IDL Visual Display Systems

When creating visualizations in IDL, you can choose to create a visualization in an IDL Intelligent Tool (iTool), in an Object Graphics display, or in a Direct Graphics display:

- **iTools** — introduced in IDL 6.0, the IDL Intelligent Tools (iTools) provide the power and flexibility of Object Graphics with a pre-built visualization system that offers a great deal of interactivity. This set of interactive utilities combine data analysis and visualization with the task of producing presentation quality graphics. See "iTools Visualizations" below for more information.
- **Object Graphics** — introduced in IDL 5.0, Object Graphics use an object-oriented programmers' interface to create graphic objects, which must then be drawn, explicitly, to a destination of the programmer's choosing. See "IDL Object Graphics" on page 191 for more information.
- **Direct Graphics** — the oldest visualization system of the three, Direct Graphics rely on the concept of a current graphics device to quickly create simple static visualizations using IDL commands like PLOT or SURFACE. See "IDL Direct Graphics" on page 192 for information.

This chapter introduces the IDL display systems and provides information on common topics shared by the systems. Topics include a discussion on coordinates, coordinate conversion, interpolation, color systems and color schemes, and fonts.

iTools Visualizations

The new IDL Intelligent Tools (iTools) are a set of interactive utilities that combine data analysis and visualization with the task of producing presentation quality graphics. Based on the IDL Object Graphics system, 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 enjoying the convenience of a point-and-click environment.

The main enhancements the new iTools provide are more mouse interactivity, WYSIWYG (What-You-See-Is-What-You-Get) printing, built-in analysis, undo-redo capabilities, layout control, and better-looking plots. These robust, pre-built tools reduce the amount of programming IDL users must do to create interactive visualizations. At the same time, the iTools integrate in a seamless manner with the IDL Command Line, user interface controls, and custom algorithms. In this way, the iTools maintain and enhance the control and flexibility IDL users rely on for data

exploration, algorithm design, and rapid application development. The following manuals provide more information:

- *iTool User's Guide* — describes how to create visualization using iTools
- *iTool Developer's Guide* — describes how to create and customize an iTool

IDL Object Graphics

The salient features of Object Graphics are:

- Object graphics are device independent. There is no concept of a current graphics device when using object-mode graphics; any graphics object can be displayed on any physical device for which a destination object can be created.
- Object graphics are object-oriented. Graphic objects are meant to be created and re-used; you may create a set of graphic objects, modify their attributes, draw them to a window on your computer screen, modify their attributes again, then draw them to a printer device without reissuing all of the IDL commands used to create the objects. Graphics objects also encapsulate functionality; this means that individual objects include method routines that provide functionality specific to the individual object.
- Object graphics are rendered in three dimensions. Rendering implies many operations not needed when drawing Direct Graphics, including calculation of normal vectors for lines and surfaces, lighting considerations, and general object overhead. As a result, the time needed to render a given object—a surface, say—will often be longer than the time taken to draw the analogous image in Direct Graphics.
- Object Graphics use a programmer's interface. Unlike Direct Graphics, which are well suited for both programming and interactive, ad hoc use, Object Graphics are designed to be used in programs that are compiled and run. While it is still possible to create and use graphics objects directly from the IDL command line, the syntax and naming conventions make it more convenient to build a program offline than to create graphics objects on the fly.
- Because Object Graphics persist in memory, there is a greater need for the programmer to be cognizant of memory issues and memory leakage. Efficient design—remembering to destroy unused object references and cleaning up—will avert most problems, but even the best designs can be memory-intensive if large numbers of graphic objects (or large datasets) are involved.

For more information on creating Object Graphic visualizations see:

- *Object Programming* — this manual introduces using IDL objects and also describes how to create custom objects in IDL.
- "Object Class and Method Reference" in the *IDL Reference Guide* manual — this section in the *IDL Reference Guide* provides complete reference material describing IDL's object classes
- *iTool User's Guide* and *iTool Developer's Guide* — these manuals provide complete details about using and creating object-based iTool displays

IDL Direct Graphics

IDL Direct Graphics is the original graphics rendering system introduced in IDL. Graphic displays creating using Direct Graphics are static — once created, no changes can be made without recreating the visualization being displayed. If you have used routines such as PLOT or SURFACE, you are already familiar with this graphics system. The salient features of Direct Graphics are:

- Direct Graphics use a graphics device (**X** for X-windows systems displays, **WIN** for Microsoft Windows displays, **PS** for PostScript files, etc.). You switch between graphics devices using the SET_PLOT command, and control the features of the current graphics device using the DEVICE command.
- IDL commands that existed in IDL 4.0 use Direct Graphics. Commands like PLOT, SURFACE, XYOUTS, MAP_SET, etc. all draw their output directly on the current graphics device.
- Once a direct-mode graphic is drawn to the graphics device, it cannot be altered or re-used. This means that if you wish to re-create the graphic on a different device, you must re-issue the IDL commands to create the graphic.
- When you add a new item to an existing direct-mode graphic (using a routine like OPLOT or XYOUTS), the new item is drawn in front of the existing items.

See "Direct Graphics" in the *IDL Quick Reference* manual for a list of available routines.

IDL Coordinate Systems

You can specify coordinates to IDL in one of the following coordinate systems:

DATA Coordinates

This coordinate system is established by the most recent PLOT, CONTOUR, or SURFACE procedure. This system usually spans the plot window, the area bounded by the plot axes, with a range identical to the range of the plotted data. The system can have two or three dimensions and can be linear, logarithmic, or semi-logarithmic. The mechanisms of converting from one coordinate system to another are described below.

DEVICE Coordinates

This coordinate system is the physical coordinate system of the selected plotting device. Device coordinates are integers, ranging from (0, 0) at the bottom-left corner to (V_x –1, V_y –1) at the upper-right corner. V_x and V_y are the number of columns and rows addressed by the device. These numbers are stored in the system variable !D as !D.X_SIZE and !D.Y_SIZE. In a widget base, device coordinates are measures from the upper-left corner

NORMAL Coordinates

The normalized coordinate system ranges from zero (0) to one (1) over each of the three axes.

Almost all of the IDL graphics procedures accept parameters in any of these coordinate systems. Most procedures use the data coordinate system by default. Routines beginning with the letters TV are notable exceptions. They use device coordinates by default. You can explicitly specify the coordinate system to be used by including one of the keyword parameters /DATA, /DEVICE, or /NORMAL in the call.

Understanding Windows and Related Device Coordinates

Images are displayed within a window (Direct Graphics) or within an instance of a window object (Object Graphics). In Direct Graphics, the WINDOW procedure is used to initialize the coordinates system for the image display. In Object Graphics,

the IDLgrWindow, IDLgrView, and IDLgrModel objects are used to initialize the coordinate system for the image display.

A coordinate system determines how and where the image appears within the window. You can specify coordinates to IDL using one of the following coordinate systems:

- Data Coordinates — This system usually spans the window with a range identical to the range of the data. The system can have two or three dimensions and can be linear, logarithmic, or semi-logarithmic.
- Device Coordinates — This coordinate system is the physical coordinate system of the selected device. Device coordinates are integers, ranging from (0, 0) at the bottom-left corner to (V_x –1, V_y –1) at the upper-right corner of the display. V_x and V_y are the number of columns and rows of the device (a display window for example).

  Note

  For images, the data coordinates are the same as the device coordinates. The device coordinates of an image are directly related to the pixel locations within an image. Unless otherwise specified, IDL draws each image pixel per each device pixel.

- Normal Coordinates — The normalized coordinate system ranges from zero to one over columns and rows of the device.

Coordinates of 3-D Graphics

Points in *xyz* space are expressed by vectors of homogeneous coordinates. These vectors are translated, rotated, scaled, and projected onto the two-dimensional drawing surface by multiplying them by transformation matrices. The geometrical transformations used by IDL, and many other graphics packages, are taken from Chapters 7 and 8 of Foley and Van Dam (Foley, J.D., and A. Van Dam (1982), *Fundamentals of Interactive Computer Graphics*, Addison-Wesley Publishing Co.). The reader is urged to consult this book for a detailed description of homogeneous coordinates and transformation matrices since this section presents only an overview. Three-dimensional graphics, coordinate systems, and transformations also are included in this chapter.

Homogeneous Coordinates

A point in homogeneous coordinates is represented as a four-element column vector of three coordinates and a scale factor w ¼¼≠ 0. For example:

$$P(wx, wy, wz, w) \equiv P(x/w, y/w, z/w, 1) \equiv (x, y, z)$$

One advantage of this approach is that translation, which normally must be expressed as an addition, can be represented as a matrix multiplication. Another advantage is that homogeneous coordinate representations simplify perspective transformations. The notion of rows and columns used by IDL is opposite that of Foley and Van Dam (1982). In IDL, the column subscript is first, while in Foley and Van Dam (1982) the row subscript is first. This changes all row vectors to column vectors and transposes matrices.

Right-Handed Coordinate System

The coordinate system is right-handed so that when looking from a positive axis to the origin, a positive rotation is counterclockwise. As usual, the *x*-axis runs across the display, the *y*-axis is vertical, and the positive *z*-axis extends out from the display to the viewer. For example, a 90-degree positive rotation about the *z*-axis transforms the *x*-axis to the *y*-axis.

Transformation Matrices

Transformation matrices, which post-multiply a point vector to produce a new point vector, must be (4, 4). A series of transformation matrices can be concatenated into a single matrix by multiplication. If A1, A2, and A3 are transformation matrices to be

applied in order, and the matrix A is the product of the three matrices, the following applies.

$$((P \bullet A_1) \bullet A_2) \bullet A_3 \equiv \mathrm{P} \bullet ((A_1 \bullet A_2) \bullet A_3) = P \bullet A$$

In Object Graphics, IDL the model object that contains the displayed object stores the transformation matrix. In Direct Graphics, IDL stores the concatenated transformation matrix in the system variable field !P.T.

Note

When displaying objects in a three-dimensional view, you can precisely configure the object position using transformation matrices. See "Translating, Rotating and Scaling Objects" in Chapter 3 of the *Object Programming* manual for details.

Note

For most Direct Graphic applications, it is not necessary to create, manipulate, or to even understand transformation matrices. See the T3D procedure, which implements most of the common transformations.

Each of the operations of translation, scaling, rotation, and shearing can be represented by a transformation matrix.

Translation

The transformation matrix to translate a point by (D_x, D_y, D_z) is shown below.

$$\begin{bmatrix} 1 & 0 & 0 & \mathrm{D_x} \\ 0 & 1 & 0 & \mathrm{D_y} \\ 0 & 0 & 1 & \mathrm{D_z} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Scaling

Scaling by factors of S_x, S_y, and S_z about the *x*-, *y*-, and *z*-axes respectively, is represented by the matrix below.

$$\begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation

Rotation about the *x*-, *y*-, and *z*-axes is represented respectively by the following three matrices:

$$R_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta_x & -\sin\theta_x & 0 \\ 0 & \sin\theta_x & \cos\theta_x & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_y = \begin{bmatrix} \cos\theta_y & 0 & \sin\theta_y & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta_y & 0 & \cos\theta_y & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_z = \begin{bmatrix} \cos\theta_z & -\sin\theta_z & 0 & 0 \\ \sin\theta_z & \cos\theta_z & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Coordinate Conversions

Depending upon the data and type of visualization, you may want to convert between normalized, data or device coordinates (described in "IDL Coordinate Systems" on page 193). This section details two-dimensional and three-dimensional coordinate system characteristics provides resources for various coordinate conversions. See the following for details:

- "Two-Dimensional Coordinate Conversion" on page 198
- "Three-Dimensional Coordinate Conversion" on page 199
- "Using Coordinate Conversions" on page 199

Two-Dimensional Coordinate Conversion

This section describes the formulae for conversions to and from each coordinate system. In the following discussion, D_x is a data coordinate, N_x is a normalized coordinate, and R_x is a raw device coordinate. Let V_x and V_y represent the size of the visible area of the currently selected display or drawing surface.

The field S is a two-element array of scaling factors used to convert X coordinates from data units to normalized units. S contains the parameters of the linear equation, converting data coordinates to normalized coordinates. S[0] is the intercept, and S[1] is the slope. Also, let D_x be the data coordinate, N_x the normalized coordinate, R_x the device coordinate, V_x the device X size (in device coordinates).

With the above variables defined, the linear two-dimensional coordinate conversions for the *x* coordinate can be written as follows:

Coordinate Conversion	Linear	Logarithmic
Data to normal	$N_x = S_0 + S_1 D_x$	$N_x = S_0 + S_1 \log D_x$
Data to device	$R_x = V_x(S + S_1 D_x)$	$R_x = V_x(S_0 + S_1 \log D_x)$
Normal to device	$R_x = N_x V_x$	$R_x = N_x V_x$
Normal to data	$D_x = (N_x - S_0)/S_1$	$D_x = 10^{(N_x - S_0)/S_1}$
Device to data	$D_x = (R_x/V_x - S_0)/S_1$	$D_x = 10^{(R_x/V_x - S_0)/S_1}$
Device to normal	$N_x = R_x/V_x$	$N_x = R_x/V_x$

Table 8-1: Equations for X-axis Coordinate Conversion

The *y*- and *z*-axis coordinates are converted in exactly the same manner, with the exception that there is no *z* device coordinate and that logarithmic *z*-axes are not permitted.

This coordinate conversion functionality is built into object graphics through the XCOORD_CONVERT and YCOORD_CONVERT properties or each type of visualization object. If you are working with a Direct Graphics display, you can use the CONVERT_COORD function.

Three-Dimensional Coordinate Conversion

To convert from a three-dimensional coordinate to a two-dimensional coordinate, IDL follows these steps:

- Data coordinates are converted to three-dimensional normalized coordinates. To convert the *x* coordinate from data to normalized coordinates, use the formula $N_x = X_0 + X_1 D_x$. The same process is used to convert the *y* and *z* coordinates using !Y.S and !Z.S.
- The three-dimensional normalized coordinate, P = (N_x, N_y, N_z), whose homogeneous representation is (N_x, N_y, N_z, 1), is multiplied by the concatenated transformation matrix !P.T:

  $P' = P \bullet$ `!P.T`
- The vector P′ is scaled by dividing by *w*, and the normalized two-dimensional coordinates are extracted:

  $N'_x = P'_x / P'_w$ and $N'_y = P'_y / P'_w$
- The normalized *xy* coordinate is converted to device coordinates as described in “Two-Dimensional Coordinate Conversion” on page 198.

Using Coordinate Conversions

How coordinate conversions are defined depend upon the display type as follows:

- **iTools** — in an iTool display, the interactive nature of the tool makes coordinate conversions transparent. There is no need to programmatically configure the transformation matrices of the objects. See Chapter 4, “Manipulating the Display” in the *iTool User’s Guide* manual for information on zooming, scaling and translation.
- **Object Graphics** — converting an object’s data coordinates into normalized coordinates for display is a common task. See “Positioning Visualizations in a View” in Chapter 3 of the *Object Programming* manual for details on the

elements involved in defining an object's position. Chapter 3, "Positioning Objects in a View" in the *Object Programming* manual also includes information on how to use coordinate conversions (see "Converting Data to Normal Coordinates") and information on programmatically defining the object's placement in a view (see "Translating, Rotating and Scaling Objects").

- **Direct Graphics** — the IDL Direct Graphics system automatically positions and sizes static visualizations so there is no need to set up a transformation matrix. However, you can convert between the supported coordinate systems. See "CONVERT_COORD" in the *IDL Reference Guide* manual for information on this conversion in Direct Graphics.

Interpolation Methods

When a visualization undergoes a geometric transformation, the location of each transformed pixel may not map directly to a center of a pixel location in the output visualization as shown in the following figure.

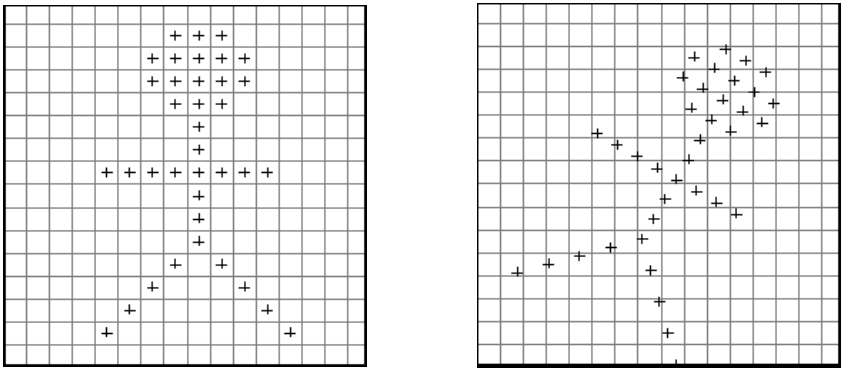


Figure 8-1: Original Pixel Center Locations (Left) and Rotated Pixel Center Locations (Right)

When the transformed pixel center does not directly coincide with a pixel in the output visualization, the pixel value must be determined using some form of interpolation. The appearance and quality of the output image is determined by the amount of error created by the chosen interpolation method. Note the differences in the line edges between the following two interpolated images.

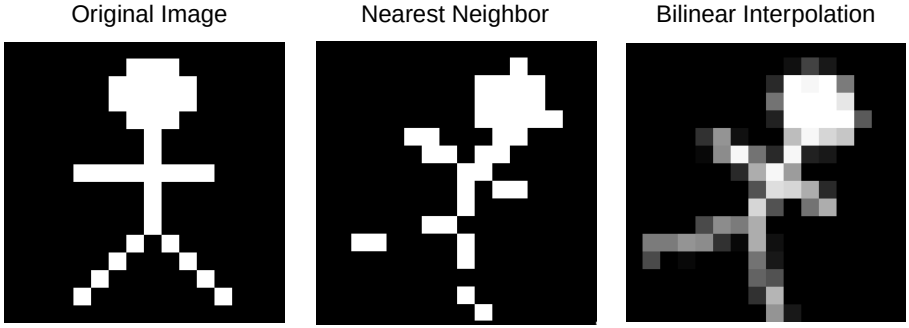


Figure 8-2: Simple Examples of Image Interpolation

There are a variety of possible interpolation methods available when using geometric transforms in IDL. Interpolation methods include:

Nearest-neighbor interpolation — Assigns the value of the nearest pixel to the pixel in the output visualization. This is the fastest interpolation method but the resulting image may contain jagged edges.

Linear interpolation — Surveys the 2 closest pixels, drawing a line between them and designating a value along that line as the output pixel value.

Bilinear interpolation — Surveys the 4 closest pixels, creates a weighted average based on the nearness and brightness of the surveyed pixels and assigns that value to the pixel in the output image.

Use cubic convolution if a higher degree of accuracy is needed. However, with still images, the difference between images interpolated with bilinear and cubic convolution methods is usually undetectable.

Trilinear interpolation — Surveys the 8 nearest pixels occurring along the x, y, and z dimensions, creates a weighted average based on the nearness and brightness of the surveyed pixels and assigns that value to the pixel in the output image.

Cubic Convolution interpolation — Approximates a sinc interpolation by using cubic polynomial waveforms instead of linear waveforms when resampling a pixel. With a one-dimension source, this method surveys 4 neighboring pixels. With a two-dimension source, the method surveys 16 pixels. Interpolation of three-dimension sources is not supported. This interpolation method results in the least amount of error, thus preserving the highest amount of fine detail in the output image. However, cubic interpolation requires more processing time.

Note

The *IDL Reference Guide* details the interpolation options available for each geometric transformation function.

Polygon Shading Method

The shading applied to each polygon, defined by its four surrounding elevations, can be either constant over the entire cell or interpolated. Constant shading takes less time because only one shading value needs to be computed for the entire polygon. Interpolated shading gives smoother results. The Gouraud method of interpolation is used: the shade values are computed at each elevation point, coinciding with each polygon vertex. The shading is then interpolated along each edge, finally, between edges along each vertical scan line.

Light-source shading is computed using a combination of depth cueing, ambient light, and diffuse reflection, adapted from Foley and Van Dam, Chapter 19 (Foley, J.D., and A. Van Dam (1982), *Fundamentals of Interactive Computer Graphics*, Addison-Wesley Publishing Co.):

$$I = I_a + dI_p(\boldsymbol{L} \bullet \boldsymbol{N})$$

where

I_a	Term due to ambient light. All visible objects have at least this intensity, which is approximately 20 percent of the maximum intensity.
$I_p(\boldsymbol{L} \bullet \boldsymbol{N})$	Term due to diffuse reflection. The reflected light is proportional to the cosine of the angle between the surface normal vector $\boldsymbol{N}$ and the vector pointing to the light source, $\boldsymbol{L}$. I_p is approximately 0.9.
d	Term for depth cueing, causing surfaces further away from the observer to appear dimmer. The normalized depth is $d=(z+2)/3$, ranging from zero for the most distant point to one for the closest.

In Direct Graphics, the SET_SHADING method modifies the light source shading parameters. In Object Graphics similar OpenGL functionality is available through the SHADING property of objects such as IDLgrPolygon, IDLgrPolyline, IDLSurface and IDLgrContour.

Color Systems

Color can play a critical role in the display and perception of digital imagery. This section provides a basic overview of color systems, display devices, image types, and the interaction of these elements within IDL. The remainder of the chapter builds upon these fundamental concepts by describing how to load and modify color tables, convert between image types, utilize color tables to highlight features, and apply color annotations to images.

Color Schemes

Color can be encoded using a number of different schemes. Many of these schemes utilize a color triple to represent a location within a three-dimensional color space. Examples of these systems include RGB (red, green, and blue), HSV (hue, saturation, and value), HLS (hue, lightness, and saturation), and CMY (cyan, magenta, and yellow). Algorithms exist to convert colors from one system to another.

Computer display devices typically rely on the RGB color system. In IDL, the RGB color space is represented as a three-dimensional Cartesian coordinate system, with the axes corresponding to the red, green, and blue contributions, respectively. Each axis ranges in value from 0 (no contribution) to 255 (full contribution). By design, this range from 0 to 255 maps nicely to the full range of a byte data type.

An individual color is encoded as a coordinate within this RGB space. Thus, a color consists of three elements: a red value, a green value, and a blue value.

The following figure shows that each displayable color corresponds to a location within a three-dimensional color cube. The origin, (0, 0, 0), where each color coordinate is 0, is black. The point at (255, 255, 255) is white, representing an additive mixture of the full intensity of each of the three colors. Points along the main diagonal - where intensities of each of the three primary colors are equal - are shades

of gray. The color yellow is represented by the coordinate (255, 255, 0), or a mixture of 100% red, plus 100% green, and no blue.

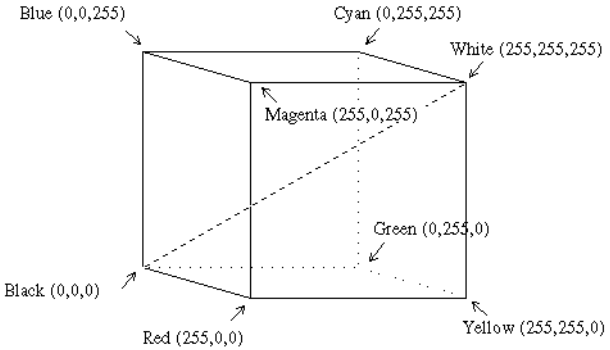


Figure 8-3: RGB Color Cube (Note: grays are on the main diagonal.)

Typically, digital display devices represent each component of an RGB color coordinate as an *n*-bit integer in the range of 0 to $2^n - 1$. Each displayable color is an RGB coordinate triple of *n*-bit numbers yielding a palette containing 2^{3n} total colors. Therefore, for 8-bit colors, each color coordinate can range from 0 to 255, and the total palette contains 2^{24} or 16,777,216 colors.

A display with an *m*-bit pixel can represent 2^m colors simultaneously, given enough pixels. In the case of 8-bit colors, 24-bit pixels are required to represent all colors. The more common case is a display with 8 bits per pixel which allows the display of $2^8 = 256$ colors selected from the much larger palette.

If there are not enough bits in a pixel to represent all colors, $m < 2^{3n}$, a color translation table is used to associate the value of a pixel with a color triple. This table is an array of color triples with an element for each possible pixel value. Given 8-bit pixels, a color table containing $2^8 = 256$ elements is required. The color table element with an index of *i* specifies the color for pixels with a value of *i*.

To summarize, given a display with an *n*-bit color representation and an *m*-bit pixel, the color translation table, *C*, is a 2^m long array of RGB triples:

$$C_i = \{r_i, g_i, b_i\}, \quad 0 \le i < 2^m$$

$$0 \le r_i, g_i, b_i < 2^n$$

Objects containing a value, or color index, of *i* are displayed with a color of C_i.

See "Color Table Manipulation" in the *IDL Quick Reference* manual for a list of color-related routines including those that covert RGB color triples to other color schemes.

Converting to Other Color Systems

IDL defaults to the RGB color system, but if you are more accustomed to other color systems, IDL is not restricted to working with only the RGB color system. You can also use either the HSV (hue, saturation, and value) system or the HLS (hue, lightness, and saturation) system. The HSV or HLS system can be specified by setting the appropriate keyword (for example /HSV or /HLS) when using IDL color routines.

IDL also contains routines to create color tables based on these color systems. The HSV routine creates a color table based on the Hue, Saturation, and Value (HSV) color system. The HLS routine creates a color table based on the Hue, Lightness, Saturation (HLS) color system. You can also convert values of a color from any of these systems to another with the COLOR_CONVERT routine. See `COLOR_CONVERT` in the *IDL Reference Guide* for more information.

Display Device Color Schemes

Most modern computer monitors use one of two basic schemes for displaying color at each pixel:

- **Indexed** - A color is specified using an index into a hardware color lookup table (or palette). Each entry of the color lookup table corresponds to an individual color, and consists of a red value, a green value, and a blue value. The size of the lookup table depends upon the hardware.
- **RGB** - A color is specified using an RGB triple: [red, green, blue]. The number of bits used to represent each of the red, green, and blue components depends upon the hardware.

The description of how color is to be interpreted on a given display device is referred to as a visual. Each visual typically has a name that indicates how color is to be represented. Two very common visual names are PseudoColor (which uses an indexed color scheme) and TrueColor (which uses an RGB color scheme).

A visual also has a depth associated with it that describes how many bits are used to represent a given color. Common bit depths include 8-bit (for PseudoColor visuals) and 16- or 24-bit (for TrueColor visuals). An n-bit visual is capable of displaying 2n total colors. Thus, an 8-bit PseudoColor visual can display 2^8 or 256 colors. A 24-bit TrueColor visual can display 2^{24} or 16,777,216 colors.

PseudoColor visuals rely heavily upon the display device's hardware color table for image display. If the color table is modified, all images being displayed using that color table will automatically update to reflect the change.

TrueColor visuals do not typically use a color table. The red, green, and blue components are provided directly.

Note

You can display TrueColor images on pseudo-color displays by using the COLOR_QUAN function. This function creates a pseudo-color palette for displaying the TrueColor image and then maps the TrueColor image to the new palette. See COLOR_QUAN in the *IDL Reference Guide* for more information.

Setting a Visual on Unix Platforms

On Unix platforms, an application (such as IDL) may choose from among the set of X visuals that are supported for the current display. Each visual is either grayscale or color. Its corresponding color table may be either fixed (read-only), or it may be changeable from within IDL (read-write). The color interpretation scheme is either

indexed or RGB. The following table shows the supported visuals for a given display, which may include any combination:

Visual	Description
StaticGray	grayscale, read-only, indexed
GrayScale	grayscale, read-write, indexed
StaticColor	color, read-only, indexed
PseudoColor	color, read-write, indexed
TrueColor	color, read-only, RGB
DirectColor	color, read-write, RGB

Table 8-2: Visuals Supported in IDL on Unix Platforms

The most common of these is PseudoColor and TrueColor. Refer to the section "Colors and IDL Graphic Systems" on page 209 to learn more about how IDL selects a visual for image display.

To get the list of supported X visual classes on a given system, type the following command at the Unix command line:

```
xdpyinfo
```

Setting a Visual on Windows Platforms

On Windows platforms, the visual is selected via the system Control Panel. To open the Control Panel, select the **Settings** → **Control Panel** item from the **Start** menu. Click on the **Display** and then select the **Settings** tab. Alter the **Color quality** setting to modify the visual before starting an IDL session. The following table shows three visuals are supported (for the particular display configuration used in this example):

Visual	Equivalence to Unix Visuals
256 Colors	8-bit PseudoColor
High Color (16 bit)	16-bit TrueColor
True Color (32 bit)	32-bit TrueColor

Table 8-3: Visuals Supported in IDL on Windows Platforms

Colors and IDL Graphic Systems

IDL supports two graphics systems: Object Graphics and Direct Graphics. This section provides detailed descriptions of how color is represented and interpreted in the Direct Graphics system.

