Using IDL
Restricted Rights Notice

The IDL® software program and the accompanying procedures, functions, and documentation described herein are sold under license agreement. Their use, duplication, and disclosure are subject to the restrictions stated in the license agreement.

Limitation of Warranty

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

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

Most Current Documentation

Because changes may be made to IDL after documentation has gone to press, please consult IDL's hypertext online help system for the most current version of this document.

Permission to Reproduce this Manual

Purchasers of IDL licenses are given limited permission to reproduce this manual provided such copies are for their use only and are not sold or distributed to third parties. All such copies must contain the title page and this notice page in their entirety.

Acknowledgments

IDL® is a trademark of Research Systems Inc., registered in the United States Patent and Trademark Office, for the computer program described herein. All other brand or product names are trademarks of their respective holders.

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

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

Portions of this software are copyrighted by INTERSOLV, Inc., 1991-1998.

IDL documentation is printed on recycled paper. Our paper has a minimum 20% post-consumer waste content and meets all EPA guidelines.
# Contents

Chapter 1:  
**Overview** ......................................................... 11  
  About IDL ................................................................. 11  
  IDL Documentation .................................................. 12  
  Typographical Conventions ....................................... 14  
  Reporting Problems .................................................. 15  

Chapter 2:  
**Running IDL** ...................................................... 19  
  Starting IDL ............................................................... 20  
  Quitting IDL .............................................................. 21  
  Environment Variables Used by IDL ............................ 22  
  Input to IDL ............................................................... 24  
  Special Characters .................................................... 26
<table>
<thead>
<tr>
<th>Using IDL</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customizing IDL</td>
<td>128</td>
</tr>
<tr>
<td>Macintosh IDL Differences</td>
<td>134</td>
</tr>
<tr>
<td>Chapter 6:</td>
<td></td>
</tr>
<tr>
<td>Programming Tools</td>
<td>137</td>
</tr>
<tr>
<td>Debugging in IDL</td>
<td>138</td>
</tr>
<tr>
<td>Debugging Commands</td>
<td>138</td>
</tr>
<tr>
<td>The IDL Code Profiler</td>
<td>141</td>
</tr>
<tr>
<td>The Variable Watch Window</td>
<td>145</td>
</tr>
<tr>
<td>Chapter 7:</td>
<td></td>
</tr>
<tr>
<td>Using the IDL GUIBuilder</td>
<td>149</td>
</tr>
<tr>
<td>Starting the IDL GUIBuilder</td>
<td>151</td>
</tr>
<tr>
<td>Creating an Example Application</td>
<td>153</td>
</tr>
<tr>
<td>IDL GUIBuilder Tools</td>
<td>164</td>
</tr>
<tr>
<td>Widget Operations</td>
<td>176</td>
</tr>
<tr>
<td>Generating Files</td>
<td>179</td>
</tr>
<tr>
<td>IDL GUIBuilder Examples</td>
<td>181</td>
</tr>
<tr>
<td>Widget Properties</td>
<td>196</td>
</tr>
<tr>
<td>Common Widget Properties</td>
<td>196</td>
</tr>
<tr>
<td>Base Widget Properties</td>
<td>202</td>
</tr>
<tr>
<td>Button Widget Properties</td>
<td>212</td>
</tr>
<tr>
<td>Text Widget Properties</td>
<td>216</td>
</tr>
<tr>
<td>Label Widget Properties</td>
<td>221</td>
</tr>
<tr>
<td>Slider Widget Properties</td>
<td>222</td>
</tr>
<tr>
<td>Droplist Widget Properties</td>
<td>224</td>
</tr>
<tr>
<td>Listbox Widget Properties</td>
<td>226</td>
</tr>
<tr>
<td>Draw Widget Properties</td>
<td>229</td>
</tr>
<tr>
<td>Table Widget Properties</td>
<td>234</td>
</tr>
<tr>
<td>Chapter 8:</td>
<td></td>
</tr>
<tr>
<td>Getting Help with IDL</td>
<td>241</td>
</tr>
<tr>
<td>IDL's Online Help System</td>
<td>242</td>
</tr>
<tr>
<td>The Help Menus</td>
<td>242</td>
</tr>
<tr>
<td>The IDL Online Help Button Bar</td>
<td>247</td>
</tr>
<tr>
<td>The Navigation Dialog</td>
<td>248</td>
</tr>
<tr>
<td>Extending IDL's Online Help System</td>
<td>249</td>
</tr>
<tr>
<td>The HELP Procedure</td>
<td>252</td>
</tr>
</tbody>
</table>
Chapter 9: The Structure of the IDL Language .............. 255
  Data Types ................................................................................................................. 256
  Numeric Constants ................................................................................................... 257
  String Constants ....................................................................................................... 258
  Type Conversion Functions ..................................................................................... 259
  Arrays ........................................................................................................................ 260
  Structures .................................................................................................................. 262
  Variables .................................................................................................................... 264
  System Variables ....................................................................................................... 266