Using Color in Object Graphics

For complete details regarding color and Object Graphics, see "Color in Object Graphics" in Chapter 2 of the *Object Programming* manual.

Using Color in Direct Graphics

More information on the following topics is available in "X Windows Visuals" in Appendix A of the *IDL Reference Guide* manual.

Visuals on UNIX Platforms

When IDL creates its first Direct Graphics window, it must select a visual to be associated with that window. By default, IDL selects an X Visual Class by requesting (in order) from the following table until a supported visual is found, but a specific visual can be explicitly requested at the beginning of an IDL session by setting the appropriate keyword to the DEVICE procedure:

Order	Visual	Depth	Related Keyword
First	TrueColor	24-bit (then 16-bit, then 15-bit)	TRUE_COLOR
Second	PseudoColor	8-bit, then 4-bit	PSEUDO_COLOR
Third	DirectColor	24-bit	DIRECT_COLOR
Fourth	StaticColor	8-bit, then 4-bit	STATIC_COLOR
Fifth	GrayScale	any depth	GRAY_SCALE
Sixth	StaticGray	any depth	STATIC_GRAY

Table 8-4: Order of Visuals and their Related DEVICE Keywords

To request an 8-bit PseudoColor visual, the syntax would be:

```
DEVICE, PSEUDO_COLOR=8
```

Another approach to setting the visual information is to include the `idl.gr_visual` and `idl.gr_depth` resources in your `.Xdefaults` file.

A visual is selected once per IDL session (when the first graphic window is created). Once selected, the same visual will be used for all Direct Graphics windows in that IDL session.

Private versus Shared Colormaps

On UNIX platforms, when a window manager is started, it creates a default colormap that can be shared among applications using the display. This is called the shared colormap.

A given application may request to use its own colormap that is not shared with other applications. This is called a private colormap.

IDL attempts, whenever possible, to get color table entries in the shared colormap. If enough colors are not available in the shared colormap, a private colormap is used. If an X Visual class and depth are specified and they do not match the default visual of the screen (see `xdpyinfo`), a private colormap is used.

If a private colormap is used, then colormap flashing may occur when an IDL window is made current (in which case, the colors of other applications on the desktop may no longer appear as you would expect), or when an application using the shared colormap is made current (in which case, the colors within the IDL graphics window may no longer appear as you would expect). This flashing behavior is to be expected. By design, the IDL graphics window has been assigned a dedicated color table so that the full range of requested colors can be utilized for image display.

Visuals on Windows Platforms

On Windows platforms, the visual that IDL uses is dependent upon the system setting. For more information, “Setting a Visual on Windows Platforms” on page 208.

IDL Color Table

IDL maintains a single current color table for Direct Graphics. Refer to the sections “Loading a Default Color Table” on page 218 and “Modifying and Converting Color Tables” on page 219. IDL provides 41 pre-defined color tables.

Foreground Color

In IDL Direct Graphics, colors used for drawing graphic primitives (such as lines, text annotations, etc.) are represented in one of two ways:

- Indexed - each color is an index into the current IDL color table
- RGB - each color is a long integer that contains the red value in the first eight bits, the green value in the next eight bits, and the blue value in the next eight bits. In other words, a color can be represented using the following equation:

```
color = red + 256*green + (256^2)*blue
```

The RGB form is only supported on TrueColor display devices.

The DECOMPOSED keyword to the DEVICE procedure is used to notify IDL whether color is to be interpreted as an index or as a composite RGB value. IDL then maps any requested color to an encoding that is appropriate for the current display device.

The foreground color (used for drawing) can be set by assigning a color value to the !P.COLOR system variable field (or by setting the COLOR keyword on the individual graphic routine).

If a color value is to be interpreted as an index, then inform IDL by setting the DECOMPOSED keyword of the DEVICE routine to 0:

```
DEVICE, DECOMPOSED = 0
```

The foreground color can then be specified by setting !P.COLOR to an index into the IDL color table. For example, if the foreground color is to be set to the RGB value stored at entry 25 in the IDL color table, then use the following IDL command:

```
!P.COLOR = 25
```

If a color value is to be interpreted as a composite RGB value, then inform IDL by setting the DECOMPOSED keyword of the DEVICE routine to 1:

```
DEVICE, DECOMPOSED = 1
```

The foreground color can then be specified by setting !P.COLOR to a composite RGB value. For example, if the foreground color is to be set to the color yellow, [255,255,0], then use the following IDL command:

```
!P.COLOR = 255 + (256*255)
```

Image Colors

Color for image data is handled in a fashion similar to other graphic primitives, except that some special cases apply based upon the organization of the image data and the visual of the current display device.

If the image is organized as a:

- two-dimensional array -

    - If the display device is PseudoColor, then each pixel is interpreted as an index into the IDL color table
    - If the display device is TrueColor and if the DECOMPOSED keyword for the DEVICE procedure is set to 0, then each pixel value is interpreted as an index into the IDL color table (thereby emulating a PseudoColor display device).
    - If the display device is TrueColor and if the DECOMPOSED keyword for the DEVICE procedure is set to 1, then each pixel value is interpreted as the value to be copied to each of the red, green, and blue components of the RGB color.
- RGB array - (Supported only for TrueColor display devices)
    - Each pixel is interpreted as an RGB color composed of the three elements in the extra color dimension of the array.

To display an RGB image on a PseudoColor device, use the COLOR_QUAN routine to convert it to an indexed form. Refer to the section “Converting Between Image Types” on page 217.

The TV command can be used to display the image in IDL. For RGB images, the TRUE keyword can be used to indicate which form of interleaving is used.

Indexed and RGB Image Organization

IDL can display four types of images: binary, grayscale, indexed, and RGB. How an image is displayed depends upon its type. Binary images have only two values, zero and one. Grayscale images represent intensities and use a normal grayscale color table. Indexed images use an associated color table. RGB images contain their own color information in layers known as bands or channels. Any of these images can be displayed with iImage, Object Graphics, or Direct Graphics.

An image consists of a two-dimensional array of pixels. The value of each pixel represents the intensity and/or color of that position in the scene. Images of this form are known as sampled or raster images, because they consist of a discrete grid of samples. Such images come from many different sources and are a common form of representing scientific and medical data.

Numerous standards have been developed over the years to describe how an image can be stored within a file. However, once the image is loaded into memory, it typically takes one of two forms: indexed or RGB. An indexed image is a two-dimensional array, and is usually stored as byte data. A two-dimensional array of a different data type can be made into an indexed image by scaling it to the range from 0 to 255 using the BYTSCL function. See the BYTSCL description *in the IDL Reference Guide* for more information.

Image Orientation

The screen coordinate system for image displays puts the origin, (0, 0), at the lower-left corner of the device. The upper-right corner has the coordinate (*xsize*–1, *ysize*–1), where *xsize* and *ysize* are the dimensions of the visible area of the display. The descriptions of the image display routines that follow assume a display size of 512 x 512, although other sizes may be used.

The system variable !ORDER controls the order in which the image is written to the screen. Images are normally output with the first row at the bottom, i.e., in bottom-to-top order, unless !ORDER is 1, in which case images are written on the screen from top to bottom. The ORDER keyword also can be specified with TV and TVSCL. It works in the same manner as !ORDER except that its effect only lasts for the duration of the single call—the default reverts to that specified by !ORDER.

An image can be displayed with any of the eight possible combinations of axis reversal and transposition by combining the display procedures with the ROTATE function.

Indexed Images

An indexed image does not explicitly contain any color information. Its pixel values represent indices into a color Look-Up Table (LUT). Colors are applied by using these indices to look up the corresponding RGB triplet in the LUT. In some cases, the pixel values of an indexed image reflect the relative intensity of each pixel. In other cases, each pixel value is simply an index, in which case the image is usually intended to be associated with a specific LUT. In this case, the LUT is typically stored with the image when it is saved to a file. For information on the LUTs provided with IDL, see "Loading a Default Color Table" on page 218.

RGB Image Interleaving

An RGB (red, green, blue) image is a three-dimensional byte array that explicitly stores a color value for each pixel. RGB image arrays are made up of width, height, and three channels of color information. Scanned photographs are commonly stored as RGB images. The color information is stored in three sections of a third dimension of the image. These sections are known as color channels, color bands, or color layers. One channel represents the amount of red in the image (the red channel), one channel represents the amount of green in the image (the green channel), and one channel represents the amount of blue in the image (the blue channel).

Color interleaving is a term used to describe which of the dimensions of an RGB image contain the three color channel values. Three types of color interleaving are supported by IDL. In Object Graphics, an RGB image is contained within an image object where the INTERLEAVE property dictates the arrangement of the channels within the image file.

- Pixel interleaving (3, w, h) — the color information is contained in the first dimension, INTERLEAVE is set to 0.
- Line interleaving (w, 3, h) — the color information is contained in the second dimension, INTERLEAVE is set to 1.
- Planar interleaving (w, h, 3) — the color information is contained in the third dimension, INTERLEAVE is set to 2. This is also known as, image interleaving.

Note

In Direct Graphics, set the TRUE keyword of TV or TVSCL to match the interleaving of the image.

Determining RGB Image Interleaving

You can determine if an image file contains an RGB image by querying the file. The CHANNELS tag of the resulting query structure will equal 3 if the file's image is RGB. The query does not determine which interleaving is used in the image, but the array returned in DIMENSIONS tag of the query structure can be used to determine the type of interleaving.

The following example queries and imports a pixel-interleaved RGB image from the `rose.jpg` image file. This RGB image is a close-up photograph of a red rose. It is pixel interleaved. Complete the following steps for a detailed description of the process.

Example Code

See `displayrgbimage_object.pro` in the `examples/doc/image` subdirectory of the IDL installation directory for code that duplicates this example.

1. Determine the path to the `rose.jpg` file:

```
file = FILEPATH('rose.jpg', $
   SUBDIRECTORY = ['examples', 'data'])
```

2. Use QUERY_IMAGE to query the file to determine image parameters:

```
queryStatus = QUERY_IMAGE(file, imageInfo)
```

3. Output the results of the file query:

```
PRINT, 'Query Status = ', queryStatus
HELP, imageInfo, /STRUCTURE
```

   The following text appears in the Output Log:

```
Query Status =            1
** Structure <14055f0>, 7 tags, length=36, refs=1:
   CHANNELS      LONG      3
   DIMENSIONS    LONG      Array[2]
   HAS_PALETTE   INT       0
   IMAGE_INDEX   LONG      0
   NUM_IMAGES    LONG      1
   PIXEL_TYPE    INT       1
   TYPE          STRING    'JPEG'
```

   The CHANNELS tag has a value of 3. Thus, the image is an RGB image.

4. Set the image size parameter from the query information:

```
imageSize = imageInfo.dimensions
```

The type of interleaving can be determined from the image size parameter and actual size of each dimension of the image. To determine the size of each dimension, you must first import the image.

5. Use READ_IMAGE to import the image from the file:

```
image = READ_IMAGE(file)
```

6. Determine the size of each dimension within the image:

```
imageDims = SIZE(image, /DIMENSIONS)
```

7. Determine the type of interleaving by comparing the dimension sizes to the image size parameter from the file query:

```
interleaving = WHERE((imageDims NE imageSize[0]) AND $
   (imageDims NE imageSize[1]))
```

8. Output the results of the interleaving computation:

```
PRINT, 'Type of Interleaving = ', interleaving
```

The following text appears in the Output Log:

```
Type of Interleaving = 0
```

The image is pixel interleaved. If the resulting value was 1, the image would have been line interleaved. If the resulting value was 2, the image would have been planar interleaved.

9. Initialize the display objects:

```
oWindow = OBJ_NEW('IDLgrWindow', RETAIN = 2, $
   DIMENSIONS = imageSize, TITLE = 'An RGB Image')
oView = OBJ_NEW('IDLgrView', $
   VIEWPLANE_RECT = [0., 0., imageSize])
oModel = OBJ_NEW('IDLgrModel')
```

10. Initialize the image object:

```
oImage = OBJ_NEW('IDLgrImage', image, $
   INTERLEAVE = interleaving[0])
```

11. Add the image object to the model, which is added to the view, then display the view in the window:

```
oModel -> Add, oImage
oView -> Add, oModel
oWindow -> Draw, oView
```

The following figure shows the resulting RGB image display.

Figure 8-4: RGB Image in Object Graphics

12. Clean up the object references. When working with objects always remember to clean up any object references with the OBJ_DESTROY routine. Since the view contains all the other objects, except for the window (which is destroyed by the user), you only need to use OBJ_DESTROY on the view object.

```
OBJ_DESTROY, oView
```

Converting Between Image Types

Sometimes an image type must be converted from indexed to RGB, RGB to grayscale, or RGB to indexed. For example, an image may be imported into IDL as an indexed image (from a PNG file for example) but it may need to be exported as an RGB image (to a JPEG file for example). The opposite may also need to be done. See “Foreground Color” on page 210 for more information on grayscale, indexed, and RGB images.

See the following routines s in the *IDL Reference Guide* for examples:

- **RGB to grayscale** — REFORM extracts the individual channels of data from an RGB image so that it can be displayed as a grayscale image
- **RGB to indexed** — COLOR_QUAN decomposes the millions of possible colors in an RGB image into the 256 used by an indexed image
- **Indexed to RGB** — TVLCT extracts the indexed image color table information, which is then assigned to an RGB image

Loading a Default Color Table

Although you can define your own color tables, IDL provides 41 pre-defined color lookup tables (LUTs). Each color table contained within this routine is specified through an index value ranging from 0 to 40, shown in the following table.

Tip

If you are running IDL on a TrueColor display, set `DEVICE, DECOMPOSED = 0` before your first color table related routine is used within an IDL session or program. See "Foreground Color" on page 210 for more information.

Number	Name	Number	Name
0	Black & White Linear	21	Hue Sat Value 1
1	Blue/White Linear	22	Hue Sat Value 2
2	Green-Red-Blue-White	23	Purple-Red + Stripes
3	Red Temperature	24	Beach
4	Blue-Green-Red-Yellow	25	Mac Style
5	Standard Gamma-II	26	Eos A
6	Prism	27	Eos B
7	Red-Purple	28	Hardcandy
8	Green/White Linear	29	Nature
9	Green/White Exponential	30	Ocean
10	Green-Pink	31	Peppermint
11	Blue-Red	32	Plasma
12	16 Level	33	Blue-Red 2
13	Rainbow	34	Rainbow 2
14	Steps	35	Blue Waves

Table 8-5: Pre-defined Color Tables

Number	Name	Number	Name
15	Stern Special	36	Volcano
16	Haze	37	Waves
17	Blue-Pastel-Red	38	Rainbow18
18	Pastels	39	Rainbow + white
19	Hue Sat Lightness 1	40	Rainbow + black
20	Hue Sat Lightness 2		

Table 8-5: Pre-defined Color Tables (Continued)

You can load a default color table in an iImage display, an Object Graphics Display or a Direct Graphics display as follows:

- iImage — select the **Edit Palette** button on the image panel. See "Using the Image Panel" in Chapter 10 of the *iTool User's Guide* manual for details.
- Object Graphics — use the LoactCT method of an IDLgrPalette object to define the color table (see "IDLgrPalette::LoadCT" in the *IDL Reference Guide* manual for details). Associate the palette object with another object using the Palette property (for example, see the PALETTE property of the IDLgrImage object). Also see "Color in Object Graphics" in Chapter 2 of the *Object Programming* manual for information on using color with indexed and RGB color models in Object Graphics.
- Direct Graphics — use the LOADCT routine or another color table related routine to set the color table. Also see "Using Color in Direct Graphics" on page 209.

Note

See "Color Table Manipulation" in the *IDL Quick Reference* manual for a list of related routines.

Modifying and Converting Color Tables

IDL contains two graphical user interface (GUI) utilities for modifying a color table, XLOADCT and XPALETTE (. The MODIFYCT routine lets you create or modify

and store a new color table. See the following topics in the *IDL Reference Guide* for examples:

- XLOADCT — allows you to preview and select among pre-defined color tables
- XPALETTE — allows you to preview and adjust pre-defined color tables
- MODIFYCT — shows how to add modified color tables to IDL's list of pre-defined color tables.

These examples are based on the default RGB (red, green, and blue) color system. IDL also contains routines that allow you to use other color systems including hue, saturation, and value (HSV) and hue, lightness, and saturation (HLS). These routines and color systems are explained in "Converting to Other Color Systems" on page 206.

Highlighting Features with a Color Table

For indexed images, custom color tables can be derived to highlight specific features. Color tables are usually designed to vary within certain ranges to show dramatic changes within an image. Some color tables are designed to highlight features with drastic color change in adjacent ranges (for example setting 0 through 20 to black and setting 21 through 40 to white).

Note

Color tables are associated with indexed images. RGB images already contain their own color information. If you want to derive a color table for an RGB image, you should convert it to an indexed image with the COLOR_QUAN routine. You should also set COLOR_QUAN's CUBE keyword to 6 to insure the resulting indexed image is an intensity representation of the original RGB image. See `COLOR_QUAN` in the *IDL Reference Guide* for more information

See the following topics in the *IDL Reference Guide* for examples:

- IDLgrPalette provides an example that creates, defines and applies a palette object to an image
- TVLCT creates, defines and applies a color table in a Direct Graphics display
- H_EQ_CT applies histogram equalization to a color table to reveal previously indistinguishable feature

Using Fonts in Graphic Displays

IDL uses three font systems for writing characters on the graphics device, whether that device be a display monitor or a printer: Hershey (vector) fonts, TrueType (outline) fonts, and device (hardware) fonts. Fonts are discussed in detail in Appendix H, "Fonts" in the *IDL Reference Guide* manual.

Both TrueType and Vector fonts are displayed identically on all of the platforms that support IDL. This means that if your cross-platform application uses either the TrueType fonts supplied with IDL or the Vector fonts, there is no need for platform-dependent code.

In a widget application, specify a font using the FONT keyword. If you choose a device font, you may need to write platform-dependent code. See "Fonts Used in Widget Applications" in Chapter 9 of the *Building IDL Applications* manual for details.

To set the font in an Object Graphics display, create an IDLgrFont object and assign this object to a text object using the IDLgrText object FONT property. See "Font Objects" in Chapter 9 of the *Object Programming* manual for more information.

Note

Within the IDLDE, you can specify what font is used in various areas (e.g., the Editor window or the Output Log window). See "Font Preferences" in Chapter 3 of the *Using IDL* manual for details.

Printing Graphics

Beginning with IDL version 5.0, IDL interacts with a system-level printer manager to allow printing of both IDL Direct Graphics and IDL Object Graphics. On Windows platforms, IDL uses the operating system's built-in printing facilities; on UNIX platforms, IDL uses the Xprinter print manager from Bristol Technology.

Use the DIALOG_PRINTERSETUP and DIALOG_PRINTJOB functions to configure your system printer and control individual print jobs from within IDL.

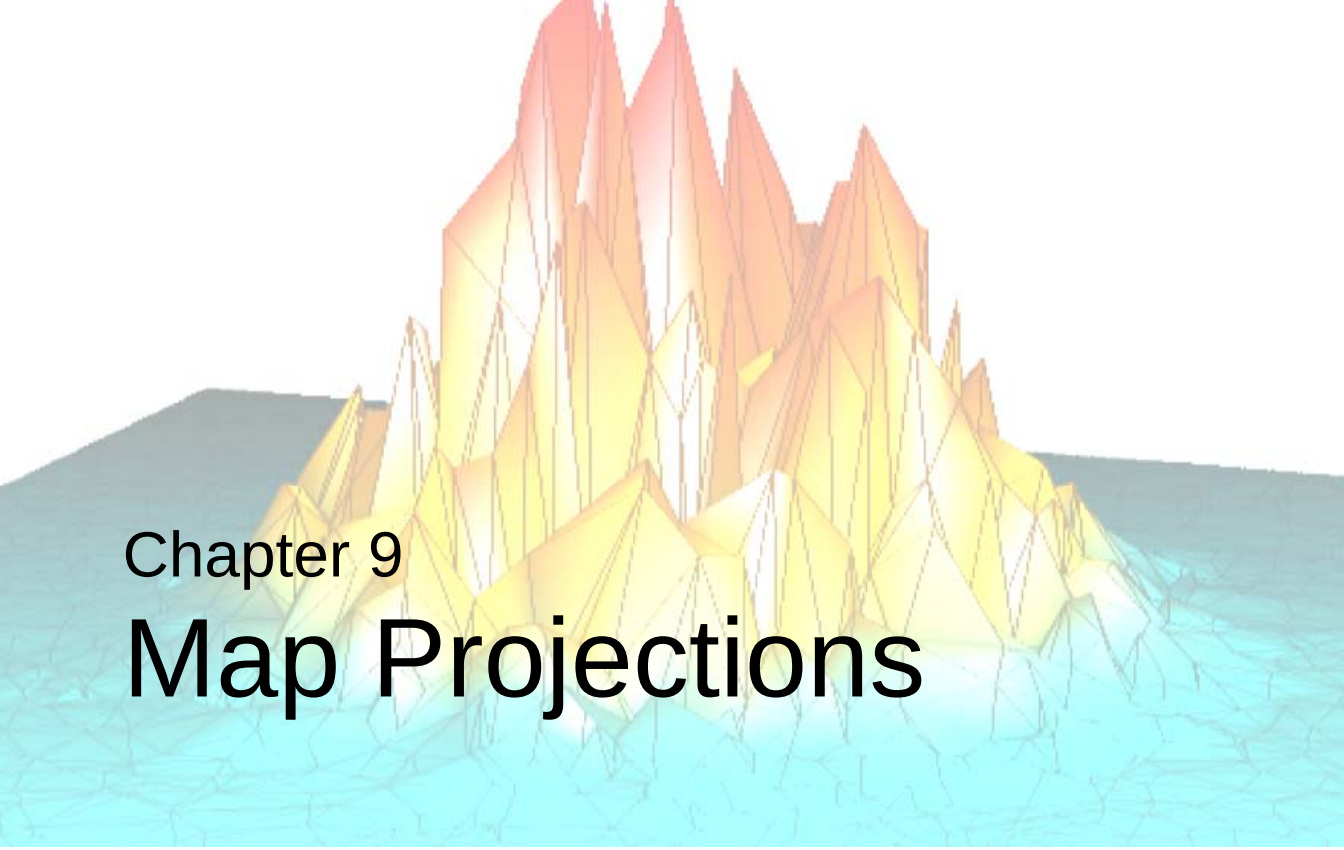
Printing IDL Direct Graphics

To print IDL Direct Graphics, you must first use the SET_PLOT procedure to make PRINTER your current device. Issue IDL commands as normal to create the graphics you wish to print, then use the CLOSE_DOCUMENT keyword to DEVICE to actually initiate the print job and print something from your printer. You can also create multiple pages before closing the document as well as being able to use tile graphics with the !P.MULTI system command.

See "Printing Graphics Output Files" in Appendix A of the *IDL Reference Guide* manual for details and examples.

Printing IDL Object Graphics

To print IDL Object Graphics, you must create a printer object to use as a destination for your Draw operations. You can also print multiple documents with the IDLgrPrinter object. See "Printer Objects" in Chapter 12 of the *Object Programming* manualfor information about printer objects and examples of their use. Also see "Bitmap and Vector Graphic Output" in Chapter 12 of the *Object Programming* manual for information of when to output to bitmap or vector graphics based on picture content.



Chapter 9
Map Projections

The following topics are covered in this chapter:

Overview of Mapping 224
Graphics Techniques for Mapping 225
Map Projection Types 227
Azimuthal Projections 228
Cylindrical Projections 237
Pseudocylindrical Projections 242
High-Resolution Continent Outlines 246
References 248

Overview of Mapping

This section introduces graphic map display considerations as well as information about common map projections. This section does not describe how to create a map display. See the following topic for these resources.

Creating a Map Display

IDL provides interactive and static map display functionality. You can use the iMap iTool to interactively configure a map display. If you prefer a static display, you can use map routines. See the following for details:

- Interactive iMap display — see Chapter 15, "Working with Maps" in the *iTool User's Guide* manual
- Map-related routines — see "Mapping" in the *IDL Quick Reference* manual

Examples of Creating Map Displays

See the following resources in the *IDL Reference Guide* for examples:

- IMAP — provides examples of displaying images and contours over a map projection.
- MAP_PROJ_FORWARD — creates a latitude and longitude grid with labels for a Goodes Homolosine map projection in an Object Graphics display. Typically MAP_PROJ_INIT is used with MAP_PROJ_FORWARD and MAP_PROJ_INVERSE.
- MAP_SET — establishes the coordinate conversion mechanism for mapping points on a globe's surface to points on a plane, according to the selected projections type. You can then use MAP_GRID and MAP_CONTINENTS to add grid lines and continents to the map display. See MAP_IMAGE for an example of warping an image to a projection.

Graphics Techniques for Mapping

Standard graphics techniques are insufficient when projecting areas on a sphere to a two-dimensional surface for two reasons. First, two points on a sphere are connected by two different lines. Second, areas may wrap around the edges of cylindrical and pseudo-cylindrical projections.

Graphical entities on the surface of a sphere can be properly represented on any map by using a combination of the following four stages: splitting, 3D clipping, projection, and rectangular clipping. The IMAP and MAP_SET procedures automatically sets up the proper mapping technique to best fit the projection selected by the user.

Warning

For proper rendering, splitting, and clipping, polygons must be traversed in counter-clockwise order when observed from outside the sphere. If this requirement is not met, the exterior, instead of the interior, of the polygons may be filled. Also, vectors connecting the points spanning the singular line for cylindrical projections will be drawn in the wrong direction if polygons are not traversed in the correct order.

Splitting

The splitting stage is used for cylindrical and pseudo-cylindrical projections. The singular line, one half of a great circle line, is located opposite the center of the projection; points on this line appear on both edges of the map. The singular line is the intersection of the surface of the sphere with a plane passing through the center of projection, one of the poles of projections, and the center of the sphere.

3D Clipping

Map graphics are clipped to one side of an arbitrary clipping plane in one or more clipping stages. For example, to draw a hemisphere centered on a given point, the clipping plane passes through the center of the sphere and has a normal vector that coincides with the given point.

Projection

In the projection stage, a point expressed in latitude and longitude is transformed to a point on the mapping plane.

Rectangular Clipping

After the map graphics have been projected onto the mapping plane, a conventional rectangular clipping stage ensures that the graphics are properly bounded and closed in the rectangular display area.

Map Projection Types

In the following sections, the available IDL projections are discussed in detail. The projections are grouped within three categories:

- “Azimuthal Projections” on page 228
- “Cylindrical Projections” on page 237
- “Pseudocylindrical Projections” on page 242

Note

The General Cartographic Transformation Package (GCTP) map projections are not described here. Documentation for the GCTP package is available from the US Geologic Survey at `http://mapping.usgs.gov`.

Note

In this text, the plane of the projection is referred to as the *UV* plane with horizontal axis *u* and vertical axis *v*.

Azimuthal Projections

With azimuthal projections, the *UV* plane is tangent to the globe. The point of tangency is projected onto the center of the plane and its latitude and longitude are the points at the center of the map projection, respectively. Rotation is the angle between North and the *v*-axis.

Important characteristics of azimuthal maps include the fact that directions or azimuths are correct from the center of the projection to any other point, and great circles through the center are projected to straight lines on the plane.

The IDL mapping package includes the following azimuthal projections:

- "Orthographic Projection" on page 229
- "Stereographic Projection" on page 229
- "Gnomonic Projection" on page 230
- "Azimuthal Equidistant Projection" on page 231
- "Aitoff Projection" on page 232
- "Lambert's Equal Area Projection" on page 233
- "Hammer-Aitoff Projection" on page 234
- "Satellite Projection" on page 235

Orthographic Projection

The orthographic projection was known by the Egyptians and Greeks 2000 years ago. This projection looks like a globe because it is a perspective projection from infinite distance. As such, it maps one hemisphere of the globe into the *UV* plane. Distortions are greatest along the rim of the hemisphere where distances and land masses are compressed.

The following figure shows an orthographic projection centered over Eastern Spain at a scale of 70 million to 1.

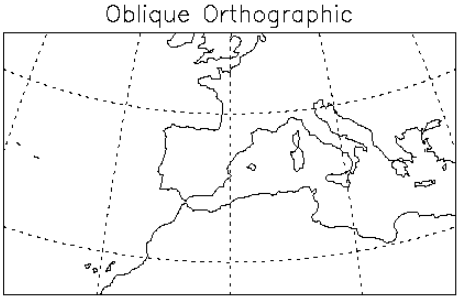


Figure 9-1: Orthographic Projection

Stereographic Projection

The stereographic projection is a true perspective projection with the globe being projected onto the *UV* plane from the point *P* on the globe diametrically opposite to the point of tangency. The whole globe except *P* is mapped onto the *UV* plane. There is great distortion for regions close to *P*, since *P* maps to infinity.

The stereographic projection is the only known perspective projection that is also conformal. It is frequently used for polar maps. For example, a stereographic view of the north pole has the south pole as its point of perspective.

The following figure shows an equatorial stereographic projection with the hemisphere centered on the equator at longitude –105 degrees.

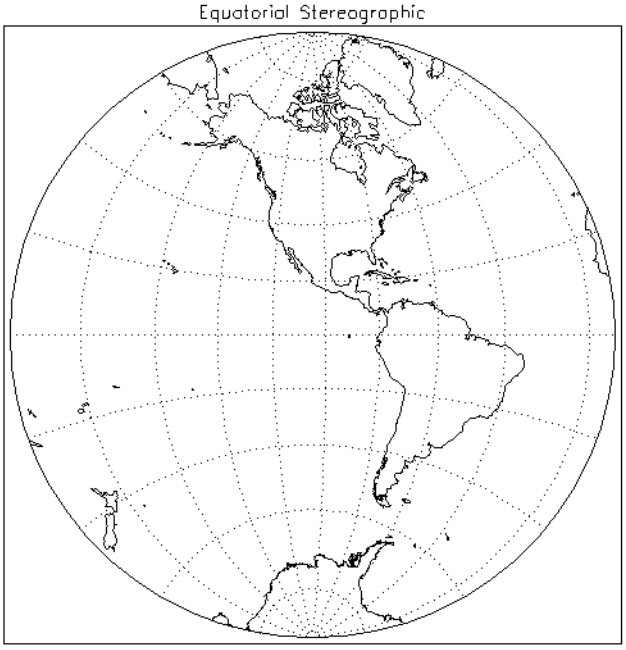


Figure 9-2: An Azimuthal Projection

Gnomonic Projection

The gnomonic projection (also called Central or Gnomic) projects all great circles to straight lines. The gnomonic projection is the perspective, azimuthal projection with point of perspective at the center of the globe. Hence, with the gnomonic projection, the interior of a hemispherical region of the globe is projected to the *UV* plane with the rim of the hemisphere going to infinity. Except at the center, there is great distortion of shape, area, and scale. The default clipping region for the gnomonic projection is a circle with a radius of 60 degrees at the center of projection.

The projection in the following figure is centered around the point at latitude 40 degrees and longitude –105 degrees. The region on the globe that is mapped lies

between 20 degrees and 70 degrees of latitude and –130 degrees and –70 degrees of longitude.

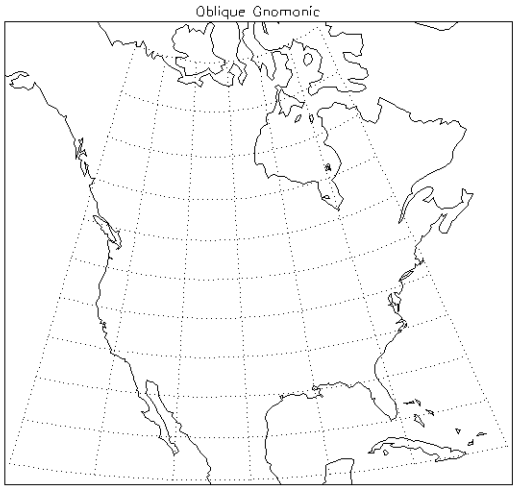


Figure 9-3: A Gnomonic Projection

Azimuthal Equidistant Projection

The azimuthal equidistant projection is also not a true perspective projection, because it preserves correctly the distances between the tangent point and all other points on the globe. Any line drawn through the tangent point reports distance correctly. Therefore, this projection type is useful for determining flight distances. The point *P* opposite the tangent point is mapped to a circle on the *UV* plane, and hence, the whole globe is mapped to the plane. There is infinite distortion close to the outer rim of the map, which is the circular image of *P*.

The following Azimuthal projection is centered at the South Pole and shows the entire globe.

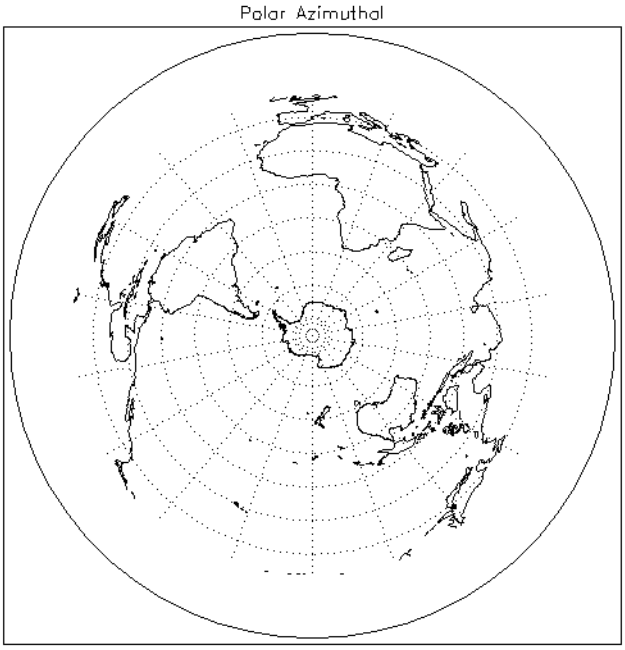


Figure 9-4: An Azimuthal Equidistant Projection

Aitoff Projection

The Aitoff projection modifies the equatorial aspect of one hemisphere of the azimuthal equidistant projection, described above. Lines parallel to the equator are stretched horizontally and meridian values are doubled, thereby displaying the world as an ellipse with axes in a 2:1 ratio. Both the equator and the central meridian are represented at true scale; however, distances measured between the point of tangency and any other point on the map are no longer true to scale.

An Aitoff projection centered on the international dateline is shown in the following figure.

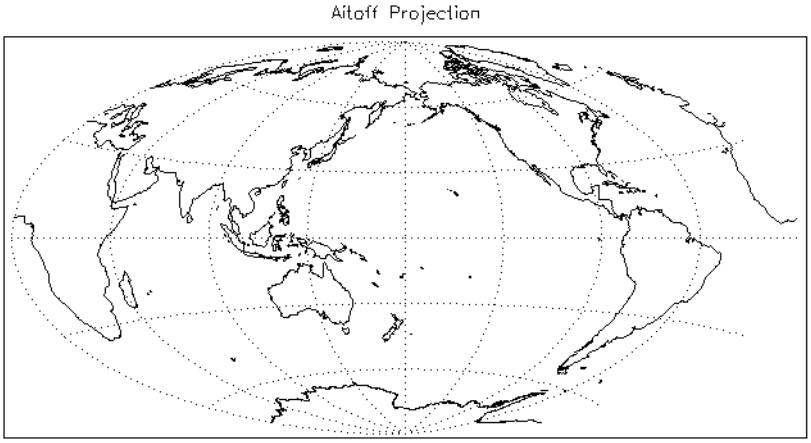


Figure 9-5: An Aitoff Projection

Lambert's Equal Area Projection

Lambert's equal area projection adjusts projected distances in order to preserve area. Hence, it is not a true perspective projection. Like the stereographic projection, it maps to infinity the point *P* diametrically opposite the point of tangency. Note also that to preserve area, distances between points become more contracted as the points become closer to *P*. Lambert's equal area projection has less overall scale variation than the other azimuthal projections.

The following figure shows the Northern Hemisphere rotated counterclockwise 105 degrees, and filled continents.

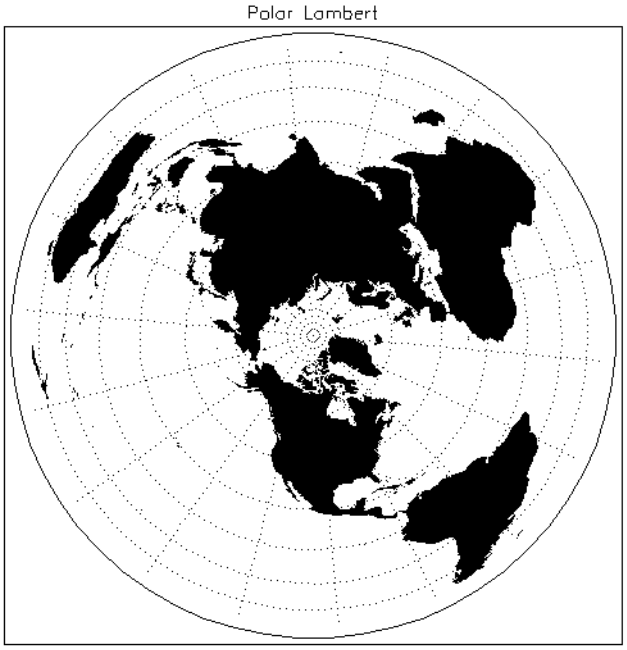


Figure 9-6: A Lambert's Equal Area Projection

Hammer-Aitoff Projection

Although the Hammer-Aitoff projection is not truly azimuthal, it is included in this section because it is derived from the equatorial aspect of Lambert's equal area projection limited to a hemisphere (in the same way Aitoff's projection is derived from the equatorial aspect of the azimuthal equidistant projection). In this derivation, the hemisphere is represented inside an ellipse with the rest of the world in the lunes of the ellipse.

Because the Hammer-Aitoff projection produces an equal area map of the entire globe, it is useful for visual representations of geographically related statistical data and distributions. Astronomers use this projection to show the entire celestial sphere on one map in a way that accurately depicts the relative distribution of the stars in different regions of the sky.

A Hammer-Aitoff projection centered on the international dateline is shown in the following figure:

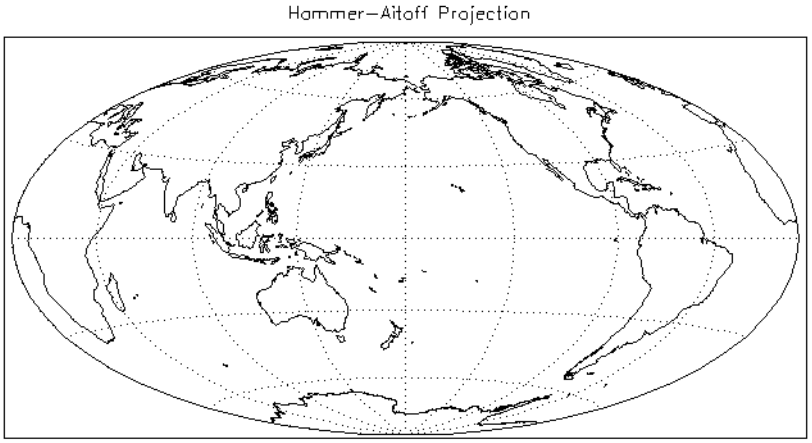


Figure 9-7: The Hammer-Aitoff Projection

Satellite Projection

The satellite projection, also called the General Perspective projection, simulates a view of the globe as seen from a camera in space. If the camera faces the center of the globe, the projection is called a Vertical Perspective projection (note that the orthographic, stereographic, and gnomonic projections are special cases of this projection), otherwise the projection is called a Tilted Perspective projection.

The globe is viewed from a point in space, with the viewing plane touching the surface of the globe at the point directly beneath the satellite (the sub-satellite point). If the projection plane is perpendicular to the line connecting the point of projection and the center of the globe, a Vertical Perspective projection results. Otherwise, the projection plane is horizontally turned Γ degrees clockwise from the north, then tilted ω degrees downward from horizontal.

The map in the accompanying figure shows the eastern seaboard of the United States from an altitude of about 160km, above Newburgh, NY.

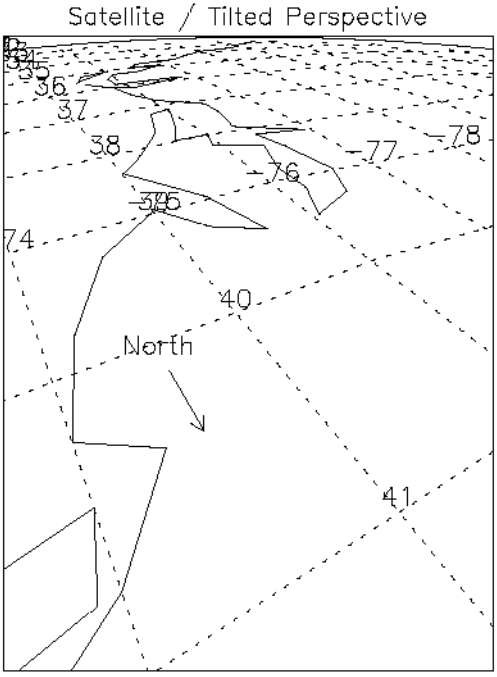


Figure 9-8: Satellite Projection

Cylindrical Projections

A cylindrical projection maps the globe to a cylinder which is formed by wrapping the *UV* plane around the globe with the *u*-axis coinciding with a great circle. The parameters P_{0lat}, P_{0lon}, and *Rot* determine the great circle that passes through the point $C=(P_{0lat}, P_{0lon})$. In the discussions below, this great circle is sometimes referred to as EQ. *Rot* is the angle between North at the map's center and the *v*-axis (which is perpendicular to the great circle). The cylinder is cut along the line parallel to the *v*-axis and passing through the point diametrically opposite to C. It is then rolled out to form a plane.

The cylindrical projections in IDL include: Mercator, Transverse Mercator, cylindrical equidistant, Miller, Lambert's conformal conic, and Alber's equal-area conic.

Mercator Projection

Mercator's projection is partially developed by projecting the globe onto the cylinder from the center of the globe. This is a partial explanation of the projection because vertical distances are subjected to additional transformations to achieve conformity—that is, local preservation of shape. Therefore, uses include navigation maps and equatorial maps. To properly use the projection, the user should be aware that the two points on the globe 90 degrees from the central great circle (e.g., the North and South Poles in the case that the selected great circle is the equator) are mapped to infinite distances. Limits are typically specified because of the great distortions around the poles when the equator is selected.

A simple mercator projection with latitude ranges from –80 degrees to 80 degrees is shown in the following figure.

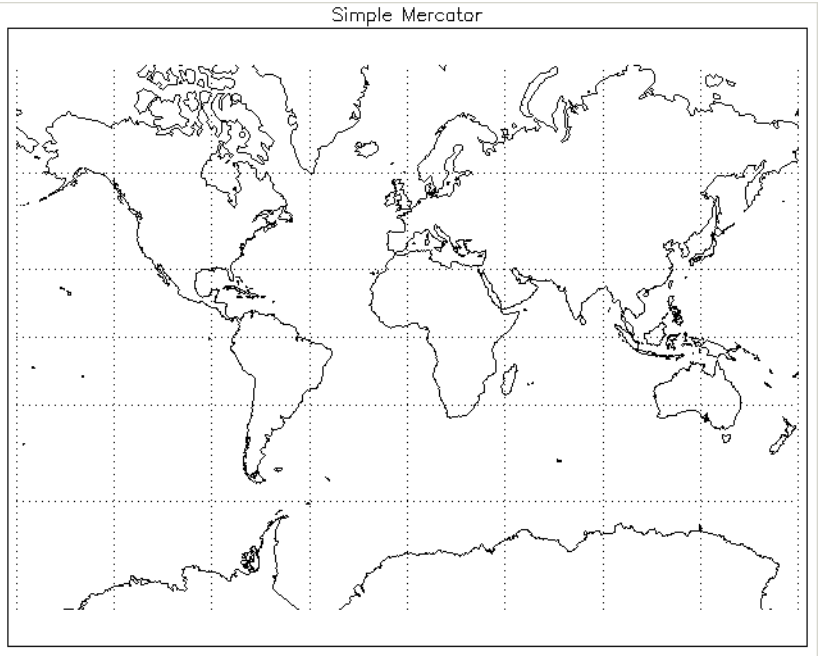


Figure 9-9: Simple Mercator Projection

Transverse Mercator Projection

The Transverse Mercator (also called the *UTM*, and *Gauss-Krueger* in Europe) projection rotates the equator of the Mercator projection 90 degrees so that it follows a specified central meridian. In other words, the Transverse Mercator involves projecting the Earth onto a cylinder which is always in contact with a meridian instead of with the Equator.

The central meridian intersects two meridians and the Equator at right angles; these four lines are straight. All other meridians and parallels are complex curves which are concave toward the central meridian. Shape is true only within small areas and the areas increase in size as they move away from the central meridian. Most other IDL projections are scaled in the range of +/– 1 to +/– 2 Pi; the UV plane of the Transverse Mercator projection is scaled in meters. The conformal nature of this

projection and its use of the meridian makes it useful for north-south regions. The Clarke 1866 ellipsoid is used for the default.

The following Transverse Mercator map shows North and South America, with a central meridian of –90 degrees West and centered on the Equator.

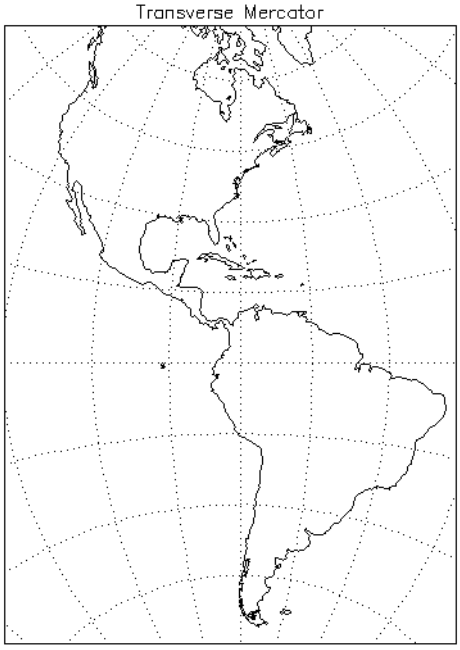


Figure 9-10: Transverse Mercator Projection

Cylindrical Equidistant Projection

The cylindrical equidistant projection is one of the simplest projections to construct. If EQ is the equator, this projection simply lays out horizontal and vertical distances on the cylinder to coincide numerically with their measurements in latitudes and longitudes on the sphere. Hence, the equidistant cylindrical projection maps the entire globe to a rectangular region bounded by

$$-180 \leq u \leq 180$$

and

$$-90 \leq v \leq 90$$

If EQ is the equator, meridians and parallels will be equally spaced parallel lines.

The following figure shows a simple cylindrical equidistant projection and an oblique cylindrical equidistant projection rotated by 45°.

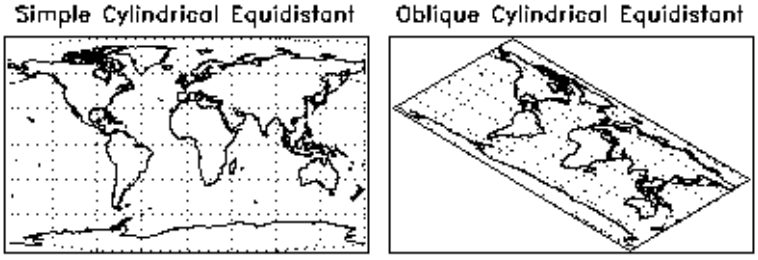


Figure 9-11: Cylindrical Projections

Miller Cylindrical Projection

The Miller projection is a simple mathematical modification of the Mercator projection, incorporating some aspects of cylindrical projections. It is not equal-area, conformal or equidistant along the meridians. Meridians are equidistant from each other, but latitude parallels are spaced farther apart as they move away from the Equator, thereby keeping shape and area distortion to a minimum. The meridians and parallels intersect each other at right angles, with the poles shown as straight lines. The Equator is the only line shown true to scale and free of distortion.

Conic Projection

The Lambert's conformal conic with two standard parallels is constructed by projecting the globe onto a cone passing through two parallels. Additional scaling achieves conformity. The pole under the cone's apex is transformed to a point, and the other pole is mapped to infinity. The scale is correct along the two standard parallels. Parallels can be specified and are projected onto circles and meridians onto equally spaced straight lines. The following figure shows the map shown in the

accompanying figure, which features North America with standard parallels at 20 degrees and 60 degrees.

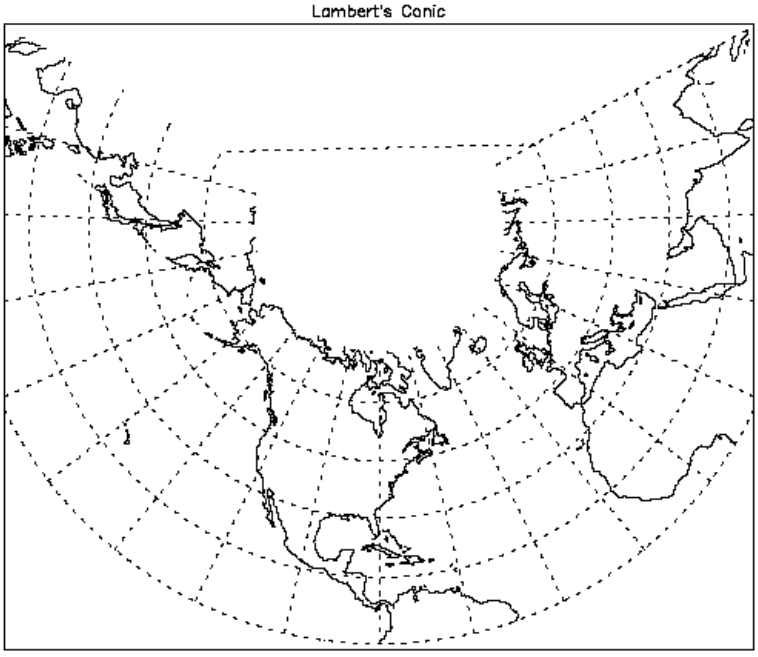


Figure 9-12: Lambert's Conformal Conic with Standard Parallels at 20° and 60°

Albers Equal-Area Conic Projection

The Albers Equal-Area Conic is like most other conics in that meridians are equally spaced radii, parallels are concentric arcs of circles and scale is constant along any parallel. To maintain equal area, the scale factor along meridians is the reciprocal of the scale factor along parallels, with the scale along the parallels between the two standard parallels too small, and the scale beyond the standard parallels too large. Standard parallels are correct in scale along the parallel, as well as in every direction.

The Albers projection is particularly useful for predominantly east-west regions. Any keywords for the Lambert conformal conic also apply to the Albers conic.

Pseudocylindrical Projections

Pseudocylindrical projections are distinguished by the fact that in their simplest form, lines of latitude are parallel straight lines and meridians are curved lines.

Robinson Cylindrical

This pseudocylindrical projection was designed by Arthur Robinson in 1963 for Rand McNally. It is suitable for World maps and is a compromise to best fulfill a number of conflicting requirements, including an uninterrupted format, minimal shearing, minimal apparent area-scale distortion for major continents, and simplicity. It was designed to make the world look right. Since its introduction, it has been adopted by the National Geographic Society for many of their world maps.

Each individual parallel is equally divided by the meridians. The poles are represented by lines rather than points to avoid compressing the northern land masses. The central meridian should always be 0 degrees longitude to retain the correct balance of shapes, sizes, and relative positions.

The following figure shows a Robinson projection.

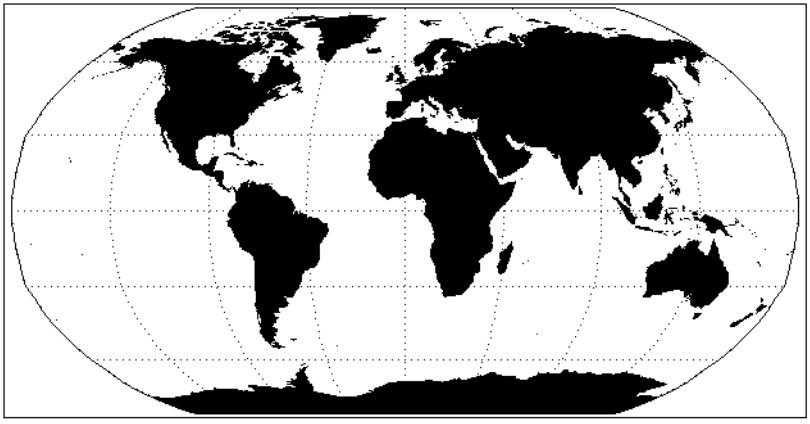


Figure 9-13: Robinson Projection

Sinusoidal Projection

With the sinusoidal projection, the central meridian is a straight line and all other meridians are equally spaced sinusoidal curves. The scaling is true along the central meridian as well as along all parallels.

The sinusoidal projection is one of the easiest projections to construct. The formulas below from Snyder (1987) give the relationship between the latitude ϕ and longitude λ of a point on the globe and its image on the *UV* plane.

$$u = \lambda \cos\phi$$

$$v = \phi$$

The following shows the sinusoidal map of the whole globe centered at longitude 0 degrees and latitude 0 degrees.

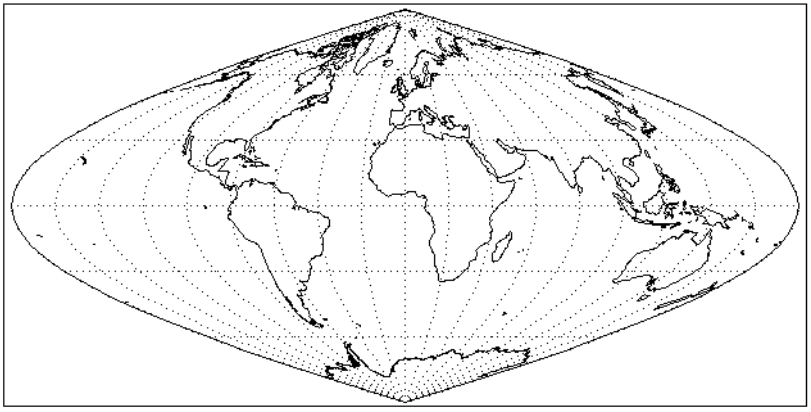


Figure 9-14: Sinusoidal Projection

Mollweide Projection

With the Mollweide projection, the central meridian is a straight line, the meridians 90 degrees from the central meridian are circular arcs and all other meridians are elliptical arcs. The Mollweide projection maps the entire globe onto an ellipse in the *UV* plane. The circular arcs encompass a hemisphere and the rest of the globe is contained in the lunes on either side.

The following figure shows a Mollweide projection in oblique form.

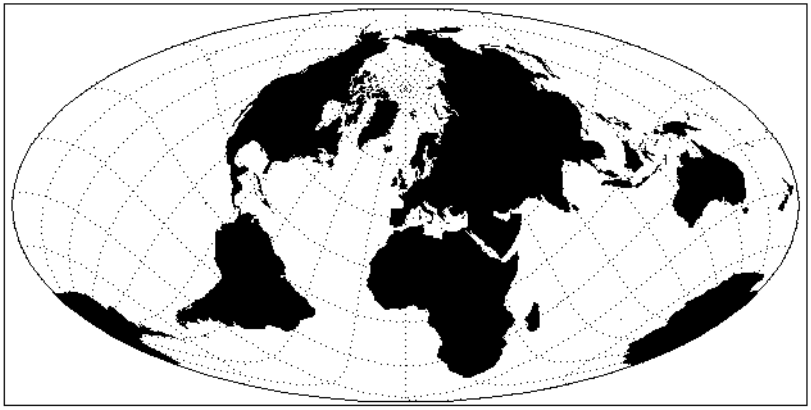


Figure 9-15: Mollweide Projection

Since the center of the projection is not on the equator, parallels of latitude are not straight lines, just as they are not straight lines with an oblique Mercator or cylindrical equidistant projection.

Goode’s Homolosine Projection

The Goode interrupted Homolosine projection, developed by J. Paul Goode, in 1923, is designed for World maps to show the continents with minimal scale and shape distortion. This is accomplished by interrupting the projection and choosing several central meridians to coincide with large land masses. This projection is a fusion of the Sinusoidal projection between the latitudes of 44.7 degrees North and South, and the Mollweide projection between these parallels and the poles.

The following figure shows an example of Goode's Homolosine projection.