Chapter 10: Expressions and Operators ......................... 267
  Operator Precedence ................................................................................................ 268
  IDL Operators ........................................................................................................... 270
  Type and Structure of Expressions .......................................................................... 278

Chapter 11: Plotting ...................................................... 283
  Plotting Keyword Parameters .................................................................................. 284
  Graphics Coordinate Systems .................................................................................. 285
  X Versus Y Plots—PLOT and O PLOT ................................................................. 287
  Annotation – The XYOUTS Procedure ................................................................. 293
  Plotting Symbols ....................................................................................................... 295
  Polygon Filling .......................................................................................................... 298
  Tick Marks ................................................................................................................ 302
  Logarithmic Scaling .................................................................................................. 305
  Multiple Plots on a Page ........................................................................................... 306
  Specifying the Location of the Plot .......................................................................... 308
  Plotting Missing Data ............................................................................................... 310
  Using the AXIS Procedure ....................................................................................... 311
  Using the CURSOR Procedure ................................................................................ 313

Chapter 12: Plotting Multi-Dimensional Arrays .............. 317
  Contour Plots ............................................................................................................ 318
  Overlaying Images and Contour Plots ...................................................................... 324
  Additional Contour Options ..................................................................................... 328
  The SURFACE Procedure ....................................................................................... 331
## Three-Dimensional Graphics
- Three-Dimensional Graphics
- Three-Dimensional Transformations
- Shaded Surfaces
- Volume Visualization
- References

## Chapter 13: Map Projections
- The MAP_SET Procedure
- The MAP_GRID Procedure
- The MAP_CONTINENTS Procedure
- Graphics Techniques for Mapping
- Map Projections Described
- Azimuthal Projections
- Cylindrical Projections
- Pseudocylindrical Projections
- Putting Data on Maps
- High-Resolution Continent Outlines
- References

## Chapter 14: Image Display Routines
- Images
- Imaging Routines
- Image Display
- Reading from the Display Device
- Color Tables
- True-Color Displays
- Controlling the Device Cursor
- References

## Chapter 15: Signal Processing
- Digital Signals
- Signal Analysis Transforms
- The Fourier Transform
- Interpreting FFT Results
- Displaying FFT Results
- Using Windows
- Aliasing

---

### Using IDL

<table>
<thead>
<tr>
<th>Contents тогохол</th>
<th>vii</th>
</tr>
</thead>
<tbody>
<tr>
<td>Three-Dimensional Graphics</td>
<td>334</td>
</tr>
<tr>
<td>Three-Dimensional Transformations</td>
<td>343</td>
</tr>
<tr>
<td>Shaded Surfaces</td>
<td>349</td>
</tr>
<tr>
<td>Volume Visualization</td>
<td>351</td>
</tr>
<tr>
<td>References</td>
<td>355</td>
</tr>
</tbody>
</table>

### Chapter 13: Map Projections 357
- The MAP_SET Procedure | 358 |
- The MAP_GRID Procedure | 360 |
- The MAP_CONTINENTS Procedure | 360 |
- Graphics Techniques for Mapping | 361 |
- Map Projections Described | 362 |
- Azimuthal Projections | 362 |
- Cylindrical Projections | 368 |
- Pseudocylindrical Projections | 373 |
- Putting Data on Maps | 375 |
- High-Resolution Continent Outlines | 377 |
- References | 378 |

### Chapter 14: Image Display Routines 379
- Images | 380 |
- Imaging Routines | 380 |
- Image Display | 381 |
- Reading from the Display Device | 385 |
- Color Tables | 386 |
- True-Color Displays | 392 |
- Controlling the Device Cursor | 395 |
- References | 396 |

### Chapter 15: Signal Processing 397
- Digital Signals | 398 |
- Signal Analysis Transforms | 400 |
- The Fourier Transform | 400 |
- Interpreting FFT Results | 401 |
- Displaying FFT Results | 402 |
- Using Windows | 407 |
- Aliasing | 409 |
## Contents

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>FFT Usage Details</td>
<td>410</td>
</tr>
<tr>
<td>The Hilbert Transform</td>
<td>411</td>
</tr>
<tr>
<td>The Wavelet Transform</td>
<td>413</td>
</tr>
<tr>
<td>Convolution</td>
<td>413</td>
</tr>
<tr>
<td>Correlation and Covariance</td>
<td>413</td>
</tr>
<tr>
<td>Digital Filtering</td>
<td>414</td>
</tr>
<tr>
<td>Finite Impulse Response (FIR) Filters</td>
<td>414</td>
</tr>
<tr>
<td>Infinite Impulse Response Filters</td>
<td>419</td>
</tr>
<tr>
<td>Routines for Signal Processing</td>
<td>423</td>
</tr>
<tr>
<td>References</td>
<td>423</td>
</tr>
</tbody>
</table>

### Chapter 16: Mathematics

- IDL’s Numerical Recipes Functions                                 | 425  |
- Accuracy & Floating-Point Operations                             | 426  |
- Arrays and Matrices                                                | 427  |
- Correlation Analysis                                              | 429  |
- Curve and Surface Fitting                                         | 431  |
- Eigenvalues and Eigenvectors                                      | 436  |
- Gridding and Interpolation                                         | 438  |
- Hypothesis Testing                                                | 444  |
- Integration                                                       | 446  |
- Linear Systems                                                    | 449  |
- Nonlinear Equations                                               | 453  |
- Optimization                                                     | 461  |
- Sparse Arrays                                                     | 462  |
- Time-Series Analysis                                              | 464  |
- Multivariate Analysis                                             | 466  |

### Index
Chapter 1

Overview

This chapter includes information about IDL, the IDL documentation set, and about contacting Research Systems regarding problems with IDL.

About IDL

IDL 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.
Advantages of IDL include:

- IDL is a complete, structured language that can be used both interactively and to create sophisticated functions, procedures, and applications.
- Operators and functions work on entire arrays (without using loops), simplifying interactive analysis and reducing programming time.
- Immediate compilation and execution of IDL commands provides instant feedback and "hands-on" interaction.
- Rapid 2D plotting, multi-dimensional plotting, volume visualization, image display, and animation allow you to observe the results of your computations immediately.
- Many numerical and statistical analysis routines— including Numerical Recipes routines— are provided for analysis and simulation of data.
- IDL's flexible input/output facilities allow you to read any type of custom data format. Support is also provided for common image standards (including BMP, GIF, JPEG, and XWD) and scientific data formats (CDF, HDF, and NetCDF).
- IDL widgets can be used to quickly create multi-platform graphical user interfaces to your IDL programs.
- IDL programs run the same across all supported platforms (Unix, VMS, Microsoft Windows, and Macintosh systems) with little or no modification. This application portability allows you to easily support a variety of computers.
- 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 Documentation**

IDL's Online Help system gives you access to the complete IDL documentation set in electronic, hypertext-linked format. You can enter the Online Help system by entering `?` at the IDL command prompt or by selecting "Online Help" from the Help menu of the IDL Development Environment user interface.

Research Systems provides a subset of the complete IDL documentation set in printed form along with your copy of the IDL software. We do not ship the full printed documentation set because some volumes cover specialized topics, which are of limited interest to some of our customers. Shipping only the volumes of greatest general interest in printed form allows us to provide the highest quality documentation set possible while minimizing the impact of our documentation.
on the environment. In addition to being available on-line, volumes not automatically shipped with new or upgrade orders are available for purchase; use the order sheet included with your shipment or consult your sales representative or distributor for details.

The IDL documentation set consists of the following volumes:

**Using IDL**

Using IDL explains IDL from an interactive user's point of view. It contains information about the IDL environment, the structure of IDL, and how to use IDL Direct Graphics to analyze your data.

**Building IDL Applications**

Building IDL Applications (this book) explains how to use the IDL language to write programs — from simple procedures to large, complex applications. It contains information on the structure of the IDL language, programming techniques, IDL Direct Graphics, and IDL's user-interface toolkit.

**IDL Reference Guide**

The Reference Guide is a two-volume set that contains detailed information about all of IDL's non-object oriented procedures, functions, system variables, and commands. Information on IDL's object-oriented features is contained in Building IDL Applications; information on IDL Object Graphics is contained in Object Graphics.

**Object Graphics**

Object Graphics contains a complete discussion of IDL Object Graphics. This volume also contains the complete reference to IDL's object class libraries.

**Note** Each of the above books includes a comprehensive index that covers all four volumes.

**Using Insight**

Using Insight contains information on IDL Insight, the graphical interface to IDL's analysis capabilities. Insight allows you to import, analyze, and visualize data without programming in the IDL language.