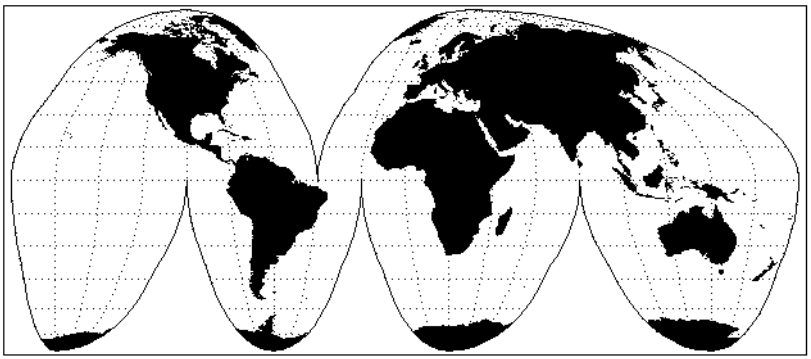


Figure 9-16: Goode's Homolosine Projection

High-Resolution Continent Outlines

IDL supports two different datasets that contain continent outlines and other geographical and political boundaries. The default data set is a low-resolution continental outline database that is automatically installed when you install IDL. The high-resolution database was adapted from the 1993 CIA World Map database by Thomas Oetli of the Swiss Meteorological Institute. The high-resolution outlines are found in an optional data set that may not have been installed when your copy of IDL was first installed.

To access the high-resolution data set, simply set the HIRES keyword when calling MAP_CONTINENTS with the COASTS, COUNTRIES, FILL_CONTINENTS, or RIVERS keywords. You can also get high-resolution continent boundaries by calling MAP_SET with the HIRES and CONTINENTS keywords set. See MAP_CONTINENTS in the *IDL Reference Guide* for an example of using the high-resolution outlines.

Resolution of Map Databases

Data points in the CIA World Map database are approximately one kilometer apart. Note, however, that in the case of the coast and river databases, actual distances between the data points may be much smaller because of convolutions in the coastline or riverbed.

Data points in the low-resolution map database are either a subset of the high-resolution database (rivers and country boundaries) or are based on the continental map database used in previous versions of IDL (the file `supmap.dat` in the `resource/maps` subdirectory of the IDL distribution). Data points in the low-resolution database are approximately 10 kilometers apart.

Neither of the map databases is intended for high-precision work.

The following table compares the low-resolution and high-resolution map databases:

Feature	Low-Resolution	High-Resolution
Coastlines, islands, and lakes (including continental outlines)	Data in file `supmap.dat`.	Entire CIA World Map

Table 9-1: Comparison of Low- and High-resolution Map Databases

Feature	Low-Resolution	High-Resolution
Continental polygons	Data extracted from `supmap.dat`.	Every 20th point of CIA World Map.
Rivers	Every 250th point of the CIA World Map.	Entire CIA World Map.
National boundaries	Every 100th point of CIA World Map.	Entire CIA World Map.

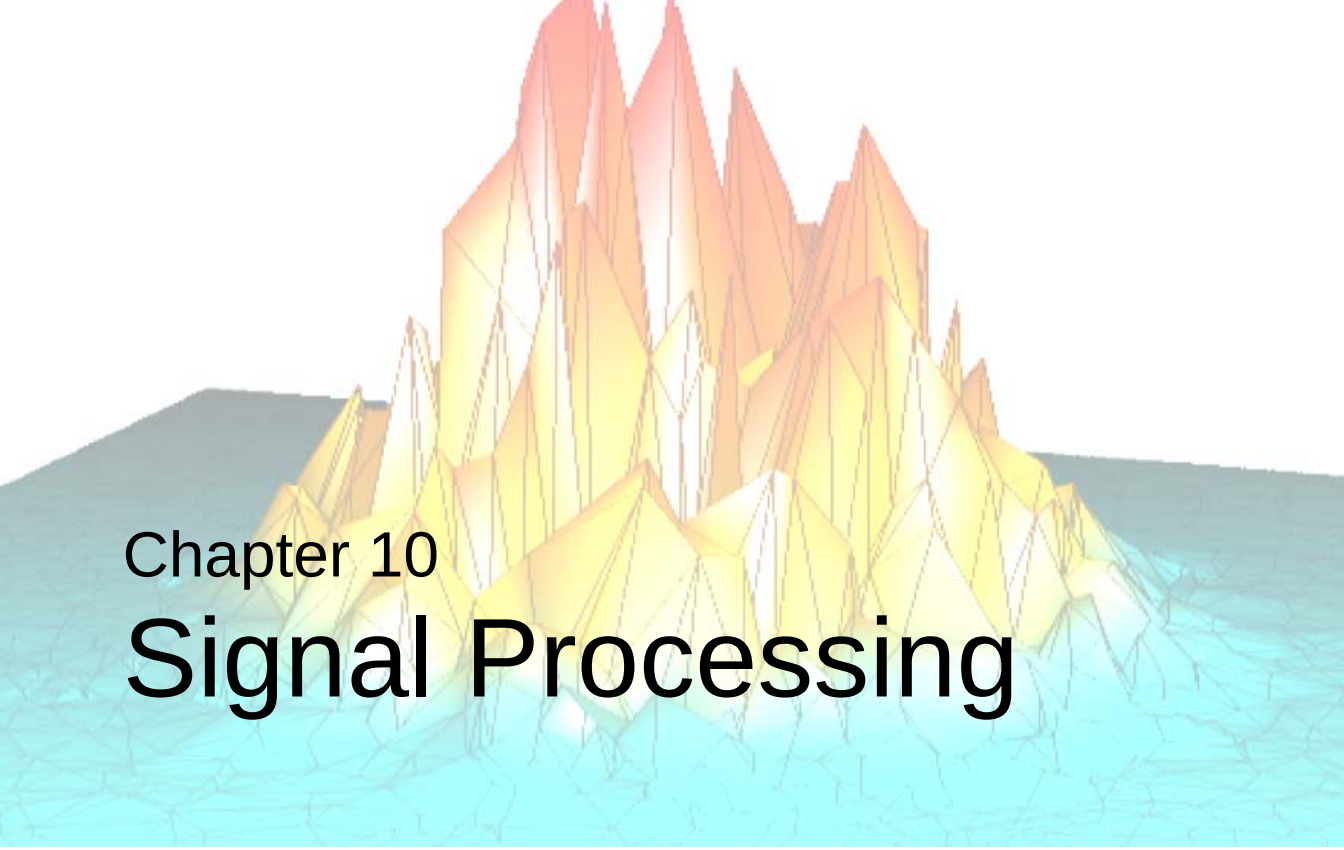
Table 9-1: Comparison of Low- and High-resolution Map Databases

References

Greenwood, David (1964), *Mapping*, University of Chicago Press, Chicago.

Pearson, Frederick II (1990), *Map Projections: Theory and Applications*, CRC Press, Inc., Boca Raton.

Snyder, John P. (1987), *Map Projections—A Working Manual*, U.S. Geological Survey Professional Paper 1395, U.S.Government Printing Office, Washington, D.C.



Chapter 10
Signal Processing

The following topics are covered in this chapter:

Overview of Signal Processing 250
Digital Signals 251
Signal Analysis Transforms 253
The Fourier Transform 254
Interpreting FFT Results 255
Displaying FFT Results 256
Using Windows 260
Aliasing 263
FFT Algorithm Details 264
The Hilbert Transform 265
The Wavelet Transform 267
Convolution 268
Correlation and Covariance 269
Digital Filtering 270
Finite Impulse Response (FIR) Filters . . . 271
FIR Filter Implementation 273
Infinite Impulse Response Filters 275
Routines for Signal Processing 250
References 278

Overview of Signal Processing

A signal, by definition, contains information. Any signal obtained from a physical process also contains noise. It is often difficult or impossible to make sense of the information contained in a digital signal by looking at it in its raw form—that is, as a sequence of real values at discrete points in time. Signal analysis transforms offer natural, meaningful, alternate representations of the information contained in a signal.

This chapter describes IDL's digital signal processing tools. Most of the procedures and functions mentioned here work in two or more dimensions. For simplicity, only one dimensional signals are used in the examples.

Routines for Signal Processing

For a list of IDL signal processing routines, see the functional category of "Signal Processing" in the *IDL Quick Reference* manual. There you will find a brief introduction to the routines. More detailed information is available in the *IDL Reference Guide*.

Running the Example Code

The examples in this chapter are written to take advantage of iTools. The example code is part of the IDL distribution. All of the files mentioned are located in the `examples/doc/signal` subdirectory of the IDL distribution. By default, this directory is part of IDL's path; if you have not changed your path, you will be able to run the examples as described here. See "!PATH" in Appendix D of the *IDL Reference Guide* manual for information on IDL's path.

Digital Signals

A one-dimensional digital signal is a sequence of data, represented as a vector in an array-oriented language like IDL. The term digital actually describes two different properties:

1. The signal is defined only at discrete points in time as a result of sampling, or because the instrument which measured the signal is inherently discrete-time in nature. Usually, the time interval between measurements is constant.
2. The signal can take on only discrete values.

In this discussion, we assume that the signal is sampled at a time interval. The concepts and techniques presented here apply equally well to any type of signal—the independent variable may represent time, space, or any abstract quantity.

The following IDL commands create a simulated digital signal $u(k)$, sampled at an interval `delt`. This simulated signal will be used in examples throughout this chapter. The simulated signal contains 1024 time samples, with a sampling interval of 0.02 seconds. The signal contains a DC component and components at 2.8, 6.5, and 11.0 cycles per second.

Enter the following commands at the IDL prompt to create the simulated signal:

```
N = 1024 ; number of samples
delt = 0.02 ; sampling interval

; Simulated signal.
u = -0.3 $
   + 1.0 * SIN(2 * !PI * 2.8 * delt * FINDGEN(N)) $
   + 1.0 * SIN(2 * !PI * 6.25 * delt * FINDGEN(N)) $
   + 1.0 * SIN(2 * !PI * 11.0 * delt * FINDGEN(N))
```

Example Code

Alternately, type `@sigprc01` at the IDL prompt to run the `sigprc01`batch file that creates the signal. See “Running the Example Code” on page 250 if IDL does not find the batch file.

Because the signal is digital, the conventional way to display it is with a histogram (or step) plot. To create a histogram plot, set the PSYM keyword to the PLOT routine equal to 10. A section of the example signal *u*(*k*) is plotted in the figure below.

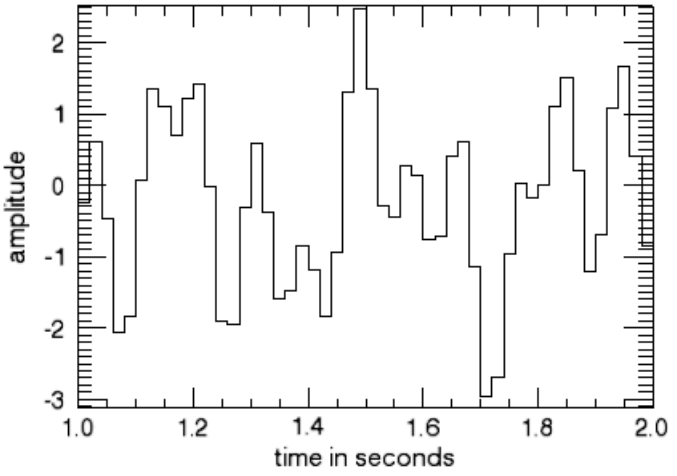


Figure 10-1: Histogram Plot of Sample Signal u(k)

Note

When the number of sampled data points is large, the steps in the histogram plot are too small to see. In such cases you should not plot in histogram mode.

Example Code

Type `@sigprc02` at the IDL prompt to run the batch file that creates this display. The source code is located in `sigprc02`, in the `examples/doc/signal` directory. See "Running the Example Code" on page 250 if IDL does not find the batch file.

Signal Analysis Transforms

Most signals can be decomposed into a sum of discrete (usually sinusoidal) signal components.The result of such decomposition is a frequency spectrum that can uniquely identify the signal. IDL provides three transforms to decompose a signal and prepare it for analysis: the Fourier transform, the Hilbert transform, and the wavelet transform.

The Fourier Transform

The Discrete Fourier Transform (DFT) is the most widely used method for determining the frequency spectra of digital signals. This is due to the development of an efficient algorithm for computing DFTs known as the Fast Fourier Transform (FFT).

The discrete Fourier transform, *v*(*m*), of an *N*-element, one-dimensional function, *u*(*k*), is defined as:

$$v(m) = \frac{1}{N} \sum_{k=0}^{N-1} u(k) \exp[-j2\pi mk/N]$$

The inverse transform is defined as:

$$u(k) = \sum_{m=0}^{N-1} v(m) \exp[j2\pi mk/N]$$

IDL implements the Fast Fourier Transform in the FFT function. You can find details on using IDL's FFT function in the following sections and in "FFT" in the *IDL Reference Guide* manual.

Interpreting FFT Results

Just as the sampled time data represents the value of a signal at discrete points in time, the result of a (forward) Fast Fourier Transform represents the spectrum of the signal at discrete frequencies. These discrete frequencies are a function of the frequency index (*m*), the number of samples collected (*N*), and the sampling interval (δ):

$$f(m) = \frac{m}{N\delta}$$

The frequencies for which the FFT of a sampled signal are defined are sometimes called frequency bins, which refers to the histogram-like nature of a discrete spectrum. The width of each frequency bin is $1/(N * \delta)$.

Due to the complex exponential in the definition of the DFT, the spectrum has a cyclic dependence on the frequency index *m*. That is:

$$v(m + pN) = v(m)$$

for *p* = any integer.

The frequency spectrum computed by IDL's FFT function for a one-dimensional time sequence is stored in a vector with indices running from 0 to *N*–1, which is also a valid range for the frequency index *m*. However, the frequencies associated with frequency indices greater than *N*/2 are above the Nyquist frequency and are not physically meaningful for sampled signals. Many textbooks choose to define the range of the frequency index *m* to be from – (*N*/2 – 1) to *N*/2 so that it is (nearly) centered around zero. From the cyclic relation above with *p* = –1:

$v(-(N/2 - 1)) = v(N/2 + 1 - N) = v(N/2 + 1)$

$v(-(N/2 - 2)) = v(N/2 + 2 - N) = v(N/2 + 2)$

...

$v(-2) = v(N - 2 - N) = v(N - 2)$

$v(-1) = v(N - 1 - N) = v(N - 1)$

This index shift is easily accomplished in IDL with the SHIFT function. See "Real and Imaginary Components" on page 256 for an example.

Displaying FFT Results

Depending on the application, there are many ways to display spectral data, the result of the (forward) FFT function.

Real and Imaginary Components

The most direct way is to plot the real and imaginary parts of the spectrum as a function of frequency index or as a function of the corresponding frequencies. The following figure displays the real and imaginary parts of the spectrum $v(m)$ of the sampled signal $u(k)$ for frequencies from $-(N/2 - 1)/(N * \delta)$ to $(N/2)/(N * \delta)$ cycles per second.

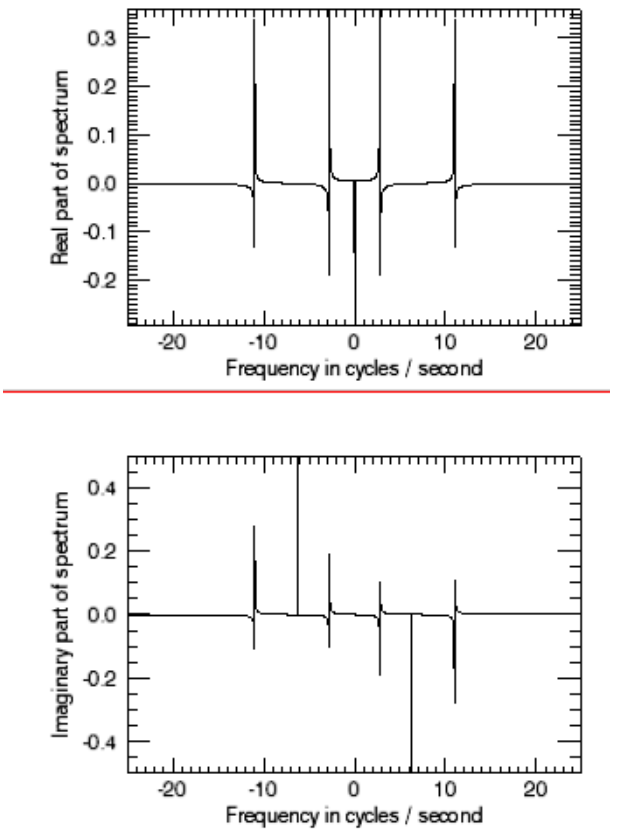


Figure 10-2: Real and Imaginary Parts of the Sample Signal

Example Code

Type `@sigprc03` at the IDL prompt to run the batch file that creates this display. The source code is located in `sigprc03`, in the `examples/doc/signal` directory. See "Running the Example Code" on page 250 if IDL does not find the batch file.

IDL's FFT function always returns a single- or double-precision complex array with the same dimensions as the input argument. In the case of a forward FFT performed on a one-dimensional vector of *N* real values, the result is an *N*-element vector of complex quantities, which takes 2*N* real values to represent. It would seem that there is twice as much information in the spectral data as there is in the time sequence data. This is not the case. For a real valued time sequence, half of the information in the frequency sequence is redundant. Specifically:

```
; 1 redundant value:
IMAGINARY(v(0)) = 0.0
; 1 redundant value:
IMAGINARY(v(N/2)) = 0.0
```

and

```
; for m=1 to N/2-1, N-2 redundant values:
v(N-m) = CONJ(v(m))
```

so that exactly *N* of the single- or double-precision values used to represent the frequency spectrum are redundant. This redundancy is evident in the previous figure. Notice that the real part of the spectrum is an even function (symmetric about zero), and the imaginary part of the spectrum is an odd function (anti-symmetric about zero). This is always the case for the spectra of real-valued time sequences.

Because of the redundancy in such spectra, it is common to display only half of the spectrum of a real time sequence. That is, only the spectral values with frequency indices from 0 to *N*/2, which correspond to frequencies from 0 to 1/(2 * δ), the Nyquist frequency. This vector of positive frequencies is generated in IDL with the following command:

```
; f = [0.0, 1.0/(N*delt), ... , 1.0/(2.0*delt)]
F = FINDGEN(N/2+1)/(N*delt)
```

Magnitude and Phase

It is also common to display the magnitude and phase of the spectrum, which have physical significance, as opposed to the real and imaginary parts of the spectrum, which do not have physical significance. Since there is a one-to-one correspondence between a complex number and its magnitude and phase, no information is lost in the transformation from a complex spectrum to its magnitude and phase. In IDL, the

magnitude is easily determined with the absolute value (ABS) function, and the phase with the arc-tangent (ATAN) function. By one widely used convention, the magnitude of the spectrum is plotted in decibels (dB) and the phase is plotted in degrees, against frequency on a logarithmic scale. The magnitude and phase of our sample signal are plotted in the same data space, shown in the figure below.

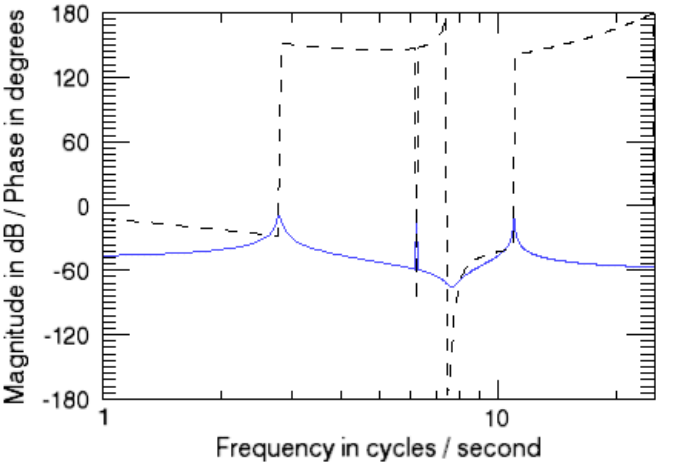


Figure 10-3: Magnitude (Solid LIne) and Phase (Dashed Line) of the Sample Signal

Example Code

Type `@sigprc04` at the IDL prompt to run the batch file that creates this display. The source code is located in `sigprc04`, in the `examples/doc/signal` directory. See "Running the Example Code" on page 250 if IDL does not find the batch file.

Using a logarithmic scale for the frequency axis has the advantage of spreading out the lower frequencies, while higher frequencies are crowded together. Note that the spectrum at zero frequency (DC) is lost completely on a semi-logarithmic plot.

The previous figure shows the strong frequency components at 2.8, 6.25, and 11.0 cycles/second as peaks in the magnitude plot, and as discontinuities in the phase plot. The magnitude peak at 6.25 cycles/second is a narrow spike, as would be expected from the pure sine wave component at that frequency in the time data sequence. The peaks at 2.8 and 11.0 cycles/second are more spread out, due to an effect known as smearing or leakage. This effect is a direct result of the definition of the DFT and is not due to any inaccuracy in the FFT. Smearing is reduced by increasing the length of

the time sequence, or by choosing a sample size which includes an integral number of cycles of the frequency component of interest. There are an integral number of cycles of the 6.25 cycles/second component in the time sequence used for this example, which is why the peak at that frequency is sharper.

The apparent discontinuity in the phase plot at around 7.45 cycles/second is an anomaly known as phase wrapping. It is a result of resolving the phase from the real and imaginary parts of the spectrum with the arctangent function (ATAN), which returns principal values between –180 and +180 degrees.

Power Spectrum

Finally, for many applications, the phase information is not useful. For these, it is often customary to plot the power spectrum, which is the square of the magnitude of the complex spectrum. The resulting plot is shown in the figure below.

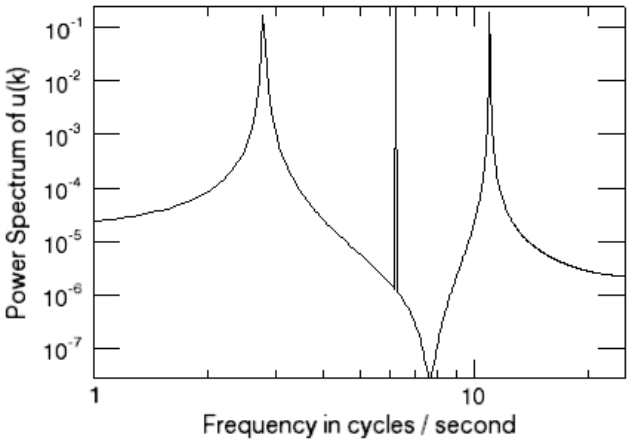


Figure 10-4: Power Spectrum of the Sample Signal

Example Code

Type `@sigprc05` at the IDL prompt to run the batch file that creates this display. The source code is located in `sigprc05`, in the `examples/doc/signal` directory. See "Running the Example Code" on page 250 if IDL does not find the batch file.

Using Windows

The smearing or leakage effect mentioned previously is a direct consequence of the definition of the Discrete Fourier Transform and of the fact that a finite time sample of a signal often does not include an integral number of some of the frequency components in the signal. The effect of this truncation can be reduced by increasing the length of the time sequence or by employing a windowing algorithm. IDL's HANNING function computes two windows which are widely used in signal processing: the Hanning window and the Hamming window.

Hanning Window

The Hanning window is defined as:

$$w(k) = \frac{1}{2}\left(1 - \cos\left(\frac{2\pi k}{N}\right)\right)$$

The resulting vector is multiplied element-by-element with the sampled signal vector before applying the FFT. For example, the following IDL command computes the Hanning window and then applies the FFT function:

```
v_n = FFT(HANNING(N)*U)
```

The power spectrum of the Hanning windowed signal shows the mitigation of the truncation effect (see the figure below).

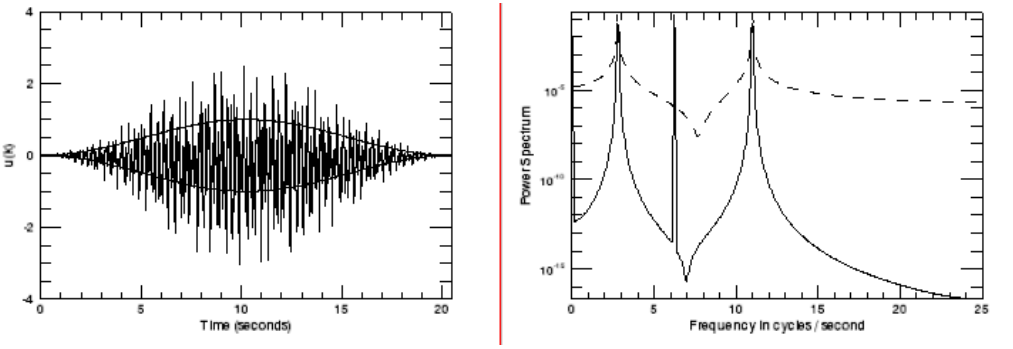


Figure 10-5: Time Series Multiplied by Hanning Window (Left) and Power Spectrum (Right) with Hanning Window (Solid) and without (Dashed)

Example Code

Type `@sigprc06` at the IDL prompt to run the batch file that creates this display. The source code is located in `sigprc06`, in the `examples/doc/signal` directory. See "Running the Example Code" on page 250 if IDL does not find the batch file.

Hamming Window

The Hamming window is defined as:

$$w(k) = 0.54 - 0.46\cos\left(\frac{2\pi k}{N}\right)$$

The resulting vector is multiplied element-by-element with the sampled signal vector before applying the FFT. For example, the following IDL command computes the Hamming window and then applies the FFT function:

```
v_m = FFT(HANNING(N, ALPHA=0.56)*U)
```

The power spectrum of the Hamming windowed signal shows the mitigation of the truncation effect (see the figure below).

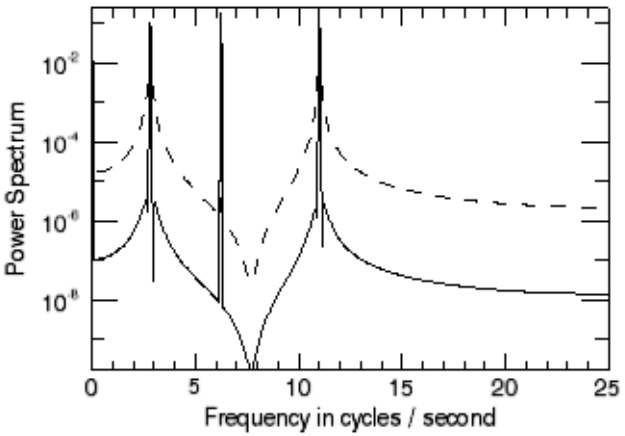


Figure 10-6: Power Spectrum with Hamming Window (Solid) and without (Dashed)

Example Code

Type `@sigprc07` at the IDL prompt to run the batch file that creates this display. The source code is located in `sigprc07`, in the `examples/doc/signal` directory. See "Running the Example Code" on page 250 if IDL does not find the batch file.

Aliasing

Aliasing is a well known phenomenon in sampled data analysis. It occurs when the signal being sampled has components at frequencies higher than the Nyquist frequency, which is equal to half the sampling frequency. Aliasing is a consequence of the fact that after sampling, every periodic signal at a frequency greater than the Nyquist frequency looks exactly like some other periodic signal at a frequency less than the Nyquist frequency. For example, suppose we add a 30 cycle per second periodic component to our sampled data sequence $u(t)$. The power spectrum of the augmented signal appears below.

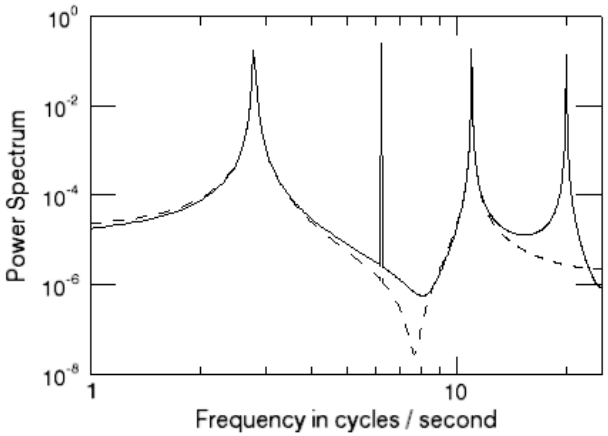


Figure 10-7: Power Spectrum of the Sample Signal After Adding a 30 Cycles per Second Component

Because the frequency of the new component is above the Nyquist frequency of 25 cycles per second (25 = 1/(2*delt)), the power spectrum shows the contribution of the new component as an alias at 20 cycles per second. To prevent aliasing, frequency components of a signal above the Nyquist frequency must be removed before sampling.