**External Development Guide**

The External Development Guide explains how to use IDL in concert with your computer's operating system or with programs written in other programming languages.
Scientific Data Formats

Scientific Data Formats contains detailed information about IDL’s routines for dealing with Common Data Format (CDF), Hierarchical Data Format (HDF), Earth Observing System extensions to HDF (HDF-EOS), and Network Common Data Format (NetCDF) files.

IDL DataMiner Guide

The DataMiner Guide contains information on using IDL to interact with databases using the Open Database Connectivity (ODBC) interface.

Note The DataMiner option must be purchased separately.

IDL HandiGuide

The HandiGuide is a handy quick reference that contains calling-sequence information on all IDL routines.

Typographical Conventions

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

- **UPPER CASE**
  IDL functions, procedures, and 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**
  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 shown in courier bold 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

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 bugs do occur. 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. This section is intended to help you report problems in a way that will help us correct the problem rapidly.

Background Information
  When a bug is reported and verified, we correct it in a later release. Sometimes, a bug 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 bug:

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

  The installation number is assigned by us when you purchase IDL. The IDL version, site number, and type of machine are printed when IDL is started. For example:

  IDL. Version 4.0.1c (sunos sparc).
  All rights reserved. Unauthorized reproduction prohibited.
  Installation number: 177.
  Licensed for use by: ACME Datawhack Corp.
  is the startup announcement from IDL version 4.0.1c under SunOS on a Sun Sparc workstation at installation number 177.
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.

Under VMS, the DCL statement:

```
write sys$output $getsyi("version")
```

will give you the operating system version.

Under Windows 95 and Windows NT version 4, select “About” from the Help menu in 
the Windows Explorer. Under Windows NT version 3.5, select “About” from the Help 
menu in the File Manager.

On the Macintosh, select “About this Macintosh” from the apple menu.

Double Check

Before reporting a problem, you should always ask yourself “Is it really a bug?” 
Sometimes, it is a simple matter of misinterpreting what is supposed to happen. Double 
check with the manual or a local expert.

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.

It is often obvious whether something is a bug or not. If IDL crashes, it is a genuine bug. 
If however, it draws a plot differently than you would expect or desire, it might be a bug, 
but it is certainly less obvious. 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. Vague terms like 
“crashes”, “blows up”, and “fails” are open to many interpretations. Does it really crash 
IDL and leave you looking at an operating system prompt? This would be our 
interpretation of “crash.” Perhaps, however, it just issues an unexpected error message 
and gives another prompt. 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 one like “IDL dumps core when I execute the following statements. 
I think it might be trying to get dynamic memory”. The second version tells us exactly 
what happened. The opinion about what was going on when the problem surfaced is also 
useful to us, but it helps to have it clearly labeled as such.

Reproducibility

Intemittent bugs 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 therefore 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 bug 1 time in 3 tries are fairly likely to be fixable. Situations where the bug happens once every few months and no one is sure what triggered it are almost hopeless.

**Simplify the Problem**

When reporting a bug, it is important to give us the shortest possible series of IDL statements that cause it. The longer and more intricate an example, the less likely it is that we can help. Sometimes a single statement triggers the bug. Often though, the problem surfaces when writing a larger system of inter-related procedures and functions. Such a situation must be simplified before we can tackle it.

Take the following steps to simplify your problem:

- Copy the procedure and function files that are involved to a scratch second copy. Never modify your only copy!
- Eliminate everything that is not involved in demonstrating the bug. 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 bug still appears.
- If a simplification causes the bug to disappear, restore the statements involved and look for other things to eliminate.
- 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 your bug report does not involve I/O, it will be much easier for us to reproduce. If you have to provide us with a copy of your data, things become more complicated.

On the other hand, if the bug involves file Input/Output, attempt to determine if the problem only happens with a certain file, or with any data. If you are running under VMS, check the file organization using the DCL `DIRECTORY/FULL` command, and include this information in your report.

The end result of such simplification should be a small number of IDL statements that demonstrate the problem.

**Bugs 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 underneath the user level shell of IDL and is not protected from small programming errors that can corrupt data, give incorrect results, or even crash IDL. In such situations, the burden of proving that a bug is within IDL and not the dynamically loaded code is entirely the programmer’s.
Although it is certainly true that a bug in this situation can be within IDL, it is very important that you exhaust all other possibilities before reporting a bug. If you decide that you need to report a bug, the comments above on simplifying things are even more important than usual. If you send us a small example that tickles the bug, we can respond quickly with a correction or advice. Otherwise, we may not even know where to begin.

**Sending Data with Your Bug Report**

If the statements required to reproduce the bug are more than a few lines or require data files, we will need you to send them to us on magnetic media or via e-mail. Call us for details.