Example Code

Type `@sigprc08` at the IDL prompt to run the batch file that creates this display. The source code is located in `sigprc08`, in the `examples/doc/signal` directory. See "Running the Example Code" on page 250 if IDL does not find the batch file.

FFT Algorithm Details

IDL's implementation of the fast Fourier transform is based on the Cooley-Tukey algorithm. The algorithm takes advantage of the fact that the discrete Fourier transform (DFT) of a discrete time series with an even number of points is equal to the sum of two DFTs, each half the length of the original. For data lengths that are a power of 2, this algorithm is used recursively, each iteration subdividing the data into smaller sets to be transformed. In the IDL FFT, this method is also extended to powers of 3 and 5. If the number of points in the original time series does not contain powers of 2, 3, or 5, the original data are still subdivided into data sets with lengths equal to the prime factors of *N*. The resulting subdivisions with lengths equal to prime numbers other than 2, 3, or 5 must be transformed using a slow DFT. The slow DFT is mathematically equivalent to the FFT, but requires N^2 operations instead of $N\log2(N)$.

This implementation means that the FFT function is fastest when the number of points is rich in powers of 2, 3, or 5. The slowest case is when the number of samples is a large prime number. In this case, a significant improvement in efficiency can be gained by padding the data set with zeros to increase the number of data points to a power of 2, 3, or 5.

For real input data of even lengths, the FFT algorithm also takes advantage of the fact that the real array can be packed into a complex array of half the length, and unpacked at the end, thus cutting the running time in half.

The Hilbert Transform

The Hilbert transform is a time-domain to time-domain transformation which shifts the phase of a signal by 90 degrees. Positive frequency components are shifted by +90 degrees, and negative frequency components are shifted by – 90 degrees. Applying a Hilbert transform to a signal twice in succession shifts the phases of all of the components by 180 degrees, and so produces the negative of the original signal. IDL's HILBERT function accepts both real and complex valued signals as inputs; the imaginary part of the result is zero for real inputs.

In optics and signal analysis, the Hilbert transform of the time signal $r(t)$ is known as the quadrature function of $r(t)$, which is used to form a complex function known as the analytic signal. The analytic signal is defined as:

$$\hat{r}(t) = r(t) - jH(r(t))$$

where j is the square root of –1 and H is the Hilbert function.

The projection of the analytic signal onto the plane defined by the real axis and the time axis is the original signal. The projection onto the plane defined by the imaginary axis and the time axis is the Hilbert transform of the original signal.

The following example plots the complex analytic signal of a periodic time signal with a slowly varying amplitude.

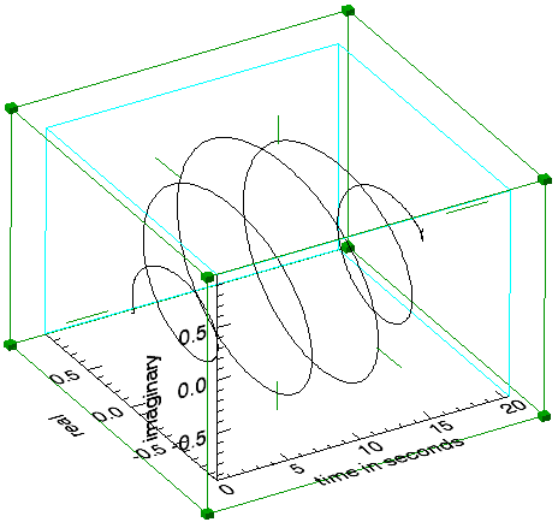


Figure 10-8: Analytic Signal for r(t)

Example Code

Type `@sigprc09` at the IDL prompt to run the batch file that creates this display. The source code is located in `sigprc09`, in the `examples/doc/signal` directory. See "Running the Example Code" on page 250 if IDL does not find the batch file.

The Wavelet Transform

Like the discrete Fourier transform, the discrete wavelet transform (DWT) is a linear operation that defines a forward and inverse relationship between the time-domain and the frequency-domain, also called the wavelet domain. This relationship is expressed through the use of basis functions. In the case of the DFT, trigonometric sines and cosines of varying angles are used. In the case of the DWT, the basis functions are more complicated and usually called mother functions or wavelets. Also like the DFT, the DWT is orthogonal, making many operations computationally efficient. For example, the inverse wavelet transform, when viewed as a matrix operator, is simply the transpose of the forward transform.

Most of the usefulness of wavelets relies on the fact that wavelet transforms can usefully be severely truncated—that is, they can be effectively turned into sparse expressions. This property is a result of the simultaneous compact representation of the wavelet basis functions in the time and frequency domains. See "WTN" in the *IDL Reference Guide* manual for an example using the wavelet transform. Also see "Wavelet Toolkit" in the *IDL Quick Reference* manual for a brief description of the available wavelet routines.

Convolution

Discrete convolution in digital signal processing is used (among other things) to smooth sampled signals using a weighted moving average. It also has many applications outside of signal processing.

IDL has two functions for doing discrete convolution: BLK_CON and CONVOL. BLK_CON takes advantage of the fact that the convolution of two signals is the Inverse Fourier transform of the product of the Fourier transforms of the two signals. BLK_CON is faster than CONVOL, but not as flexible. Among the many applications for discrete convolution is the implementation of digital filters. See the example in the “Finite Impulse Response (FIR) Filters” on page 271.

Correlation and Covariance

Correlation and covariance (which is correlation with any non-zero mean values of the signals removed beforehand) are closely related to convolution. They are useful in analyzing signals with random components. Autocorrelation and autocovariance of signals are computed with the A_CORRELATE function, and crosscorrelation and crosscovariance are computed with the C_CORRELATE function. See “Time-Series Analysis” on page 316 for details.

Digital Filtering

Digital filters can be implemented on a computer to remove unwanted frequency components (noise) from a sampled signal. Two broad classes of filters are Finite Impulse Response (FIR) or Moving Average (MA) filters, and Infinite Impulse Response (IIR) or AutoRegressive Moving Average (ARMA) filters. Both of these classes of filters are described in the following sections:

- "Finite Impulse Response (FIR) Filters" on page 271
- "Infinite Impulse Response Filters" on page 275

Finite Impulse Response (FIR) Filters

Digital filters that have an impulse response which reaches zero in a finite number of steps are (appropriately enough) called Finite Impulse Response (FIR) filters. An FIR filter can be implemented non-recursively by convolving its impulse response (which is often used to define an FIR filter) with the time data sequence it is filtering. FIR filters are somewhat simpler than Infinite Impulse Response (IIR) filters, which contain one or more feedback terms and must be implemented with difference equations or some other recursive technique.

IDL's DIGITAL_FILTER function computes the impulse response of an FIR filter based on Kaiser's window, which in turn is based on the modified Bessel function. The Kaiser filter is "nearly optimum in the sense of having the largest energy in the mainlobe for a given peak sidelobe level" [Jackson, Leland B., *Digital Filters and Signal Processing*]. The DIGITAL_FILTER function constructs lowpass, highpass, bandpass, or bandstop filters. The figure below plots a bandstop filter which suppresses frequencies between 7 cycles per second and 15 cycles per second for data sampled every 0.02 seconds.

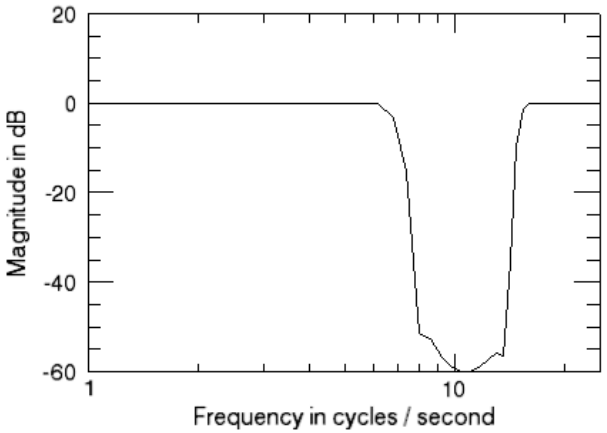


Figure 10-9: Bandstop FIR Filter

Example Code

Type `@sigprc10` at the IDL prompt to run the batch file that creates this display. The source code is located in `sigprc10`, in the `examples/doc/signal` directory. See "Running the Example Code" on page 250 if IDL does not find the batch file.

Other FIR filters can be designed based on the Hanning and Hamming windows (see "Using Windows" on page 260), or any other user-defined window function. The design procedure is simple:

1. Compute the impulse response of an ideal filter using the inverse FFT.
2. Apply a window to the impulse response. The modified impulse response defines the FIR filter.

The figure below shows the plot using the same sampling period and frequency cutoffs as above, and the corresponding ideal filter is constructed in the frequency domain using the Hanning window.

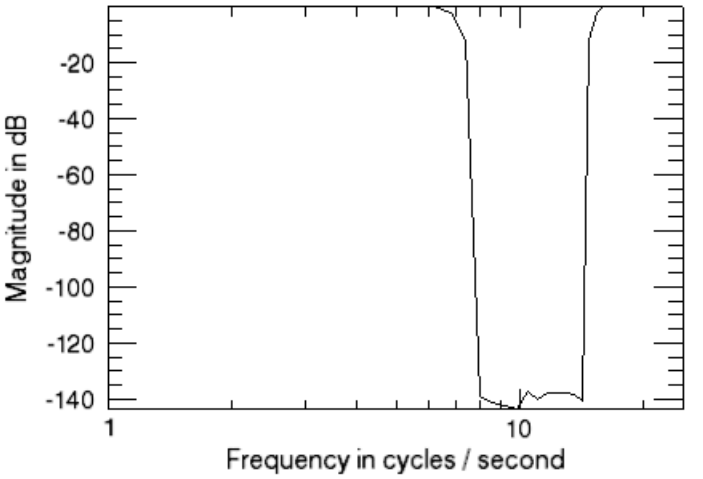


Figure 10-10: Bandstop Filter Using Hanning Window

Example Code

Type `@sigprc11` at the IDL prompt to run the batch file that creates this display. The source code is located in `sigprc11`, in the `examples/doc/signal` directory. See "Running the Example Code" on page 250 if IDL does not find the batch file.

FIR Filter Implementation

The simplest FIR (Finite Impulse Response) filter to apply to a signal is the rectangular or boxcar filter, which is implemented with IDL's SMOOTH function, or the closely related MEDIAN function.

Applying other FIR filters to signals is straightforward since the filter is non-recursive. The filtered signal is simply the convolution of the impulse response of the filter with the original signal. The impulse response of the filter is computed with the DIGITAL_FILTER function or by the procedure in the previous section.

IDL's BLK_CON function provides a simple and efficient way to convolve a filter with a signal. Using $u(k)$ from the previous example and the bandstop filter created above creates the plot shown in the figure below.

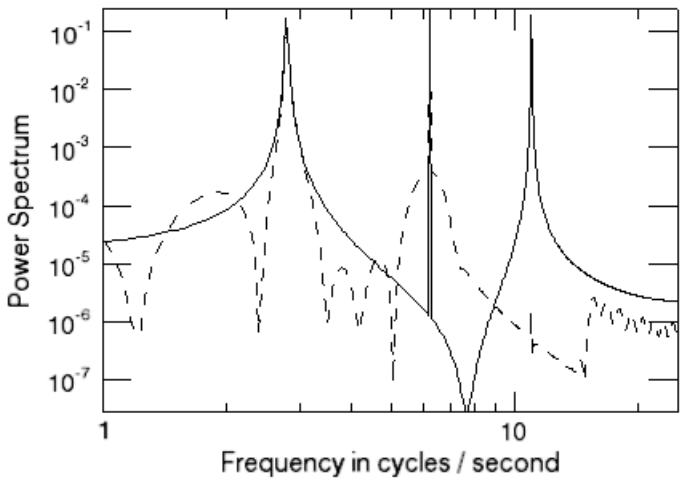


Figure 10-11: Digital Signal Before and After Filtering

Example Code

Type `@sigprc12` at the IDL prompt to run the batch file that creates this display. The source code is located in `sigprc12`, in the `examples/doc/signal` directory. See "Running the Example Code" on page 250 if IDL does not find the batch file.

The frequency response of the filtered signal shows that the frequency component at 11.0 cycles / second has been filtered out, while the frequency components at 2.8 and 6.25 cycles / second, as well as the DC component, have been passed by the filter.

Infinite Impulse Response Filters

Digital filters which must be implemented recursively are called Infinite Impulse Response (IIR) filters because, theoretically, the response of these filters to an impulse never settles to zero. In practice, the impulse response of many IIR filters approaches zero asymptotically, and may actually reach zero in a finite number of samples due to the finite word length of digital computers.

One method of designing digital filters starts with the Laplace transform representation of an analog filter with the required frequency response. For example, the Laplace transform representation (or continuous transfer function) of a second order notch filter with the notch at f_0 cycles per second is:

$$\frac{y(s)}{u(s)} = \frac{\left(\frac{f_0}{2\pi} + s^2\right)}{\left(1 + 2s\left(\frac{f_0}{2\pi}\right) + s^2\right)}$$

where *s* is the Laplace transform variable. Then the continuous transfer function is converted to the equivalent discrete transfer function using one of several techniques. One of these is the bilinear (Tustin) transform, where

```
(2/δ)*(z-1)/(z+1)
```

is substituted for the Laplace transform variable *s*. In this expression, *z* is the unit delay operator.

For the notch filter above, the bilinear transformation yields the following discrete transfer function:

$$\frac{y(z)}{u(z)} = \frac{\left(\frac{1 + c^2}{2} - 2cz + \frac{1 + c^2}{2}z^2\right)}{(c^2 - 2cz + z^2)}$$

where $c = (1 - \pi * f_0 * \delta) / (1 + \pi * f_0 * \delta)$.

Enter the following IDL statements to compute the coefficients of the discrete transfer function:

```
delt = 0.02
; Notch frequency in cycles per second:
f0 = 6.5
c = (1.0-!PI*F0*delt) / (1.0+!PI*F0*delt)
```

```
b = [(1+c^2)/2, -2*c, (1+c^2)/2]
a = [ c^2, -2*c, 1]
```

Example Code

Alternately, type `@sigprc13` at the IDL prompt to run the `sigprc13` batch file and create the plot variables. See "Running the Example Code" on page 250 if IDL does not find the batch file.

IIR Filter Implementation

Since an Infinite Impulse Response filter contains feedback loops, its output at every time step depends on previous outputs, and the filter must be implemented recursively with difference equations. The discrete transfer function

$$y(z) = \left(\frac{b_0 + b_1 z + \ldots + b_{nb} z^{nb}}{a_0 + a_1 z + \ldots + a_{na} z^{nb}} \right) u(z)$$

is implemented with the difference equation

$$y(k) = \frac{(b_0 u(k - nb) + b_1 u(k - nb + 1) + \ldots + b_{nb} u(k) - a_0 y(k - na) - a_1 y(k - na + 1) - \ldots - a_{na-1} y(k - 1))}{a_{na}}$$

An IIR filter is stable if the absolute values of the roots of the denominator of the discrete transfer function $a(z)$ are all less than one. The impulse response of a stable IIR filter approaches zero as the time index k approaches infinity. The frequency response function of a stable IIR filter is the Discrete Fourier Transform of the filter's impulse response.

The figure below plots the impulse and frequency response functions of the notch filter defined above using recursive difference equations.

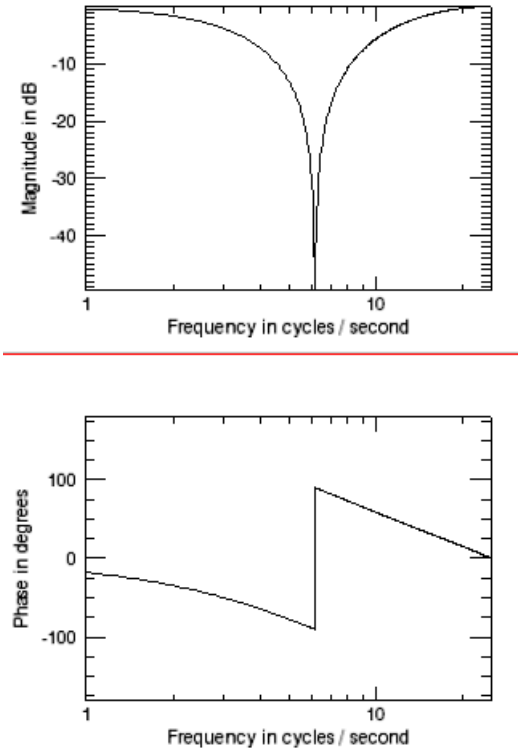


Figure 10-12: Impulse and Frequency Response of a Notch Filter

Example Code

Type `@sigprc14` at the IDL prompt to run the batch file that creates this display. The source code is located in `sigprc14`, in the `examples/doc/signal` directory. See "Running the Example Code" on page 250 if IDL does not find the batch file.

Note

Because the impulse response approaches zero, IDL may warn of floating-point underflow errors. This is an expected consequence of the digital implementation of an Infinite Impulse Response filter.

The same code could be used to filter any input sequence $u(k)$.

References

Bracewell, Ronald N., *The Fourier Transform and Its Applications*, New York: McGraw-Hill, 1978. ISBN 0-07-007013-X

Chen, Chi-Tsong, *One-Dimensional Digital Signal Processing*, New York: Marcel Dekker, Inc., 1979. ISBN 0-8247-6877-9

Jackson, Leland B., *Digital Filters and Signal Processing*, Boston: Kluwer Academic Publishers, 1986. ISBN 0-89838-174-6

Mayeda, Wataru, *Digital Signal Processing*, Englewood Cliffs, NJ: Prentice-Hall, Inc., 1993. ISBN 0-13-211301-5

Morgera, Salvatore D. and Krishna, Hari, *Digital Signal Processing: Applications to Communications and Algebraic Coding Theories*, Boston: Academic Press, 1989. ISBN 0-12-506995-2

Oppenheim, Alan V. and Schafer, Ronald W., *Discrete-time signal processing*, Englewood Cliffs, NJ: Prentice-Hall, 1989. ISBN 0-13-216292-X

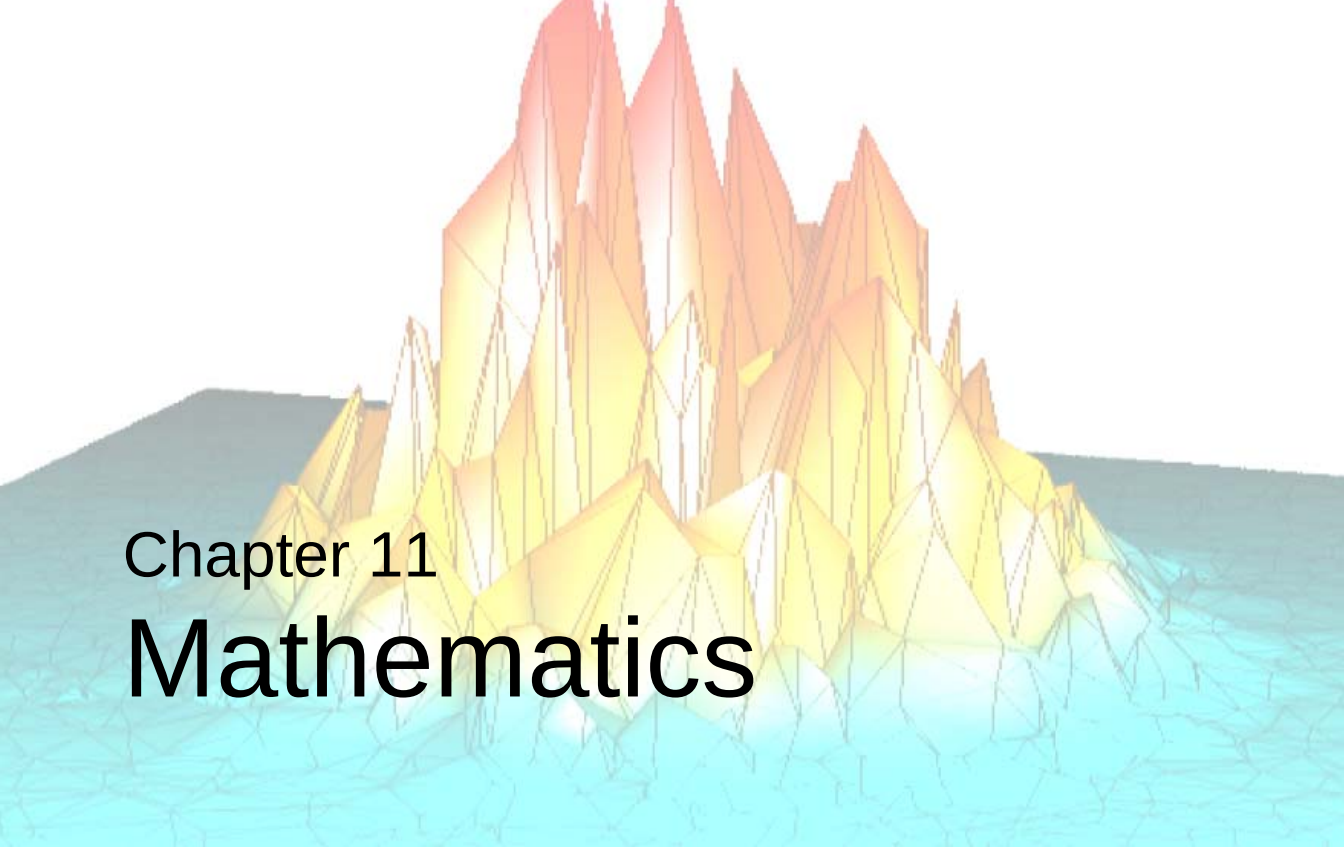
Peled, Abraham and Liu, Bede, *Digital Signal Processing*, New York: John Wiley & Sons, Inc., 1976. ISBN 0-471-01941-0

Press, William H. et al. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge: Cambridge University Press, 1992. ISBN 0-521-43108-5

Proakis, John G. and Manolakis, Dimitris G., *Digital Signal Processing: Principles, Algorithms, and Applications*, New York: Macmillan Publishing Company, 1992. ISBN 0-02-396815-X

Rabiner, Lawrence R. and Gold, Bernard, *Theory and application of digital signal processing*, Englewood Cliffs, NJ: Prentice-Hall, 1975. ISBN 0-139-14101-4

Strang, Gilbert and Nguyen, Truong, *Wavelets and Filter Banks*, Wellesley, MA: Wellesley-Cambridge Press, 1996. ISBN 0-961-40887-1



Chapter 11
Mathematics

The following topics are covered in this chapter:

Overview of Mathematics in IDL 280
IDL's Numerical Recipes Functions 281
Correlation Analysis 282
Curve and Surface Fitting 286
Eigenvalues and Eigenvectors 288
Gridding and Interpolation 294
Hypothesis Testing 295
Integration 297
Linear Systems 302
Nonlinear Equations 309
Optimization 311
Sparse Arrays 313
Time-Series Analysis 316
Multivariate Analysis 319
References 325

Overview of Mathematics in IDL

This chapter documentsIDL's mathematics and statistics procedures and functions. These include Numerical Recipes™ algorithms published in *Numerical Recipes in C: The Art of Scientific Computing* (Second Edition). For a list of IDL mathematical routines, see the functional category of "Mathematics" in the *IDL Quick Reference* manual. There you will find a brief introduction to the routines. Detailed information is available in the *IDL Reference Guide*. This chapter also includes introductory discussions of the following topics and an overview of the way IDL handles the particular problems involved:

- "Correlation Analysis" on page 282
- "Curve and Surface Fitting" on page 286
- "Eigenvalues and Eigenvectors" on page 288
- "Gridding and Interpolation" on page 294
- "Hypothesis Testing" on page 295
- "Integration" on page 297
- "Linear Systems" on page 302
- "Nonlinear Equations" on page 309
- "Optimization" on page 311
- "Sparse Arrays" on page 313
- "Time-Series Analysis" on page 316
- "Multivariate Analysis" on page 319

References are provided at the end of each section for a more detailed description and understanding of the topic.

Research Systems, Inc. is extremely interested in the accuracy of its algorithms. Bug reports, documentation errors and suggestions for future mathematics and statistics enhancements can be sent to RSI via:

Internet: `support@RSInc.com`

Fax: (303) 786-9909

Note

Floating-point numbers are inherently inaccurate. See "Accuracy and Floating Point Operations" on page 272 for details on roundoff and truncation errors.

IDL's Numerical Recipes Functions

IDL includes a number of routines based on algorithms published in *Numerical Recipes in C: The Art of Scientific Computing* (Second Edition). Routines derived from Numerical Recipes are noted as such in the *IDL Reference Guide* and in the IDL Online Help.

In IDL versions up to and including IDL version 3.6, mathematics functions based on Numerical Recipes algorithms required that input be in column-major format. This is no longer the case. Routines based on Numerical Recipes algorithms have been reworked and renamed, so that all IDL functions now expect input arrays to be in row-major format (composed of row vectors).

Note

To maintain compatibility with IDL programs based on earlier versions, the old routines (using the older input convention) are still available. No alterations need be made to existing code as a result of this change in IDL. We recommend that all new IDL programs take advantage of the new names and input convention.

Correlation Analysis

Given two *n*-element sample populations, *X* and *Y*, it is possible to quantify the degree of fit to a linear model using the correlation coefficient. The correlation coefficient, *r*, is a scalar quantity in the interval [-1.0, 1.0], and is defined as the ratio of the covariance of the sample populations to the product of their standard deviations.

$$r = \frac{\text{covariance of X and Y}}{(\text{standard deviation of X})(\text{standard deviation of Y})}$$

or

$$r = \frac{\frac{1}{N-1}\sum_{i=0}^{N-1}\left(x_i - \left[\sum_{k=0}^{N-1}\frac{x_k}{N}\right]\right)\left(y_i - \left[\sum_{k=0}^{N-1}\frac{y_k}{N}\right]\right)}{\sqrt{\frac{1}{N-1}\sum_{i=0}^{N-1}\left(x_i - \left[\sum_{k=0}^{N-1}\frac{x_k}{N}\right]\right)^2}\sqrt{\frac{1}{N-1}\sum_{i=0}^{N-1}\left(y_i - \left[\sum_{k=0}^{N-1}\frac{y_k}{N}\right]\right)^2}}$$

The correlation coefficient is a direct measure of how well two sample populations vary jointly. A value of *r* = +1 or *r* = –1 indicates a perfect fit to a positive or negative linear model, respectively. A value of *r* close to +1 or –1 indicates a high degree of correlation and a good fit to a linear model. A value of *r* close to 0 indicates a poor fit to a linear model.

Correlation Example

The following sample populations represent a perfect positive linear correlation.

```
X = [-8.1,  1.0, -14.3, 4.2, -10.1, 4.3, 6.3, 5.0, 15.1, -2.2]
Y = [-9.8, -0.7, -16.0, 2.5, -11.8, 2.6, 4.6, 3.3, 13.4, -3.9]
;Compute the correlation coefficient of X and Y.
PRINT, CORRELATE(X, Y)
```

IDL prints:

```
1.00000
```

The following sample populations represent a high negative linear correlation.

```
X = [ 1.8, -2.7,  0.7, -0.5, -1.3, -0.9,  0.6, -1.5,  2.5,  3.0]
Y = [-4.7,  9.8, -3.7,  2.8,  5.1,  3.9, -3.6,  5.8, -7.3, -7.4]
;Compute the correlation coefficient of X and Y:
PRINT, CORRELATE(X, Y)
```

IDL prints:

```
-0.979907
```

The following sample populations represent a poor linear correlation.

```
X = [-1.8,  0.1, -0.1, 1.9, 0.5,  1.1, 1.9,  0.3, -0.2, -1.0]
Y = [ 1.5, -1.0, -0.6, 1.1, 0.7, -0.7, 1.1, -0.1,  0.6, -0.1]
;Compute the correlation coefficient of X and Y:
PRINT, CORRELATE(X, Y)
```

IDL prints:

```
0.0322859
```

Notes on Interpreting the Correlation Coefficient

When interpreting the value of the correlation coefficient, it is important to remember the following two caveats:

1. Although a high degree of correlation (a value close to +1 or –1) indicates a good mathematical fit to a linear model, its applied interpretation may be completely nonsensical. For example, there may be a high degree of correlation between the number of scientists using IDL to study atmospheric phenomena and the consumption of alcohol in Russia, but the two events are clearly unrelated.
2. Although a correlation coefficient close to 0 indicates a poor fit to a linear model, it does not mean that there is no correlation between the two sample populations. It is possible that the relationship between *X* and *Y* is accurately described by a nonlinear model. See “Curve and Surface Fitting” on page 286 for further details on fitting data to linear and nonlinear models.

Multiple Linear Models

The fundamental principles of correlation that apply to the linear model of two sample populations may be extended to the multiple-linear model. The degree of relationship between three or more sample populations may be quantified using the

multiple correlation coefficient. The degree of relationship between two sample populations when the effects of all other sample populations are removed may be quantified using the partial correlation coefficient. Both of these coefficients are scalar quantities in the interval [0.0, 1.0]. A value of +1 indicates a perfect linear relationship between populations. A value close to +1 indicates a high degree of linear relationship between populations; whereas a value close to 0 indicates a poor linear relationship between populations. (Although a value of 0 indicates no linear relationship between populations, remember that there may be a nonlinear relationship.)

Partial Correlation Example

Define the independent (*X*) and dependent (*Y*) data.

```
X = [[0.477121, 2.0, 13.0], $
    [0.477121, 5.0, 6.0], $
    [0.301030, 5.0, 9.0], $
    [0.000000, 7.0, 5.5], $
    [0.602060, 3.0, 7.0], $
    [0.698970, 2.0, 9.5], $
    [0.301030, 2.0, 17.0], $
    [0.477121, 5.0, 12.5], $
    [0.698970, 2.0, 13.5], $
    [0.000000, 3.0, 12.5], $
    [0.602060, 4.0, 13.0], $
    [0.301030, 6.0, 7.5], $
    [0.301030, 2.0, 7.5], $
    [0.698970, 3.0, 12.0], $
    [0.000000, 4.0, 14.0], $
    [0.698970, 6.0, 11.5], $
    [0.301030, 2.0, 15.0], $
    [0.602060, 6.0, 8.5], $
    [0.477121, 7.0, 14.5], $
    [0.000000, 5.0, 9.5]]
Y = [97.682, 98.424, 101.435, 102.266, 97.067, 97.397, $
   99.481, 99.613, 96.901, 100.152, 98.797, 100.796, $
   98.750, 97.991, 100.007, 98.615, 100.225, 98.388, $
   98.937, 100.617]
```

Compute the multiple correlation of *Y* on the first column of *X*. The result should be 0.798816.

```
PRINT, M_CORRELATE(X[0,*], Y)
```

IDL prints:

```
0.798816
```

Compute the multiple correlation of *Y* on the first two columns of *X*. The result should be 0.875872.

```
PRINT, M_CORRELATE(X[0:1,*], Y)
```

IDL prints:

```
0.875872
```

Compute the multiple correlation of *Y* on all columns of *X*. The result should be 0.877197.

```
PRINT, M_CORRELATE(X, Y)
```

IDL prints:

```
0.877197
;Define the five sample populations.
X0 = [30, 26, 28, 33, 35, 29]
X1 = [0.29, 0.33, 0.34, 0.30, 0.30, 0.35]
X2 = [65, 60, 65, 70, 70, 60]
X3 = [2700, 2850, 2800, 3100, 2750, 3050]
Y  = [37, 33, 32, 37, 36, 33]
```

Compute the partial correlation of *X*1 and *Y* with the effects of *X*0, *X*2 and *X*3 removed.

```
PRINT, P_CORRELATE(X1, Y, REFORM([X0,X2,X3], 3, N_ELEMENTS(X1)))
```

IDL prints:

```
0.996017
```

Routines for Computing Correlations

See "Correlation Analysis" (in the functional category "Mathematics" in the *IDL Quick Reference* manual) for a brief description of IDL routines for computing correlations. Detailed information is available in the *IDL Reference Guide*.

Curve and Surface Fitting

The problem of curve fitting may be stated as follows:

Given a tabulated set of data values $\{x_i, y_i\}$ and the general form of a mathematical model (a function $f(x)$ with unspecified parameters), determine the parameters of the model that minimize an error criterion. The problem of surface fitting involves tabulated data of the form $\{x_i, y_i, z_i\}$ and a function $f(x, y)$ of two spatial dimensions.

For example, we can use the CURVEFIT routine to determine the parameters A and B of a user-supplied function $f(x)$, such that the sums of the squares of the residuals between the tabulated data $\{x_i, y_i\}$ and function are minimized. We will use the following function and data:

$f(x) = a\ (1 - e^{-bx})$

x_i = [0.25, 0.75, 1.25, 1.75, 2.25]

y_i = [0.28, 0.57, 0.68, 0.74, 0.79]

First we must provide a procedure written in IDL to evaluate the function, f, and its partial derivatives with respect to the parameters a_0 and a_1:

```
PRO funct, X, A, F, PDER
   F = A[0] * (1.0 - EXP(-A[1] * X))
   ; If the function is called with four parameters,
   ; calculate the partial derivatives:
   IF N_PARAMS() GE 4 THEN BEGIN
      ; PDER's column dimension is equal to the number of
      ; elements in xi and its row dimension is equal to
      ; the number of parameters in the function F:
      pder = FLTARR(N_ELEMENTS(X), 2)
      ; Compute the partial derivatives with respect to
      ; a0 and place in the first row of PDER:
      pder[*, 0] = 1.0 - EXP(-A[1] * X)
      ; Compute the partial derivatives with respect to
      ; a1 and place in the second row of PDER:
      pder[*, 1] = A[0] * x * EXP(-A[1] * X)
  ENDIF
END
```

Note

The function will not calculate the partial derivatives unless it is called with four parameters. This allows the calling routine (in this case CURVEFIT) to avoid the extra computation in cases when the partial derivatives are not needed.

Next, we can use the following IDL commands to find the function's parameters:

```
;Define the vectors of tabulated:
X = [0.25, 0.75, 1.25, 1.75, 2.25]
;data values:
Y = [0.28, 0.57, 0.68, 0.74, 0.79]
;Define a vector of weights:
W = 1.0 / Y
;Provide an initial guess of the function's parameters:
A = [1.0, 1.0]
;Compute the parameters a0 and a1:
yfit = CURVEFIT(X, Y, W, A, SIGMA_A, FUNCTION_NAME = 'funct')
;Print the parameters, which are returned in A:
PRINT, A
```

IDL prints:

```
  0.787386  1.71602
```

Thus the nonlinear function that best fits the data is:

$f(x) = 0.787386 (1 - e^{-1.71602x})$

Routines for Curve and Surface Fitting

See "Curve and Surface Fitting" (in the functional category "Mathematics" in the *IDL Quick Reference* manual) for a brief description of IDL routines for curve and surface fitting. Detailed information is available in the *IDL Reference Guide*.

Eigenvalues and Eigenvectors

Consider a system of equations that satisfies the array-vector relationship $Ax = \lambda x$, where A is an n-by-n array, x is an n-element vector, and λ is a scalar. A scalar λ and nonzero vector x that simultaneously satisfy this relationship are referred to as an eigenvalue and an eigenvector of the array A, respectively. The set of all eigenvectors of the array A is then referred to as the eigenspace of A. Ideally, the eigenspace will consist of n linearly-independent eigenvectors, although this is not always the case.

IDL computes the eigenvalues and eigenvectors of a real symmetric n-by-n array using Householder transformations and the QL algorithm with implicit shifts. The eigenvalues of a real, n-by-n nonsymmetric array are computed from the upper Hessenberg form of the array using the QR algorithm. Eigenvectors are computed using inverse subspace iteration.

Although it is not practical for numerical computation, the problem of computing eigenvalues and eigenvectors can also be defined in terms of the determinant function. The eigenvalues of an n-by-n array A are the roots of the polynomial defined by $\det(A - \lambda I)$, where I is the identity matrix (an array with 1s on the main diagonal and 0s elsewhere) with the same dimensions as A. By expressing eigenvalues as the roots of a polynomial, we see that they can be either real or complex. If an eigenvalue is complex, its corresponding eigenvectors are also complex.

The following examples demonstrate how to use IDL to compute the eigenvalues and eigenvectors of real, symmetric and nonsymmetric n-by-n arrays. Note that it is possible to check the accuracy of the computed eigenvalues and eigenvectors by algebraically manipulating the definition given above to read $Ax - \lambda x = 0$; in this case 0 denotes an n-element vector, all elements of which are zero.

Symmetric Array with *n* Distinct Real Eigenvalues

To compute eigenvalues and eigenvectors of a real, symmetric, n-by-n array, begin with a symmetric array A.

Note

The eigenvalues and eigenvectors of a real, symmetric n-by-n array are real numbers.

```
A = [[ 3.0,  1.0, -4.0], $
     [ 1.0,  3.0, -4.0], $
     [-4.0, -4.0,  8.0]]
```

```
; Compute the tridiagonal form of A:
TRIRED, A, D, E
; Compute the eigenvalues (returned in vector D) and
; the eigenvectors (returned in the rows of the array A):
TRIQL, D, E, A
; Print eigenvalues:
PRINT, D
```

IDL prints:

```
  2.00000  4.76837e-07  12.0000
```

The exact values are: [2.0, 0.0, 12.0].

```
;Print the eigenvectors, which are returned as row vectors in A:
PRINT, A
```

IDL prints:

```
  0.707107  -0.707107   0.00000
 -0.577350  -0.577350  -0.577350
 -0.408248  -0.408248   0.816497
```

The exact eigenvectors are:

$$\begin{bmatrix} 1/\sqrt{2} & -1/\sqrt{2} & 0 \\ -1/\sqrt{3} & -1/\sqrt{3} & -1/\sqrt{3} \\ -1/\sqrt{6} & -1/\sqrt{6} & 2/\sqrt{6} \end{bmatrix}$$

Nonsymmetric Array with *n* Distinct Real and Complex Eigenvalues

To compute the eigenvalues and eigenvectors of a real, nonsymmetric *n*-by-*n* array, begin with an array *A*. In this example, there are *n* distinct eigenvalues and *n* linearly-independent eigenvectors.

```
A = [[ 1.0, 0.0,  2.0], $
     [ 0.0, 1.0, -1.0], $
     [-1.0, 1.0,  1.0]]
; Reduce to upper Hessenberg format:
hes = ELMHES(A)
; Compute the eigenvalues:
evals = HQR(hes)
; Print the eigenvalues:
PRINT, evals
```

IDL prints:

```
(  1.00000, -1.73205)(  1.00000,  1.73205)
(  1.00000,  0.00000)
```

Note

The three eigenvalues are distinct, and that two are complex. Note also that complex eigenvalues of an *n*-by-*n* real, nonsymmetric array always occur in complex conjugate pairs.

```
; Initialize a variable to contain the residual:
residual = 1
; Compute the eigenvectors and the residual for each
; eigenvalue/eigenvector pair, using double-precision arithmetic:
evecs = EIGENVEC(A, evals, /DOUBLE, RESIDUAL=residual)
; Print the eigenvectors, which are returned as
; row vectors in evecs:
PRINT, evecs[*,0]
```

IDL prints:

```
(  0.68168704,  0.18789033)( -0.34084352, -0.093945164)
(  0.16271780, -0.59035830)
PRINT, evecs[*,1]
```

IDL prints:

```
(  0.18789033,  0.68168704)( -0.093945164, -0.34084352)
( -0.59035830,  0.16271780)
PRINT, evecs[*,2]
```

IDL prints:

```
(  0.70710678,  0.0000000)(  0.70710678,  0.0000000)
( -2.3570226e-21, 0.0000000)
```

We can check the accuracy of these results using the relation $Ax - \lambda x = 0$. The array contained in the variable specified by the RESIDUAL keyword contains the result of this computation.

```
PRINT, residual
```

IDL prints:

```
( -1.2021898e-07,  1.1893681e-07)(  6.0109490e-08, -5.9468404e-08)
(  1.0300230e-07,  1.0411269e-07)
(  1.1893681e-07, -1.2021898e-07)( -5.9468404e-08,  6.0109490e-08)
(  1.0411269e-07,  1.0300230e-07)
(      0.0000000,      0.0000000)(      0.0000000,      0.0000000)
```

The results are all zero to within machine precision.

Repeated Eigenvalues

To compute the eigenvalues and eigenvectors of a real, nonsymmetric *n*-by-*n* array, begin with an array *A*. In this example, there are fewer than *n* distinct eigenvalues, but *n* independent eigenvectors are available.

```
A = [[8.0, 0.0, 3.0], $
     [2.0, 2.0, 1.0], $
     [2.0, 0.0, 3.0]]
; Reduce A to upper Hessenberg form and compute the eigenvalues.
; Note that both operations can be combined into a single command.
evals = HQR(ELMHES(A))
; Print the eigenvalues:
PRINT, evals
```

IDL prints:

```
(  9.00000,  0.00000) (  2.00000,  0.00000)
(  2.00000,  0.00000)
```

Note

The three eigenvalues are real, but only two are distinct.

```
; Initialize a variable to contain the residual:
residual = 1
; Compute the eigenvectors and residual, using
; double-precision arithmetic:
evecs = EIGENVEC(A, evals, /DOUBLE, RESIDUAL=residual)
; Print the eigenvectors:
PRINT, evecs[*,0]
```

IDL prints:

```
(  0.90453403,  0.0000000)(  0.30151134,  0.0000000)
(  0.30151134,  0.0000000)
PRINT, evecs[*,1]
```

IDL prints:

```
( -0.27907279,  0.0000000)( -0.78140380,  0.0000000)
(  0.55814557,  0.0000000)
PRINT, evecs[*,2]
```

IDL prints:

```
( -0.27907279,  0.0000000)( -0.78140380,  0.0000000)
(  0.55814557,  0.0000000)
```

We can compute an independent eigenvector for the repeated eigenvalue (2.0) by perturbing it slightly, allowing the algorithm EIGENVEC to recognize the eigenvalue as distinct and to compute a linearly-independent eigenvector.

```
newresidual = 1
evecs[*,2] = EIGENVEC(A, evals[2]+1.0e-6, /DOUBLE, $
   RESIDUAL = newresidual)
PRINT, evecs[*,2]
```

IDL prints:

```
( -0.33333333,  0.0000000)(  0.66666667,  0.0000000)
(  0.66666667,  0.0000000)
```

Once again, we can check the accuracy of these results by checking that each element in the residuals —for both the original eigenvectors and the perturbed eigenvector— is zero to within machine precision.

The So-called Defective Case

In the so-called defective case, there are fewer than *n* distinct eigenvalues and fewer than *n* linearly-independent eigenvectors. Begin with an array *A*:

```
A = [[2.0, -1.0], $
     [1.0,  0.0]]
; Reduce A to upper Hessenberg form and compute the eigenvalues.
; Note that both operations can be combined into a single command.
evals = HQR(ELMHES(A))
; Print the eigenvalues:
PRINT, evals
```

IDL prints:

```
(  1.00000,  0.00000)(  1.00000,  0.00000)
```

Note

The two eigenvalues are real, but not distinct.

```
; Compute the eigenvectors, using double-precision arithmetic:
evecs = EIGENVEC(A, evals, /DOUBLE)
; Print the eigenvectors:
PRINT, evecs[*,0]
```

IDL prints:

```
(  0.70710678,  0.0000000)(  0.70710678,  0.0000000)
PRINT, evecs[*,1]
```

IDL prints:

```
(  0.70710678,  0.0000000)(  0.70710678,  0.0000000)
```

We attempt to compute an independent eigenvector using the method described in the previous example:

```
evecs[*,1] = EIGENVEC(A, evals[1]+1.0e-6, /DOUBLE)
PRINT, evecs[1,*]
```

IDL prints:

```
(  0.70710678,  0.0000000)(  0.70710678,  0.0000000)
```

In this example, *n* independent eigenvectors do not exist. This situation is termed the defective case and cannot be resolved analytically or numerically.

Routines for Computing Eigenvalues and Eigenvectors

See "Eigenvalues and Eigenvectors" (in the functional category "Mathematics" in the *IDL Quick Reference* manual) for a brief description of IDL routines for computing eigenvalues and eigenvectors. Detailed information is available in the *IDL Reference Guide*.

Gridding and Interpolation

Given a set of tabulated data in *n*-dimensions with each dimension being described as follows:

1. $\{x_i, y_i = f(x_i)\}$,
2. $\{x_i, y_i, z_i = f(x_i, y_i)\}$, or
3. $\{x_i, y_i, z_i, w_i = f(x_i, y_i, z_i)\}$

it is possible to calculate intermediate values of the function *f* using interpolation. IDL includes a variety of routines to solve this type of problem.

The determination of intermediate values is based upon an interpolating function that establishes a relationship between the tabulated data points. Different algorithms employ different types of interpolating functions suitable for different types of data trends.

Unlike curve-fitting algorithms, interpolation requires that the interpolating function be an exact fit at each of the tabulated data points. Interpolation does not use any type of error analysis and its accuracy depends upon the behavior of the interpolating function between successive data points. Polynomial, spline, and nearest-neighbor are among the interpolation methods used in IDL. Kriging is another interpolation method, one which does not require an exact fit at each tabulated data point. Kriging applies a weighting to each of the tabulated data points based on spatial variance and trends among the points. Weights are computed by combining calculations of spatial continuity and anistropy within either an exponential or spherical semivariogram model.

Gridding, a topic closely related to interpolation, is the problem of creating uniformly-spaced planar data from irregularly-spaced data. IDL handles this type of problem by constructing a Delaunay triangulation. This method is highly accurate and has great utility since many of IDL's graphics routines require uniformly-gridded data. Extrapolation, the estimation of values outside the range of tabulated data, is also possible using this method.

Routines for Gridding and Interpolation

See "Gridding and Interpolation" (in the functional category "Mathematics" in the *IDL Quick Reference* manual) for a brief description of IDL routines for gridding and interpolation. Detailed information is available in the *IDL Reference Guide*.

Hypothesis Testing

Hypothesis testing tests one or more sample populations for a statistical characteristic or interaction. The results of the testing process are generally used to formulate conclusions about the probability distributions of the sample populations.

Hypothesis testing involves four steps:

- The formulation of a hypothesis.
- The selection and collection of sample population data.
- The application of an appropriate test.
- The interpretation of the test results.

For example, suppose the FDA wishes to establish the effectiveness of a new drug in the treatment of a certain ailment. Researchers test the assumption that the drug is effective by administering it to a sample population and collecting data on the patients' health. Once the data are collected, an appropriate statistical test is selected and the results analyzed. If the interpretation of the test results suggests a statistically significant improvement in the patients' condition, the researchers conclude that the drug will be effective in general.

It is important to remember that a valid or successful test does not prove the proposed hypothesis. Only by disproving competing or opposing hypotheses can a given assumption's validity be statistically established.

One- and Two-sided Tests

In the above example, only the hypothesis that the drug would significantly improve the condition of the patients receiving it was tested. This type of test is called one-sided or one-tailed, because it is concerned with deviation in one direction from the norm (in this case, improvement of the patients' condition). A hypothesis designed to test the improvement or ill-effect of the trial drug on the patient group would be called two-sided or two-tailed.

Parametric and Nonparametric Tests

Tests of hypothesis are usually classified into parametric and nonparametric methods. Parametric methods make assumptions about the underlying distribution from which sample populations are selected. Nonparametric methods make no assumptions about a sample population's distribution and are often based upon magnitude-based ranking, rather than actual measurement data. In many cases it is possible to replace a

parametric test with a corresponding nonparametric test without significantly affecting the conclusion.

The following example demonstrates this by replacing the parametric T-means test with the nonparametric Wilcoxon Rank-Sum test to test the hypothesis that two sample populations have significantly different means of distribution.

Define two sample populations.

```
X = [257, 208, 296, 324, 240, 246, 267, 311, 324, 323, 263, $
     305, 270, 260, 251, 275, 288, 242, 304, 267]
Y = [201,  56, 185, 221, 165, 161, 182, 239, 278, 243, 197, $
     271, 214, 216, 175, 192, 208, 150, 281, 196]
```

Compute the T-statistic and its significance, using IDL's TM_TEST function, assuming that *X* and *Y* belong to Normal populations with the same variance.

```
PRINT, TM_TEST(X, Y)
```

IDL prints:

```
5.52839  2.52455e-06
```

The small value of the significance (2.52455e-06) indicates that *X* and *Y* have significantly different means.

Compute the Wilcoxon Rank-Sum Test, using IDL's RS_TEST function, to test the hypothesis that *X* and *Y* have the same mean of distribution.

```
PRINT, RS_TEST(X, Y)
```

IDL prints:

```
-4.26039  1.01924e-05
```

The small value of the computed probability (1.01924e-05) requires the rejection of the proposed hypothesis and the conclusion that *X* and *Y* have significantly different means of distribution.

Each of IDL's 11 parametric and nonparametric hypothesis testing functions is based upon a well-known and widely-accepted statistical test. Each of these functions returns a two-element vector containing the statistic on which the test is based and its significance. Examples are provided and demonstrate how the result is interpreted.

Routines for Hypothesis Testing

See "Hypothesis Testing" (in the functional category "Mathematics" in the *IDL Quick Reference* manual) for a brief description of IDL routines for hypothesis testing. More detailed information is available in the *IDL Reference Guide*.

Integration

Numerical methods of approximating integrals are important in many areas of pure and applied science. For a function of a single variable, $f(x)$, it is often the case that the antiderivative $F = \int f(x)\,dx$ is unavailable using standard techniques such as trigonometric substitutions and integration-by-parts formulas. These standard techniques become increasingly unusable when integrating multivariate functions, $f(x, y)$ and $f(x, y, z)$. Numerically approximating the integral operator provides the only method of solution when the antiderivative is not explicitly available. IDL offers the following numerical methods for the integration of uni-, bi-, and trivariate functions:

- Integration of a univariate function over an open or closed interval is possible using one of several routines based on well known methods developed by Romberg and Simpson.

$$I = \int_{x=a}^{x=b} f(x)dx$$

- The problem of integrating over a tabulated set of data $\{ x_i, y_i = f(x_i) \}$ can be solved using a highly accurate 5-point Newton-Cotes formula. This method is more accurate and efficient than using interpolation or curve-fitting to find an approximate function and then integrating.
- Integration of a bivariate function over a regular or irregular region in the *x-y* plane is possible using an iterated Gaussian Quadrature routine.

$$I = \int_{x=a}^{x=b} \int_{y=p(x)}^{y=q(x)} f(x, y)dydx$$

- Integration of a trivariate function over a regular or irregular region in *x-y-z* space is possible using an iterated Gaussian Quadrature routine.

$$I = \int_{x=a}^{x=b} \int_{y=p(x)}^{y=q(x)} \int_{z=u(x,y)}^{z=v(x,y)} f(x, y, z)dzdydx$$

Note

IDL's iterated Gaussian Quadrature routines, INT_2D and INT_3D, follow the *dy-dx* and *dz-dy-dx* order of evaluation, respectively. Problems not conforming to this standard must be changed as described in the following example.

A Bivariate Function

Suppose that we wish to evaluate

$$\int_{y=0}^{y=4}\int_{x=\sqrt{y}}^{x=2} y \cdot \cos(x^5)\,dx\,dy$$

The order of integration is initially described as a *dx-dy* region in the *x-y* plane. Using the diagram below, you can easily change the integration order to *dy-dx*.

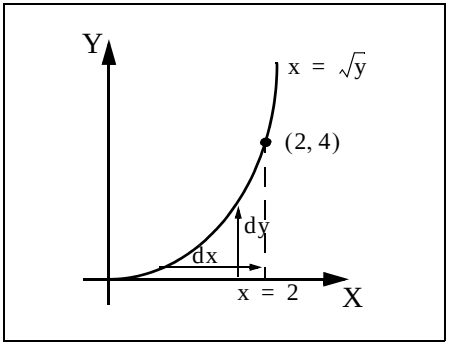


Figure 11-1: The Bivariate Function

The integral is now of the form

$$\int_{x=0}^{x=2}\int_{y=0}^{y=x^2} y \cdot \cos(x^5)\,dy\,dx$$

The new expression can be evaluated using the INT_2D function.

To use INT_2D, we must specify the function to be integrated and expressions for the upper and lower limits of integration. First, we write an IDL function for the integrand, the function *f* (*x*, *y*):

```
FUNCTION fxy, X, Y
  RETURN, Y * COS(X^5)
END
```

Next, we write a function for the limits of integration of the inner integral. Note that the limits of the outer integral are specified numerically, in vector form, while the limits of the inner integral must be specified as an IDL function even if they are constants. In this case, the function is:

```
FUNCTION pq_limits, X
  RETURN, [0.0, X^2]
END
```

Now we can use the following IDL commands to print the value of the integral expressed above. First, we define a variable AB_LIMITS containing the vector of lower and upper limits of the outer integral. Next, we call INT_2D. The first argument is the name of the IDL function that represents the integrand (FXY, in this case). The second argument is the name of the variable containing the vector of limits for the outer integral (AB_LIMITS, in this case). The third argument is the name of the IDL function defining the lower and upper limits of the inside integral (PQ_LIMITS, in this case). The fourth argument (48) refers to the number of transformation points used in the computation. As a general rule, the number of transformation points used with iterated Gaussian Quadrature should increase as the integrand becomes more oscillatory or the region of integration becomes more irregular.

```
ab_limits = [0.0, 2.0]
PRINT, INT_2D('fxy', ab_limits, 'pq_limits', 48)
```

IDL prints:

```
0.055142668
```

This is the exact solution to 9 decimal accuracy.

A Trivariate Function

Suppose that we wish to evaluate

$$\int_{x=-2}^{x=2}\int_{y=-\sqrt{4-x^2}}^{y=\sqrt{4-x^2}}\int_{z=0}^{z=\sqrt{4-x^2-y^2}} z(x^2+y^2+z^2)^{3/2}\,dz\,dy\,dx$$

This integral can be evaluated using the INT_3D function. As with INT_2D, we must specify the function to be integrated and expressions for the upper and lower limits of integration. Note that in this case IDL functions must be provided for the upper and lower integration limits of both inside integrals.

For the above integral, the required functions are the integrand $f(x, y, z)$:

```
FUNCTION fxyz, X, Y, Z
  RETURN, Z * (X^2 + Y^2 + Z^2)^1.5
END
```

The limits of integration of the first inside integral:

```
FUNCTION pq_limits, X
  RETURN, [-SQRT(4.0 - X^2), SQRT(4.0 -X^2)]
END
```

The limits of integration of the second inside integral:

```
FUNCTION uv_limits, X, Y
  RETURN, [0.0, SQRT(4.0 - X^2 - Y^2)]
END
```

We can use the following IDL commands to determine the value of the above integral using 6, 10, 20 and 48 transformation points.

For 6 transformation points:

```
PRINT, INT_3D('fxyz', [-2.0, 2.0], $
   'pq_limits', 'uv_limits', 6)
```

IDL prints:

```
57.417720
```

For 10 transformation points:

```
PRINT, INT_3D('fxyz', [-2.0, 2.0], $
   'pq_limits', 'uv_limits', 10)
```

IDL prints:

```
57.444248
```

20 transformation points:

```
PRINT, INT_3D('fxyz', [-2.0, 2.0], $
   'pq_limits', 'uv_limits', 20)
```

IDL prints:

```
57.446201
```

48 transformation points:

```
PRINT, INT_3D('fxyz', [-2.0, 2.0], $
   'pq_limits', 'uv_limits', 48)
```

IDL prints:

```
57.446265
```

The exact solution to 6-decimal accuracy is 57.446267.

Routines for Differentiation and Integration

See "Differentiation and Integration" (in the functional category "Mathematics" in the *IDL Quick Reference* manual) for a brief description of IDL routines for differentiation and integration. Detailed information is available in the *IDL Reference Guide*.

Linear Systems

IDL offers a variety of methods for the solution of simultaneous linear equations. In order to use these routines successfully, the user should consider both existence and uniqueness criteria and the potential difficulties in finding the solution numerically.

The solution vector x of an n-by-n linear system $Ax = b$ is guaranteed to exist and to be unique if the coefficient array A is invertible. Using a simple algebraic manipulation, it is possible to formulate the solution vector x in terms of the inverse of the coefficient array A and the right-side vector b: $x = A^{-1}b$. Although this relationship provides a concise mathematical representation of the solution, it is never used in practice. Array inversion is computationally expensive (requiring a large number of floating-point operations) and prone to severe round-off errors.

An alternate way of describing the existence of a solution is to say that the system $Ax = b$ is solvable if and only if the vector b may be expressed as a linear combination of the columns of A. This definition is important when considering the solutions of non-square (over- and under-determined) linear systems.

While the invertabiltiy of the coefficient array A may ensure that a solution exists, it does not help in determining the solution. Some systems can be solved accurately using numerical methods whereas others cannot. In order to better understand the accuracy of a numerical solution, we can classify the condition of the system it solves.

The scalar quantity known as the condition number of a linear system is a measure of a solution's sensitivity to the effects of finite-precision arithmetic. The condition number of an n-by-n linear system $Ax = b$ is computed explicitly as $|A||A^{-1}|$ (where $||$ denotes a Euclidean norm). A linear system whose condition number is small is considered well-conditioned and well suited to numerical computation. A linear system whose condition number is large is considered ill-conditioned and prone to computational errors. To some extent, the solution of an ill-conditioned system may be improved using an extended-precision data type (such as double-precision float). Other situations require an approximate solution to the system using its Singular Value Decomposition.

The following two examples show how the singular value decomposition may be used to find solutions when a linear system is over- or underdetermined.

Overdetermined Systems

In the case of the overdetermined system (when there are more linear equations than unknowns), the vector b cannot be expressed as a linear combination of the columns

of array *A*. (In other words, b lies outside of the subspace spanned by the columns of *A*.) Using IDL's SVDC procedure, it is possible to determine a projected solution of the overdetermined system (b is projected onto the subspace spanned by the columns of A and then the system is solved). This type of solution has the property of minimizing the residual error $E = b - Ax$ in a least-squares sense.

Suppose that we wish to solve the following linear system:

$$\begin{bmatrix} 1.0 & 2.0 \\ 1.0 & 3.0 \\ 0.0 & 0.0 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = \begin{bmatrix} 4.0 \\ 5.0 \\ 6.0 \end{bmatrix}$$

The vector *b* does not lie in the two-dimensional subspace spanned by the columns of *A* (there is no linear combination of the columns of *A* that yield *b*), and therefore an exact solution is not possible.

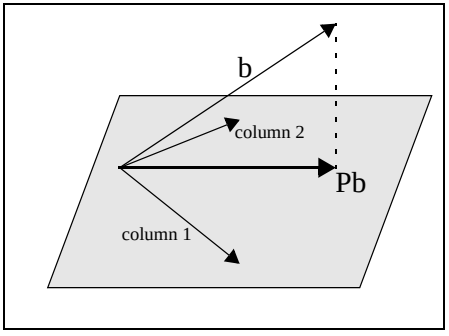


Figure 11-2: Overdetermined System Diagram

It is possible, however, to find a solution to this system that minimizes the residual error by orthogonally projecting the vector b onto the two-dimensional subspace spanned by the columns of the array *A*. The projected vector is then used as the right-hand side of the system. The orthogonal projection of *b* onto the column space of *A* may be expressed with the array-vector product $A(A^TA)^{-1}A^Tb$, where $A(A^TA)^{-1}A^T$ is known as the projection matrix, P.

In this example, the array-vector product Pb yields:

$$\begin{bmatrix} 4.0 \\ 5.0 \\ 0.0 \end{bmatrix}$$

and we wish to solve the linear system

$$\begin{bmatrix} 1.0 & 2.0 \\ 1.0 & 3.0 \\ 0.0 & 0.0 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = \begin{bmatrix} 4.0 \\ 5.0 \\ 0.0 \end{bmatrix} \quad \text{where} \quad \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = \begin{bmatrix} 2.0 \\ 1.0 \end{bmatrix}$$

In many cases, the explicit calculation of the projected solution is numerically unstable, resulting in large accumulated round-off errors. For this reason it is best to use singular value decomposition to effect the orthogonal projection of the vector b onto the subspace spanned by the columns of the array A.

The following IDL commands use singular value decomposition to solve the system in a numerically stable manner. Begin with the array A:

```
A = [[1.0, 2.0], $
     [1.0, 3.0], $
     [0.0, 0.0]]
; Define the right-hand side vector B:
B = [4.0, 5.0, 6.0]
; Compute the singular value decomposition of A:
SVDC, A, W, U, V
```

Create a diagonal array WP of reciprocal singular values from the output vector W. To avoid overflow errors when the reciprocal values are calculated, only elements with absolute values greater than or equal to 1.0×10^{-5} are reciprocated.

```
N = N_ELEMENTS(W)
WP = FLTARR(N, N)
FOR K = 0, N-1 DO $
   IF ABS(W(K)) GE 1.0e-5 THEN WP(K, K) = 1.0/W(K)
```

We can now express the solution to the linear system as a array-vector product. (See Section 2.6 of *Numerical Recipes* for a derivation of this formula.)

```
X = V ## WP ## TRANSPOSE(U) ## B
; Print the solution:
PRINT, X
```

IDL Prints:

```
2.00000
1.00000
```

Underdetermined Systems

In the case of the underdetermined system (when there are fewer linear equations than unknowns), a unique solution is not possible. Using IDL's SVDC procedure it is possible to determine the minimal norm solution. Given a vector norm, this type of solution has the property of having the minimal length of all possible solutions with respect to that norm.

Suppose that we wish to solve the following linear system.

$$\begin{bmatrix} 1.0 & 3.0 & 3.0 & 2.0 \\ 2.0 & 6.0 & 9.0 & 5.0 \\ -1.0 & -3.0 & 3.0 & 0.0 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1.0 \\ 5.0 \\ 5.0 \end{bmatrix}$$

Using elementary row operations it is possible to reduce the system to

$$\begin{bmatrix} 1.0 & 3.0 & 3.0 & 2.0 \\ 0.0 & 0.0 & 3.0 & 1.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1.0 \\ 3.0 \\ 0.0 \end{bmatrix}$$

It is now possible to express the solution x in terms of x_1 and x_3:

$$x = \begin{bmatrix} -2 - 3x_1 - x_3 \\ x_1 \\ 1 - x_3/3 \\ x_3 \end{bmatrix}$$

The values of x_1 and x_3 are completely arbitrary. Setting $x_1 = 0$ and $x_3 = 0$ results in one possible solution of this system:

Another possible solution is obtained using singular value decomposition and results in the minimal norm condition. The minimal norm solution for this system is:

$$x = \begin{bmatrix} -2.0 \\ 0.0 \\ 1.0 \\ 0.0 \end{bmatrix}$$

$$x = \begin{bmatrix} -0.211009 \\ -0.633027 \\ 0.963303 \\ 0.110092 \end{bmatrix}$$

Note that this vector also satisfies the solution *x* as it is expressed in terms of x_1 and x_3.

The following IDL commands use singular value decomposition to find the minimal norm solution. Begin with the array *A*:

```
A = [[ 1.0, 3.0, 3.0, 2.0], $
     [ 2.0, 6.0, 9.0, 5.0], $
     [-1.0, -3.0, 3.0, 0.0]]
; Define the right-hand side vector B:
B = [1.0, 5.0, 5.0]
; Compute the decomposition of A:
SVDC, A, W, U, V
```

Create a diagonal array WP of reciprocal singular values from the output vector W. To avoid overflow errors when the reciprocal values are calculated, only elements with absolute values greater than or equal to 1.0×10^{-5} are reciprocated.

```
N = N_ELEMENTS(W)
WP = FLTARR(N, N)
FOR K = 0, N-1 DO $
   IF ABS(W(K)) GE 1.0e-5 THEN WP(K, K) = 1.0/W(K)
```

We can now express the solution to the linear system as a array-vector product. (See Section 2.6 of *Numerical Recipes* for a derivation of this formula.) The solution is expressed in terms of x_1 and x_3 with minimal norm.

```
X = V ## WP ## TRANSPOSE(U) ## B
;Print the solution:
PRINT, X
```

IDL Prints:

```
-0.211009
-0.633027
 0.963303
```

```
0.110092
```

Complex Linear Systems

We can use IDL's LU_COMPLEX function to compute the solution to a linear system with real and complex coefficients. Suppose we wish to solve the following linear system:

$$\begin{bmatrix} -1+0i & 1-3i & 2+0i & 3+3i \\ -2+0i & -1+3i & 0+1i & 3+1i \\ 3+0i & 0+4i & 0-1i & 0-3i \\ 2+0i & 1+1i & 2+1i & 2+1i \end{bmatrix} \begin{bmatrix} z_0 \\ z_1 \\ z_2 \\ z_3 \end{bmatrix} = \begin{bmatrix} 15-2i \\ -2-1i \\ -20+11i \\ -10+10i \end{bmatrix}$$

```
;First we define the real part of the complex coefficient array:
re = [[-1, 1, 2, 3], $
      [-2, -1, 0, 3], $
      [3, 0, 0, 0], $
      [2, 1, 2, 2]]
;Next, we define the imaginary part of the coefficient array:
im = [[0, -3, 0, 3], $
      [0, 3, 1, 1], $

      [0, 4, -1, -3], $
      [0, 1, 1, 1]]
; Combine the real and imaginary parts to form
; a single complex coefficient array:
A = COMPLEX(re, im)
; Define the right-hand side vector B:
B = [COMPLEX(15,-2), COMPLEX(-2,-1), COMPLEX(-20,11), $
   COMPLEX(-10,10)
; Compute the solution using double-precision complex arithmetic:
Z = LU_COMPLEX(A, B, /DOUBLE)
PRINT, TRANSPOSE(Z), FORMAT = '(f5.2, ",", f5.2, "i")'
```

IDL prints:

```
-4.00, 1.00i
 2.00, 2.00i
 0.00, 3.00i
-0.00,-1.00i
```

We can check the accuracy of the computed solution by computing the residual, Az–b:

```
PRINT, A##Z-B
```

IDL prints:

```
(      0.00000,      0.00000)
(      0.00000,      0.00000)
(      0.00000,      0.00000)
(      0.00000,      0.00000)
```

Routines for Solving Simultaneous Linear Equations

See “Linear Systems” (in the functional category “Mathematics” in the *IDL Quick Reference* manual) for a brief description of IDL routines for solving simultaneous linear equations. Detailed information is available in the *IDL Reference Guide*.

Nonlinear Equations

The problem of finding a solution to a system of *n* nonlinear equations, F(*x*) = 0, may be stated as follows:

given F: $R^n \rightarrow R^n$, find x_* (an element of R^n) such that $F(x_*) = 0$

For example:

$$F(x) = \begin{bmatrix} x_0 + x_1 - 3 \\ x_0^2 + x_1^2 - 9 \end{bmatrix}$$

$x_* = [0, 3]$ or $x_* = [3, 0]$

Note

A solution to a system of nonlinear equations is not necessarily unique.

The most powerful and successful numerical methods for solving systems of nonlinear equations are loosely based upon a simple two-step iterative method frequently referred to as Newton's method. This method begins with an initial guess and constructs a solution by iteratively approximating the *n*-dimensional nonlinear system of equations with an *n*-by-*n* linear system of equations.

The first step formulates an *n*-by-*n* linear system of equations (J*s* = – F) where the coefficient array J is the Jacobian (the array of first partial derivatives of F), *s* is a solution vector, and – F is the negative of the nonlinear system of equations. Both J and – F are evaluated at the current value of the *n*-element vector *x*.

$J(x_k)\ s_k = -F(x_k)$

The second step uses the solution s_k of the linear system as a directional update to the current approximate solution x_k of the nonlinear system of equations. The next approximate solution x_{k+1} is a linear combination of the current approximate solution x_k and the directional update s_k.

$x_{k+1} = x_k + s_k$

The success of Newton's method relies primarily on providing an initial guess close to a solution of the nonlinear system of equations. In practice this proves to be quite difficult and severely limits the application of this simple two-step method.

IDL provides two algorithms that are designed to overcome the restriction that the initial guess be close to a solution. These algorithms implement a line search which checks, and if necessary modifies, the course of the algorithm at each step ensuring

progress toward a solution of the nonlinear system of equations. IDL's NEWTON and BROYDEN functions are among a class of algorithms known as quasi-Newton methods.

The solution of an *n*-dimensional system of nonlinear equations, $F(x) = 0$, is often considered a root of that system. As a one-dimensional counterpart to NEWTON and BROYDEN, IDL provides the FX_ROOT and FZ_ROOTS functions.

Routines for Solving Nonlinear Equations

See "Nonlinear Equations" (in the functional category "Mathematics" in the *IDL Quick Reference* manual) for a brief description of IDL routines for solving systems of nonlinear equations. Detailed information is available in the *IDL Reference Guide*.

Optimization

The problem of finding an unconstrained minimizer of an *n*-dimensional function, *f*, may be stated as follows:

given f: $R^n \rightarrow R$, find x_* (an element of R^n) such that $f(x_*)$ is a minimum of f.

For example:

$f(x) = (x_0 - 3)^4 + (x_1 - 2)^2$

$x_* = [3, 2]$

In minimizing an *n*-dimensional function *f*, it is a necessary condition that the gradient at the minimizer x_*, $\nabla f(x_*)$, be the zero vector. Mathematically expressing this condition defines the following system of nonlinear equations.

$$\begin{bmatrix} \frac{\partial f(x)}{\partial x_0} \\ \frac{\partial f(x)}{\partial x_1} \\ \cdots \\ \frac{\partial f(x)}{\partial x_{n-1}} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \cdots \\ 0 \end{bmatrix}$$

This relation might suggest that finding a minimizer is equivalent to solving a system of linear equations based on the gradient. In most cases, however, this is not true. It is just as likely that a solution, x_*, of $\nabla f(x)=0$ be a maximizer or a local minimizer of *f*. Thus the gradient alone does not provide sufficient information in determining the role of x_*.

IDL provides two algorithms that do sufficiently determine the global minimizer of an n-dimensional function. IDL's DFPMIN routine is among a class of algorithms known as variable metric methods and requires a user-supplied analytic gradient of the function to be minimized. IDL's POWELL routine implements a direction-set method that does not require a user-supplied analytic gradient. The utility of the POWELL routine is evident as the function to be minimized becomes more complicated and partial derivatives become more difficult to calculate.

Routines for Optimization

See "Optimization" (in the functional category "Mathematics" in the *IDL Quick Reference* manual) for a brief description of IDL routines for optimization. Detailed information is available in the *IDL Reference Guide*.

Sparse Arrays

The occurrence of zero elements in a large array is both a computational and storage inconvenience. An array in which a large percentage of elements are zeros is referred to as being *sparse*.

Because standard linear algebra techniques are highly inefficient when dealing with sparse arrays, IDL incorporates a collection of routines designed to handle them effectively. These routines use the row-indexed sparse storage method, which stores the array in structure form, as a vector of data and a vector of indices. The length of each vector is equal to 1 plus the number of diagonal elements of the array plus the number of off-diagonal elements with an absolute magnitude greater than or equal to a specified threshold value. Diagonal elements of the array are always retained even if their absolute magnitude is less than the specified threshold. Sparse array routines that handle array-vector and array-array multiplication, file input/output, and the solution of systems of simultaneous linear equations are included.

Note

For more information on IDL's sparse array storage method, see section 2.7, "Sparse Linear Systems," in *Numerical Recipes in C: The Art of Scientific Computing* (Second Edition), published by Cambridge University Press.

When considering using IDL's sparse array routines, remember that the computational savings gained by working in sparse storage format is at least partially offset by the need to first convert the arrays to that format. Although an absolute determination of when to use sparse format is not possible, the example below demonstrates the time savings when solving a 500 by 500 linear system in which approximately 50% of the coefficient array's elements as zeros.

Diagonally-Dominant Array

Create a 500-by-500 element pseudo-random diagonally-dominant floating-point array in which approximately 50% of the elements as zeros. (In a diagonally-dominant array, the diagonal element in a given row is greater than the sum of the absolute values of the non-diagonal elements in that row.)

```
N = 500L
A = RANDOMN(SEED, N, N)*10
; Set elements with absolute magnitude greater than or
; equal to eight to zero:
I = WHERE(ABS(A) GE 8)
A[I] = 0.0
; Set each diagonal element to the absolute sum of
```

```
; its row elements plus 1.0:
diag = TOTAL(ABS(A), 1)
A(INDGEN(N) * (N+1)) = diag + 1.0
; Create a right-hand side vector, b, in which 40% of
; the elements are ones and 60% are twos.
B = [REPLICATE(1.0, 0.4*N), REPLICATE(2.0, 0.6*N)]
```

We now calculate a solution to this system using two different methods, measuring the time elapsed. First, we compute the solution using the iterative biconjugate gradient method and a sparse array storage format. Note that we include everything between the start and stop timer commands as a single operation, so that only computation time (as opposed to typing time) is recorded.

```
; Begin with an initial guess:
X = REPLICATE(1.0, N_ELEMENTS(B))
; Start the timer:
start = SYSTIME(1) & $
; Solve the system:
result1 = LINBCG(SPRSIN(A), B, X) & $
; Stop the timer.
stop = SYSTIME(1)
; Print the time taken, in seconds:
PRINT, 'Time for Iterative Biconjugate Gradient:', stop-start
```

IDL prints:

```
Time for Iterative Biconjugate Gradient       1.1259040
```

Remember that your result will depend on your hardware configuration.

Next, we compute the solution using LU decomposition.

```
; Start the timer:
start = SYSTIME(1) & $
; Compute the LU decomposition of A:
LUDC, A, index & $
; Compute the solution:
result2 = LUSOL(A, index, B) & $
; Stop the timer:
stop = SYSTIME(1)
; Print the time taken, in seconds:
PRINT, 'Time for LU Decomposition:', stop-start
```

IDL prints:

```
Time for LU decomposition       14.871168
```

Finally, we can compare the absolute error between result1 and result2. The following commands will print the indices of any elements of the two results that differ by more than 1.0×10^{-5}, or a –1 if the two results are identical to within five decimal places.

```
error = ABS(result1-result2)
PRINT, WHERE(error GT 1.0e-5)
```

IDL prints:

```
-1
```

See the documentation for the WTN function for an example using IDL's sparse array functions with image data.

Note

The times shown here were recorded on a DEC 3000 Alpha workstation running OSF/1; they are shown as examples only. Your times will depend on your specific computing platform.

Routines for Handling Sparse Arrays

See "Sparse Arrays" (in the functional category "Mathematics" in the *IDL Quick Reference* manual) for a brief description of IDL routines for handling sparse arrays. More detailed information is available in the *IDL Reference Guide*.

Time-Series Analysis

A time-series is a sequential collection of data observations indexed over time. In most cases, the observed data is continuous and is recorded at a discrete and finite set of equally-spaced points. An *n*-element time-series is denoted as $x = (x_0, x_1, x_2, \ldots, x_{n-1})$, where the time-indexed distance between any two successive observations is referred to as the sampling interval.

A widely held theory assumes that a time-series is comprised of four components:

- A trend or long term movement.
- A cyclical fluctuation about the trend.
- A pronounced seasonal effect.
- A residual, irregular, or random effect.

Collectively, these components make the analysis of a time-series a far more challenging task than just fitting a linear or nonlinear regression model. Adjacent observations are unlikely to be independent of one another. Clusters of observations are frequently correlated with increasing strength as the time intervals between them become shorter. Often the analysis is a multi-step process involving graphical and numerical methods.

The first step in the analysis of a time-series is the transformation to stationary series. A stationary series exhibits statistical properties that are unchanged as the period of observation is moved forward or backward in time. Specifically, the mean and variance of a stationary time-series remain fixed in time. The sample autocorrelation function is a commonly used tool in determining the stationarity of a time-series. The autocorrelation of a time-series measures the dependence between observations as a function of their time differences or lag. A plot of the sample autocorrelation coefficients against corresponding lags can be very helpful in determining the stationarity of a time-series.

For example, suppose the IDL variable *X* contains time-series data:

```
X = [5.44, 6.38, 5.43, 5.22, 5.28, $
     5.21, 5.23, 4.33, 5.58, 6.18, $
     6.16, 6.07, 6.56, 5.93, 5.70, $
     5.36, 5.17, 5.35, 5.61, 5.83, $
     5.29, 5.58, 4.77, 5.17, 5.33]
```

The following IDL commands plot both the time-series data and the sample autocorrelation versus the lags.

```
; Set the plotting window to hold two plots and plot the data:
IPLOT, X, VIEW_GRID=[1,2]
```

Compute the sample autocorrelation function for time lagged values 0 – 20 and plot.

```
lag = INDGEN(21)
result = A_CORRELATE(X, lag)
IPLOT, lag, result, /VIEW_NEXT
; Add a reference line at zero:
IPLOT, [0,20], [0,0], /OVERPLOT
```

The following figure shows the resulting graphs.

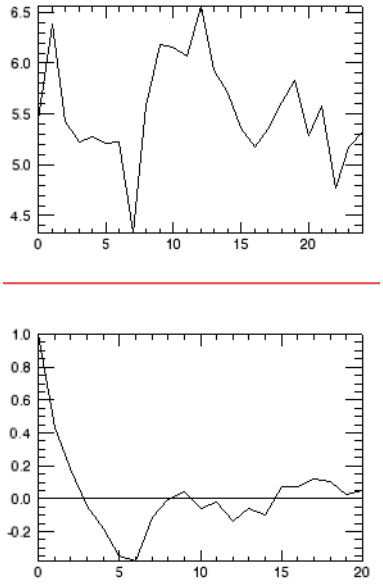


Figure 11-3: Time-series data (Top) and Autocorrelation of that Data Versus the Lag (Bottom)

The top graph plots time-series data. The bottom graph plots the autocorrelation of that data versus the lag. Because the time-series has a significant autocorrelation up to a lag of seven, it must be considered non-stationary.

Nonstationary components of a time-series may be eliminated in a variety of ways. Two frequently used methods are known as moving averages and forward differencing. The method of moving averages dampens fluctuations in a time-series

by taking successive averages of groups of observations. Each successive overlapping sequence of k observations in the series is replaced by the mean of that sequence. The method of forward differencing replaces each time-series observation with the difference of the current observation and its adjacent observation one step forward in time. Differencing may be computed recursively to eliminate more complex nonstationary components.

Once a time-series has been transformed to stationarity, it may be modeled using an autoregressive process. An autoregressive process expresses the current observation, x_t, as a combination of past time-series values and residual white noise. The simplest case is known as a first order autoregressive model and is expressed as

$$x_t = \phi x_{t-1} + \omega_t$$

The coefficient ϕ is estimated using the time-series data. The general autoregressive model of order p is expressed as

$$x_t = \phi_1 x_{t-1} + \phi_2 x_{t-2} + \ldots + \phi_p x_{t-p} + \omega_t$$

Modeling a stationary time-series as a p-th order autoregressive process allows the extrapolation of data for future values of time. This process is know as forecasting.

Routines for Time-Series Analysis

See "Time-Series Analysis" (in the functional category "Mathematics" in the *IDL Quick Reference* manual) for a brief description of IDL routines for time-series analysis. Detailed information is available in the *IDL Reference Guide*.

Multivariate Analysis

IDL provides a number of tools for analyzing multivariate data. These tools are broadly grouped into two categories: Cluster Analysis and Principal Components Analysis.

Cluster Analysis

Cluster Analysis attempts to construct a sensible and informative classification of an initially unclassified sample population using a set of common variables for each individual. The clusters are constructed so as to group samples with the similar features, based upon a set of variables. The samples (contained in the rows of an input array) are each assigned a cluster number based upon the values of their corresponding variables (contained in the columns of an input array).

In computing a cluster analysis, a predetermined number of cluster centers are formed and then each sample is assigned to the unique cluster which minimizes a distance criterion based upon the variables of the data. Given an m-column, n-row array, IDL's CLUST_WTS and CLUSTER functions compute n cluster centers and n clusters, respectively. Conceivably, some clusters will contain multiple samples while other clusters will contain none. The choice of clusters is arbitrary; in general, however, the user will want to specify a number less than the default (the number of rows in the input array). The cluster number (the number that identifies the cluster group) assigned to a particular sample or group of samples is not necessarily unique.

It is possible that not all variables play an equal role in the classification process. In this situation, greater or lesser importance may be given to each variable using the VARIABLE_WTS keyword to the CLUST_WTS function. The default behavior is to assume all variables contained in the data array are of equal importance.

Under certain circumstances, a classification of variables may be desired. The CLUST_WTS and CLUSTER functions provide this functionality by first transposing the *m*-column, *n*-row input array using the TRANSPOSE function and then interchanging the roles of variables and samples.

Example of Cluster Analysis

Define an array with 5 variables (columns) and 9 samples (rows):

```
array = [[ 99,  79,  63,  87, 249 ], $
         [ 67,  41,  36,  51, 114 ], $
         [ 67,  41,  36,  51, 114 ], $
         [ 94, 191, 160, 173, 124 ], $
         [ 42, 108,  37,  51,  41 ], $
```

```
            [ 67,  41,  36,  51, 114 ], $
            [ 94, 191, 160, 173, 124 ], $
            [ 99,  79,  63,  87, 249 ], $
            [ 67,  41,  36,  51, 114 ]]
; Compute the cluster weights with four cluster centers:
weights = CLUST_WTS(array, N_CLUSTERS = 4)
; Compute the cluster assignments, for each sample,
; into one of four clusters:
result  = CLUSTER(array, weights, N_CLUSTERS = 4)
; Display the cluster assignment and corresponding sample (row):
FOR k = 0, 8 DO $
   PRINT, result[k], array[*, k]
```

IDL prints:

```
1       99       79       63       87      249
3       67       41       36       51      114
3       67       41       36       51      114
0       94      191      160      173      124
2       42      108       37       51       41
3       67       41       36       51      114
0       94      191      160      173      124
1       99       79       63       87      249
3       67       41       36       51      114
```

Samples 0 and 7 contain identical data and are assigned to cluster #1. Samples 1, 2, 5, and 8 contain identical data and are assigned to cluster #3. Samples 3 and 6 contain identical data and are assigned to cluster #0. Sample 4 is unique and is assigned to cluster #2.

If this example is run several times, each time computing new cluster weights, it is possible that the cluster number assigned to each grouping of samples may change.

Principal Components Analysis

Principal components analysis is a mathematical technique which describes a multivariate set of data using derived variables. The derived variables are formulated using specific linear combinations of the original variables. The derived variables are uncorrelated and are computed in decreasing order of importance; the first variable accounts for as much as possible of the variation in the original data, the second variable accounts for the second largest portion of the variation in the original data, and so on. Principal components analysis attempts to construct a small set of derived variables which summarize the original data, thereby reducing the dimensionality of the original data.

The principal components of a multivariate set of data are computed from the eigenvalues and eigenvectors of either the sample correlation or sample covariance

matrix. If the variables of the multivariate data are measured in widely differing units (large variations in magnitude), it is usually best to use the sample correlation matrix in computing the principal components; this is the default method used in IDL's PCOMP function.

Another alternative is to standardize the variables of the multivariate data prior to computing principal components. Standardizing the variables essentially makes them all equally important by creating new variables that each have a mean of zero and a variance of one. Proceeding in this way allows the principal components to be computed from the sample covariance matrix. IDL's PCOMP function includes COVARIANCE and STANDARDIZE keywords to provide this functionality.

For example, suppose that we wish to restate the following data using its principal components. There are three variables, each consisting of five samples.

	Var 1	Var 2	Var 3
Sample 1	2.0	1.0	3.0
Sample 2	4.0	2.0	3.0
Sample 3	4.0	1.0	0.0
Sample 4	2.0	3.0	3.0
Sample 5	5.0	1.0	9.0

Table 11-1: Data for Principal Component Analysis

We compute the principal components (the coefficients of the derived variables) to 2 decimal accuracy and store them by row in the following array.

$$\begin{bmatrix} 0.87 & -0.70 & 0.69 \\ 0.01 & -0.64 & -0.66 \\ 0.49 & 0.32 & -0.30 \end{bmatrix}$$

The derived variables $\{z_1, z_2, z_3\}$ are then computed as follows:

$$z1 = (0.87)\begin{bmatrix}2.0\\4.0\\4.0\\2.0\\5.0\end{bmatrix} + (-0.70)\begin{bmatrix}1.0\\2.0\\1.0\\3.0\\1.0\end{bmatrix} + (\ 0.69)\begin{bmatrix}3.0\\3.0\\0.0\\3.0\\9.0\end{bmatrix}$$

$$z2 = (0.01)\begin{bmatrix}2.0\\4.0\\4.0\\2.0\\5.0\end{bmatrix} + (-0.64)\begin{bmatrix}1.0\\2.0\\1.0\\3.0\\1.0\end{bmatrix} + (-0.66)\begin{bmatrix}3.0\\3.0\\0.0\\3.0\\9.0\end{bmatrix}$$

$$z3 = (0.49)\begin{bmatrix}2.0\\4.0\\4.0\\2.0\\5.0\end{bmatrix} + (\ 0.32)\begin{bmatrix}1.0\\2.0\\1.0\\3.0\\1.0\end{bmatrix} + (-0.30)\begin{bmatrix}3.0\\3.0\\0.0\\3.0\\9.0\end{bmatrix}$$

In this example, analysis shows that the derived variable z_1 accounts for 57.3% of the total variance of the original data, the derived variable z_2 accounts for 28.2% of the total variance of the original data, and the derived variable z_3 accounts for 14.5% of the total variance of the original data.

Example of Derived Variables from Principal Components

The following example constructs an appropriate set of derived variables, based upon the principal components of the original data, which may be used to reduce the dimensionality of the data. The data consist of four variables, each containing of twenty samples.

```
; Define an array with 4 variables and 20 samples:
data = [[19.5, 43.1, 29.1, 11.9], $
        [24.7, 49.8, 28.2, 22.8], $
        [30.7, 51.9, 37.0, 18.7], $
        [29.8, 54.3, 31.1, 20.1], $
        [19.1, 42.2, 30.9, 12.9], $
```

```
    [25.6, 53.9, 23.7, 21.7], $
    [31.4, 58.5, 27.6, 27.1], $
    [27.9, 52.1, 30.6, 25.4], $
    [22.1, 49.9, 23.2, 21.3], $
    [25.5, 53.5, 24.8, 19.3], $
    [31.1, 56.6, 30.0, 25.4], $
    [30.4, 56.7, 28.3, 27.2], $
    [18.7, 46.5, 23.0, 11.7], $
    [19.7, 44.2, 28.6, 17.8], $
    [14.6, 42.7, 21.3, 12.8], $
    [29.5, 54.4, 30.1, 23.9], $
    [27.7, 55.3, 25.7, 22.6], $
    [30.2, 58.6, 24.6, 25.4], $
    [22.7, 48.2, 27.1, 14.8], $
    [25.2, 51.0, 27.5, 21.1]]
```

The variables that will contain the values returned by the COEFFICIENTS, EIGENVALUES, and VARIANCES keywords to the PCOMP routine must be initialized as nonzero values prior to calling PCOMP.

```
coef = 1 & eval = 1 & var = 1
; Compute the derived variables based upon
; the principal components.
result = PCOMP(data, COEFFICIENTS = coef, $
   EIGENVALUES = eval, VARIANCES = var)
; Display the array of derived variables:
PRINT, result, FORMAT = '(4(f5.1, 2x))'
```

IDL prints:

```
 81.4   15.5   -5.5    0.5
102.7   11.1   -4.1    0.6
109.9   20.3   -6.2    0.5
110.5   13.8   -6.3    0.6
 81.8   17.1   -4.9    0.6
104.8    6.2   -5.4    0.6
121.3    8.1   -5.2    0.6
111.3   12.6   -4.0    0.6
 97.0    6.4   -4.4    0.6
102.5    7.8   -6.1    0.6
118.5   11.2   -5.3    0.6
118.9    9.1   -4.7    0.6
 81.5    8.8   -6.3    0.6
 88.0   13.4   -3.9    0.6
 74.3    7.5   -4.8    0.6
113.4   12.0   -5.1    0.6
109.7    7.7   -5.6    0.6
117.5    5.5   -5.7    0.6
 91.4   12.0   -6.1    0.6
102.5   10.6   -4.9    0.6
```

Display the percentage of total variance for each derived variable:

```
PRINT, var
```

IDL prints:

```
0.712422
0.250319
0.0370950
0.000164269
```

Display the percentage of variance for the first two derived variables; the first two columns of the resulting array above.

```
PRINT, TOTAL(var[0:1])
```

IDL prints:

```
0.962741
```

This indicates that the first two derived variables (the first two columns of the resulting array) account for 96.3% of the total variance of the original data, and thus could be used to summarize the original data.

Routines for Multivariate Analysis

See "Multivariate Analysis" (in the functional category "Mathematics" in the *IDL Quick Reference* manual) for a brief description of IDL routines for multivariate analysis. Detailed information is available in the *IDL Reference Guide*.

References

Correlation Analysis

Harnet, Donald L. *Introduction to Statistical Methods*. Reading, Massachusetts: Addison-Wesley, 1975. ISBN 0-201-02752-6

Neter, John., William Wasserman, and G.A. Whitmore. *Applied Statistics*. Newton, Massachusetts: Allyn and Bacon, 1988. ISBN 0-205-10328-6

Press, William H. *et al*. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge: Cambridge University Press, 1992. ISBN 0-521-43108-5

Curve and Surface Fitting

Bevington, Philip R. *Data Reduction and Error Analysis for the Physical Sciences*. New York: McGraw-Hill, 1969.

Lancaster, Peter and Kestutis Salkauskas. *Curve and Surface Fitting* (*An Introduction*). San Diego: Academic Press, 1986. ISBN 0-124-36060-0

Eigenvalues and Eigenvectors

Press, William H. *et al*. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge: Cambridge University Press, 1992. ISBN 0-521-43108-5

Strang, Gilbert. *Linear Algebra and Its Applications*. San Diego: Harcourt Brace Jovanovich, 1988. ISBN 0-155-551005-3

Gridding and Interpolation

Lancaster, Peter and Kestutis Salkauskas. *Curve and Surface Fitting (An Introduction)*. San Diego: Academic Press, 1986. ISBN 0-124-36060-0

Press, William H. *et al*. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge: Cambridge University Press, 1992. ISBN 0-521-43108-5

Hypothesis Testing

Harnett, Donald H. *Introduction to Statistical Methods*. Reading, Massachusetts: Addison-Wesley, 1975. ISBN 0-201-02752-6

Kraft, Charles H. and Constance Van Eeden. *A Nonparametric Introduction to Statistics*. New York: Macmillan, 1968.

Sprent, Peter. *Applied Nonparametric Statistical Methods*. London: Chapman and Hall, 1989. ISBN 0-412-30600-X

Integration

Chapra, Steven C. and Raymond P. Canale. *Numerical Methods for Engineers*. New York: McGraw-Hill, 1988. ISBN 0-070-79984-9

Press, William H. *et al*. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge: Cambridge University Press, 1992. ISBN 0-521-43108-5

Linear Systems

Golub, Gene H. and Van Loan, Charles F. *Matrix Computations*. Baltimore: Johns Hopkins University Press, 1989. ISBN 0-8018-3772-3

Kreyszig, Erwin. *Advanced Engineering Mathematics*. New York: Wiley & Sons, Inc., 1993. ISBN 0-471-55380-8

Press, William H. *et al*. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge: Cambridge University Press, 1992. ISBN 0-521-43108-5

Strang, Gilbert. *Linear Algebra and Its Applications*. San Diego: Harcourt Brace Jovanovich, 1988. ISBN 0-155-551005-3

Nonlinear Equations

Dennis, J.E. Jr. and Robert B. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Englewood Cliffs, NJ: Prentice-Hall, 1983. ISBN 0-136-27216-9

Press, William H. *et al*. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge: Cambridge University Press, 1992. ISBN 0-521-43108-5

Optimization

Dennis, J.E. Jr. and Robert B. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Englewood Cliffs, NJ: Prentice-Hall, 1983. ISBN 0-136-27216-9

Press, William H. *et al*. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge: Cambridge University Press, 1992. ISBN 0-521-43108-5

Sparse Arrays

Press, William H. *et al*. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge: Cambridge University Press, 1992. ISBN 0-521-43108-5

Time-Series Analysis

Chatfield, C. *The Analysis of Time Series*. London: Chapman and Hall, 1975. ISBN 0-412-31820-2

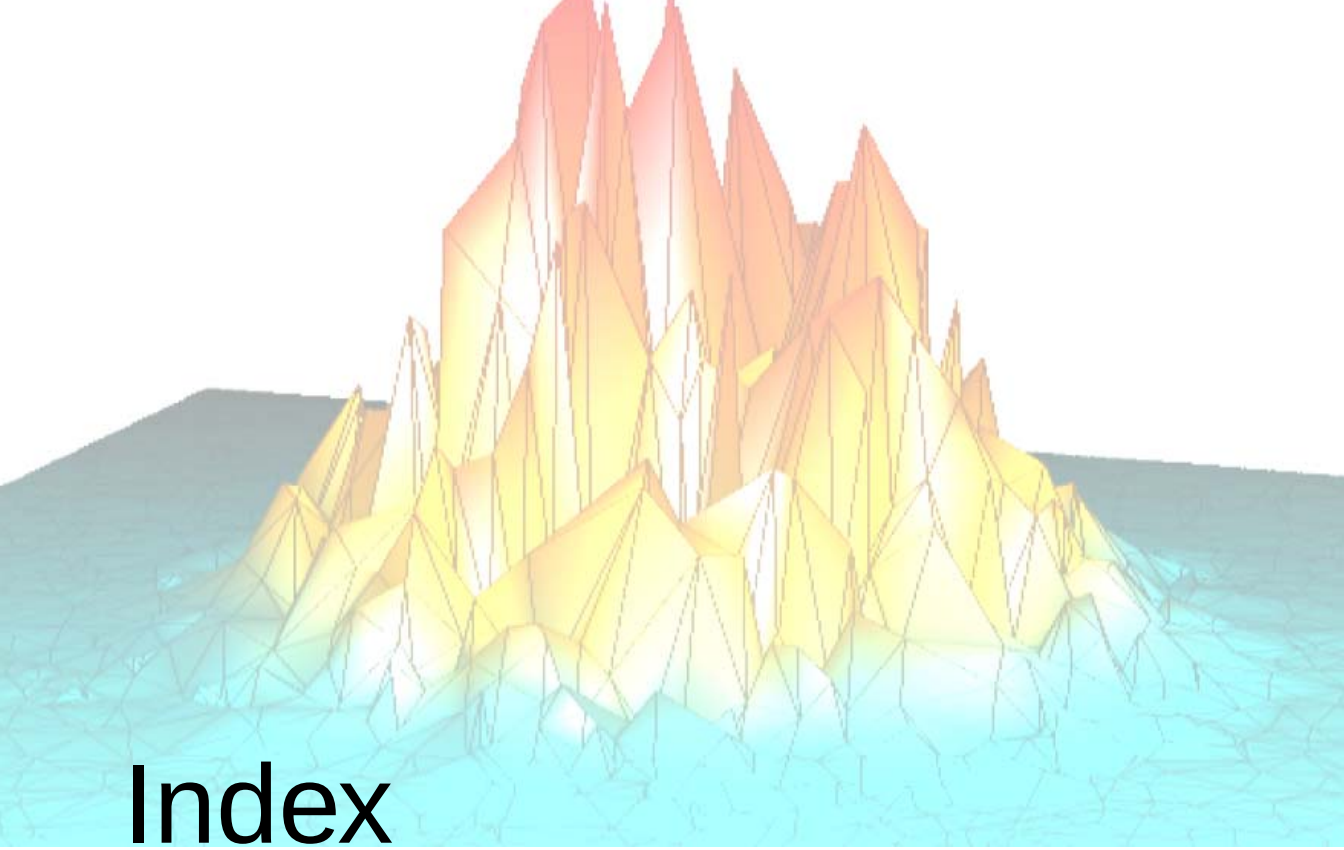
Neter, John., William Wasserman, and G.A. Whitmore. *Applied Statistics*. Newton, Massachusetts: Allyn and Bacon, 1988. ISBN 0-205-10328-6

Multivariate Analysis

Jackson, Barbara Bund. *Multivariate Data Analysis*. Homewood, Illinois: R.D. Irwin, 1983. ISBN 0-256-02848-6

Everitt, Brian S. *Cluster Analysis*. New York: Halsted Press, 1993. ISBN 0-470-22043-0

Kachigan, Sam Kash. *Multivariate Statistical Analysis*. New York: Radius Press, 1991. ISBN 0-942154-91-6



Index

Symbols

!ORDER system variable, 213

A

accuracy
  numerical algorithms, 280
action routines, 141
Aitoff map projection, 232
Albers equal-area conic projection, 241
aliasing, 263
analytic signal, 265
ARMA filter, 275
arrays
  data type, determining type
    SIZE function, 179
  rotating, 197
  sparse, 313
  stored in structure form, 313
ASCII files
  IDLDE import macro, 167
  reading, 153
autoregressive moving average filters, 275
azimuthal equidistant map projection, 231
azimuthal map projections, 228

B

backing store
  bitmap buffered, 105
  graphics, 105
  system buffered, 105
bandpass
  filters, 271

bandstop filters, 271
batch files
  startup preference, 111
bilinear
  interpolation, 202
  transform, 275
binary files
  IDLDE import macro, 169
  reading, 154
bitmap buffered backing store, 105
boxcar filter, 273
Bristol Technology
  printer manager, 81
  printing graphics, 222

C

central map projection, 230
CIA World Map database, 246
clipboard support
  graphics windows, 56
cluster analysis
  routines, 324
CMY color system, 204
color
  channels, 214
  Direct Graphics, 211
  images
    Direct Graphics, 211
  systems
    CMY, 204
    converting, 206
    HLS, 204
    HSV, 204
    RGB, 204
  tables. *See* color tables
  visuals
    Unix, 207
    Windows, 208
color tables
  highlighting image features, 220
  indexed image (LUT), 214
  modifying, 219
colormaps, 210
  flashing, 134
  sharing (Motif), 134
colors
  reserving for IDL, 134
Command Line
  IDLDE, 57
command line options
  Motif platform, 136
command line switches, 23
compiling
  from memory preference, 108
  preferences, 107, 108
conformal conic map projection, 240
Control Panel Buttons
  modifying in Motif, 138
  Motif platform, 58
converting
  color systems, 206
  color tables, 219
  image types, 217
Cooley-Tukey algorithm, 264
coordinate systems
  device, 194
  normalized, 194
  window, 193
coordinates
  converting
    three-dimensional coordinates, 199
  converting two-dimensional coordinates, 198
  data, 193
  device, 193
  homogeneous, 195
  normal, 193
correlation
  analysis, 282
  coefficient, 282, 283
  routines, 285
cubic convolution interpolation, 202

curve fitting
  discussion, 286
  routines, 287
customizing IDL, preferences, 95
cyclical fluctuation, 316
cylindrical equidistant map projection, 239
cylindrical map projections, 237

D

data coordinates, 193
data formats
  supported, 12
data type
  type code, 179
data types
  determining array size, 179
  IDL indices, 177
  type codes, 177
Delaunay triangulation, 294
deleting
  lines in Output Log, 98
derived variables, 320
device
  coordinates, 193
  independent graphics, 191
DFT, 254
differentiation routines, 301
digital filters, 270
digital signal processing, 251
DIGITAL_FILTER function, 271
Direct Graphics, 192
  color
    indexed, 211
    RGB, 211
  printing, 222
  visuals
    Unix, 209
    Windows, 210
  window coordinates, 194
direct graphics
  clipboard support, 56
discrete Fourier transform, 254
discrete wavelet transform, 267
DISPLAY environment variable, 20
displayrgbimage_object.pro, 215
DWT, 267

E

editing
  resource files, 133
Editor window
  compiling and saving, 107
  multiple, 102
  preferences, 107
Editor windows
  defined, 55
editors, external (Motif), 127
eigenvalues
  complex, 289
  real, 288
  repeated, 291, 292
  routines for computing, 293
eigenvectors
  complex, 289
  real, 288
  repeated, 292
  routines for computing, 293
environment variables
  CLASSPATH, 20
  DISPLAY, 20, 20
  HOME, 20
  IDLJAVAB_CONFIG, 21
  IDLJAVAB_LIB_LOCATION, 21
  LM_LICENSE_FILE, 21
  PATH, 16, 21
  TERM, 21
equal-area map projection, 241
examples
  batch files
    sigprc01, 251

sigprc02, 252
sigprc03, 257
sigprc04, 258
sigprc05, 259
sigprc06, 261
sigprc07, 262
sigprc08, 263
sigprc09, 266
sigprc10, 271
sigprc11, 272
sigprc12, 273
sigprc13, 276
sigprc14, 277
image
displayrgbimage_object.pro, 215
exiting IDL
confirm exit, 97
options, 46
exporting
formatted image files, 160
unformatted image files, 161
expressions
determining data type
SIZE function, 179
external
editors (Motif), 127

F

Fast Fourier transform
Cooley-Tukey algorithm, 264
defined, 254
discrete, 254
implementation, 264
using windowing algorithms, 260
file
See also files.
access, 149
search path, 115
supported formats, 12
file access
See also reading.
about, 150
routines, 171
file formats
about supported, 12
general data, 13
image, 12
scientific data, 13
file information
returning, 174
file selection
using dialogs, 151
file types, supported, 12
FILE_INFO function
using, 188
files
See also file.
accessing, 149
exporting
See also writing.
formatted, 160
unformatted, 161
importing
See also reading.
formatted, 158
unformatted, 159
querying, 174
returning
file information, 174
specifying search path, 115
filtering
autoregressive moving average, 275
bandpass, 271
bandstop, 271
boxcar, 273
digital, 270
FIR, 271
highpass, 271
lowpass, 271
rectangular, 273
filters

IIR filter, 275
Kaiser's window, 271
moving average, 271
notch, 275
finding
text, IDLDE search features, 65
finite impulse response filters, 271
FIR filter, 271
flashing colormaps, 134
fonts
preferences, 112
specifying
Motif platform, 113
Windows platform, 112
frequency plot leakage, 258
frequency plot smearing, 258
frequency response function, 276

G

Gaussian
iterated quadrature, 297
Gauss-Krueger map projection, 238
general perspective map projection, 235
geometric transformations
interpolation methods, 201
glyph. *See* TrueType fonts
gnomic map projection, 230
gnomonic map projection, 230
Gouraud shading, 203
graphics
clipboard support, 56
coordinate systems, 195
device independent graphics, 191
devices
direct graphics, 192
image file formats
supported, 12
modes, 190
object-oriented, 191
windows
backing store, 105
layout preferences, 104
OS clipboard support, 56
sizing, 104
gridding
data extrapolation, 294
Delaunay triangulation, 294
routines, 294
uniformly-spaced planar data, 294

H

Hammer-Aitoff map projection, 234
Hamming window
defined, 261
Hanning window
defined, 260
hardware rendering, setting preference, 106
HDF files
IDLDE import macros, 170
HDF-EOS
IDLDE import macro, 170
help
PDF files
overview, 43
hiding
toolbars, 103
highlighting
image features, 220
highpass filters, 271
high-resolution continent outlines, 246
Hilbert transform, 265
histogram
plot, 252
HLS color system
color schemes, 204
HOME environment variable, 20
homogeneous coordinates, 195
HSV color system
color schemes, 204
hypothesis testing

routines, 296
statistics, 295

I

IDL
direct graphics, 192
iTools, 190
object graphics, 191
IDL GUIBuilder
access, 56
generating
files, menu option, 61
IDLDE
about, 52
preferences, 93
IIR filter
digital filtering, 275
using, 275
image files, 165
image interleaving, 214
image objects
displaying
RGB, 215
interleaving, 214
pixel interleaving, 214
images
dialog for reading, 151
dialog for saving, 152
exporting files, 160, 161
file selection
using a dialog, 151
highlighting features, 220
import macro, 165
importing files, 159
info structure, 175
orientation, 213
QUERY_IMAGE, 178
querying, 175
raster, 213
RGB interleaving, 215
import macro, IDLDE
ASCII files, 167
binary files, 169
image files, 165
scientific data formats, 170
importing
data, 149, 149
unformatted image files, 159
indexed images
color tables, 214
infinite impulse response filters, 275
integration
bivariate functions, 298
discussion, 297
numerical, 297
routines, 301
trivariate functions, 299
interleaving
determining, 215
image, 214
image objects, 214
line, 214
pixel, 214
planar, 214
interpolation
bilinear, 202
cubic convolution, 202
image quality, 201
linear, 202
methods, 202
nearest-neighbor, 202
routines, 294
tabulated data points, 294
trilinear, 202
iTools
introduction, 17

K

Kaiser filter, 271
keyboard

shortcuts, 33
using accelerators, Macintosh, 33

L

Lambert's conformal conic map projection, 240
Lambert's equal area map projection, 233
launching IDL, 15
layout, graphics window preferences, 104
leakage, 258
light source
shading, 203
line interleaving, 214
linear
algebra, 282
correlation, 282
systems
condition number, 302
overdetermined, 302
solving simultaneous equations, 302
underdetermined, 305
linear equations, simultaneous, 308
linear interpolation, 202
linear systems, routines, 308
lines
Output Log, 98
saved in recall buffer, 98
LM_LICENSE_FILE variable, 21
Look-Up Table (LUT), 214
lowpass filters, 271

M

Macintosh
one-button mouse, 32
macros
IDLDE
creating in UNIX, 121
creating in Windows, 124
pre-defined, 164
working with, 119
magnitude
signal spectra, 257
map projections
Aitoff, 232
Albers equal-area conic, 241
azimuthal, 228
azimuthal equidistant, 231
central gnomic, 230
cylindrical, 237
cylindrical equidistant, 239
general perspective, 235
gnomonic, 230
Hammer-Aitoff, 234
high-resolution outlines, 246
Lambert's conformal conic, 240
Lambert's equal area, 233
Mercator, 237
Miller cylindrical, 240
Mollweide, 243
orthographic, 229
overview, 224
pseudocylindrical, 242
Robinson, 242
satellite, 235
sinusoidal, 243
stereographic, 229
Transverse Mercator, 238
mathematics
routines, 280
memory
object graphics system, 191
optimizing Windows performance, 98
Menu Editor
opening, 64
menus
IDLDE keyboard shortcuts, 33
Mercator map projection, 237
Microsoft Windows
mouse differences, 32

Miller cylindrical map projection, 240
minimization, 311
  See also optimization
modifying color tables, 219
Mollweide map projection, 243
Motif widgets, 141
mouse
  emulating three-button, 32
moving average filter, 271
multiple correlation coefficient, 284
Multiple Document Panel, 55
multivariate analysis
  routines, 324

N

nearest-neighbor interpolation, 202
NETCDF files
  IDLDE import macro, 170
Newton's method, 309
nonlinear equations
  discussion, 309
  routines, 310
nonparametric hypothesis tests, 295
normal
  coordinates, 193
notch filter, 275
numerical integration, 297
Numerical Recipes in C, 281
Nyquist frequency, 263

O

OBJ_CLASS function
  using, 186
OBJ_ISA function
  using, 186
OBJ_VALID function
  using, 187
object graphics
  about, 191
  choosing a renderer, 106
  clipboard support, 56
  printing, 222
objects
  information about, 186
  object graphics
    clipboard support, 56
  object-oriented
    graphics, 191
Oetli, Thomas, 246
one-tailed hypothesis tests, 295
optimization
  discussion, 311
  routines, 312
origin
  image data, 213
orthographic map projection, 229
Output Log
  overview, 57
  preferences, 98

P

parametric hypothesis tests, 295
partial correlation coefficient, 284
path
  IDLDE, 115
PATH environment variable, 16
PDF, 43
performance
  improvement, 105
  optimizing memory, 98
phase
  signal spectra, 257
pixels
  data
    information (QUERY_IMAGE), 177
  interleaving, 214
  two-dimensional image arrays, 213
planar interleaving, 214

plotting
  frequency smearing, 258
  step plots, 252
Portable Document Format, 43
power spectrum, 259
preferences
  change directories, 99
  changing, 93
  read-only files, 99
  startup, 97
principal components analysis, 320
print manager, 81, 222
printing
  direct graphics
    overview, 222
  from IDLDE, 62
  graphics, 222
  in UNIX, 81
  in Windows, 80
private colormaps, 210
project
  interface, 55
projections
  Aitoff, 232
  Albers equal-area conic, 241
  azimuthal, 228
  azimuthal equidistant, 231
  central gnomic, 230
  cylindrical, 237
  cylindrical equidistant, 239
  general perspective, 235
  gnomonic, 230
  Hammer-Aitoff, 234
  high-resolution continent outlines, 246
  Lambert's conformal conic, 240
  Lambert's equal area, 233
  Mercator, 237
  Miller cylindrical, 240
  Mollweide, 243
  orthographic, 229
  projection matrix, 303
  pseudocylindrical, 242
  Robinson, 242
  satellite, 235
  sinusoidal, 243
  stereographic, 229
  Transverse Mercator, 238
Properties dialog
  opening, 64
PseudoColor visuals, 207
pseudocylindrical map projections, 242

Q

quadrature function, 265
querying
  images, 175
  structure tags, 175

R

raster images, 213
reading
  ASCII data, 153, 167
  binary data, 154, 169
  HDF files, 170
  HDF-EOS files, 170
  image files, 151, 165
  netCDF files, 170
  scientific format data, 170
recall buffer
  persistence, 99
  preferences, 98
recent
  files list, 62
  projects, 62
rectangular filter, 273
rendering
  hardware versus software, 106
replacing text, 66
resampling images

see also interpolation
reserving colors, 134
resolution of map databases, 246
resource files, 133
resources for an X Window, 132
RGB color system, 204
RGB images
displaying
Object Graphics
images
Object Graphic RGB image, 215
right-handed coordinate system, 195
Robinson map projection, 242
rotating
arrays, 197
images, 197
routines
cluster analysis, 324
correlation, 285
curve and surface fitting, 287
differentiation/integration, 301
eigenvalues/eigenvectors, 293
gridding/interpolation, 294
hypothesis testing, 296
linear systems, 308
mathematical, 280
multivariate analysis, 324
nonlinear equations, 310
optimization, 312
signal processing, 250
sparse arrays, 315
time-series analysis, 318
row-indexed sparse storage method, 313

S

sampled
data analysis, 263
images, 213
sampling
aliasing data, 263
satellite map projection, 235
saving
files, 61
image files, 152
scaling
matrices, 196
scientific data format
IDLDE import macro, 170
search path
specifying with preferences, 115
seasonal effect, 316
shading
Gouraud interpolation, 203
light source, 203
shared colormaps, 210
shared colormaps (Motif), 134
reading
See also file access.
signal
analysis transforms, 253
processing, 251
signal processing
routines, 250
sigprc01 batch file, 251
sigprc02 batch file, 252
sigprc03 batch file, 257
sigprc04 batch file, 258
sigprc05 batch file, 259
sigprc06 batch file, 261
sigprc07 batch file, 262
sigprc08 batch file, 263
sigprc09 batch file, 266
sigprc10 batch file, 271
sigprc11 batch file, 272
sigprc12 batch file, 273
sigprc13 batch file, 276
sigprc14 batch file, 277
simultaneous linear equations, 302
singular value decomposition, 302
sinusoidal map projection, 243

sizing graphics windows, 104
smearing frequency plots, 258
SMOOTH function, 273
software rendering
  setting preference for, 106
sparse arrays, 313
  routines, 315
splash screen preference, 97
standard
  data file formats, 13
  image file formats, 12
  scientific data formats, 13
standardized variables, 321
starting
  IDL, 15
startup
  working directory, 110
startup file
  batch file execution, 111
  overview, 30
startup preferences
  options, 110
  specifying, 97
stationary series, 316
statistics
  hypothesis testing, 295
  routines, 280
Status Bar, IDLDE, 58
step plot, 252
stereographic map projection, 229
structure tags
  image query, 175
structures
  arrays stored in structure form, 313
supported file formats, 12
surface fitting
  discussion, 286
  routines, 287
switches, command line, 23
system buffered backing store, 105
system variables
  !ORDER, 213

T

TERM environment variable, 21
text
  replacing, 66
  searching in IDLDE, 65
three-dimensional
  coordinate conversion, 199
  graphics, 195
  transformation
    matrices, 195
time-series analysis, 316
  routines, 318
toolbars
  IDLDE, 54, 55, 55, 57
  Motif platform, 58, 138
  show/hide preference, 103
  specifying layout, 103
transformation matrices, 195
transforms
  Fourier, 254
  Hilbert, 265
  Tustin bilinear, 275
  wavelet, 267
translation, 196
Transverse Mercator map projection, 238
trend analysis, 316
triangulation
  drawing fonts, 106
  TrueType fonts, 106
trilinear interpolation, 202
TrueColor visuals, 207, 211
TrueType fonts
  graphic preferences, 106
Tustin transform, 275
two-tailed hypothesis tests, 295
typographical conventions, 45

U

unconstrained minimizer, 311
UTM (Transverse Mercator) map projection, 238

V

Variable Watch Window
  IDLDE, 58
variables
  data type, determining
    SIZE function, 179
  derived, 320
  standardized, 320

W

wavelet transform, 267
windowing
  Hamming windowed signal, 261
windowing algorithm
  HANNING function, 260
windows
  arranging layout, 104
  clipboard support for graphics, 56
  separating the IDLDE, 103
  show/hide preference
    Microsoft Windows platform, 102
    Motif platform, 102
working directory, changing on startup, 110
writing
  image files, 152

X

X resources
  using, 132
Xprinter
  defined, 81
  printing graphics, 222