**Contact Us**

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

**Mail**

Research Systems, Inc.
4990 Pearl East Circle
Boulder, Colorado 80301

**Telephone**

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

**Electronic Mail**

support@rsinc.com
# Chapter 2

## Running IDL

The following topics are covered in this chapter:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Starting IDL</td>
<td>20</td>
</tr>
<tr>
<td>Quitting IDL</td>
<td>21</td>
</tr>
<tr>
<td>Environment Variables Used by IDL</td>
<td>22</td>
</tr>
<tr>
<td>Input to IDL</td>
<td>24</td>
</tr>
<tr>
<td>Special Characters</td>
<td>26</td>
</tr>
<tr>
<td>Executive Commands</td>
<td>27</td>
</tr>
<tr>
<td>Printing Graphics</td>
<td>28</td>
</tr>
<tr>
<td>Preparing and Running Programs</td>
<td>28</td>
</tr>
<tr>
<td>Issuing Operating System Commands</td>
<td>31</td>
</tr>
<tr>
<td>Batch Execution</td>
<td>32</td>
</tr>
<tr>
<td>Startup File</td>
<td>33</td>
</tr>
<tr>
<td>Noninteractive IDL</td>
<td>34</td>
</tr>
<tr>
<td>SAVE and RESTORE</td>
<td>34</td>
</tr>
<tr>
<td>Journaling</td>
<td>35</td>
</tr>
</tbody>
</table>
This chapter explains IDL special characters, executive commands, the various commands you can enter in response to the IDL prompt, how to prepare and run IDL programs, how to set up IDL to work with your terminal or workstation, and other general information about IDL.

Installation and licensing instructions for IDL can be found in your installation guide.

**Starting IDL**

To run IDL under Unix or VMS in command-line mode, enter `idl` at the operating system prompt. To run the IDL Development Environment graphical user interface, enter `idlde` at the Unix prompt, or `idl/de` at the VMS prompt. To run IDL under Windows or the Macintosh OS, double-click on the IDL icon. For a description of the IDL graphical user interface, see Chapter 3, “The IDL for Motif Interface”, Chapter 4, “The IDL for Windows Interface”, or Chapter 5, “The IDL for Macintosh Interface”.

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

- **Unix:** Be sure that your PATH environment variable includes the directory that contains IDL. You can find other recommended settings for environment variables at the end of this chapter.

- **VMS:** See your system manager (or the IDL installation instructions) for the proper commands to include in your LOGIN.COM file.

- **Windows:** Be sure that the path listed in the “Properties” dialog for the IDL icon (this is found under the File menu in the Program Manager in Windows NT 3.51, or by right-clicking on the IDL shortcut icon in Windows 95 and Windows NT 4.0) accurately reflects the location of the IDL executable file `idlde.exe`.

**Startup Switches Accepted by IDL**

You can alter some IDL behaviors by supplying command line switches along with the IDL command. Different switches are available on different platforms. IDL can also be started in noninteractive mode by specifying the name of a batch file at startup time. See “Noninteractive IDL” on page 34 for details.

- **Unix:** The Unix version of IDL accepts the following command line switches:

  - `-rt=file` Start IDL with a runtime license. The file argument should be an IDL `.sav` file. If no file is specified, IDL attempts to
run a file named \texttt{runtime.sav}. If you are creating IDL runtime applications, consult the IDL Runtime Guide.

-\texttt{w} \hspace{1cm} \text{Start IDL with the graphical user interface. This is the same as entering idlde at the command prompt.}

-\texttt{autow} \hspace{1cm} \text{Start IDL with the graphical user interface if possible, otherwise start IDL in command-line mode.}

-\texttt{nw} \hspace{1cm} \text{Run IDL in command-line mode no matter what. Note that specifying \texttt{idlde -nw} at the command line will start IDL in command line mode.}

Additional command line options for the graphical user interface are discussed in “The IDL for Motif Interface” on page 39.

\textbf{VMS:} The VMS version of IDL accepts the following command line qualifiers:

\texttt{/RUNTIME=\textit{file}} \hspace{1cm} \text{Start IDL with a runtime license. The file argument should be an IDL .sav file. If no file is specified, IDL attempts to run a file named \texttt{runtime.sav}. If you are developing IDL runtime applications, consult the IDL Runtime Guide.}

\texttt{/DE} \hspace{1cm} \text{Start IDL with the graphical user interface.}

\texttt{/[NO]WINDOW} \hspace{1cm} \text{Start IDL with the graphical user interface (same as /DE). If the NO prefix is included, IDL starts in command-line mode.}

\texttt{/[NO]AUTOWINDOW} \hspace{1cm} \text{Start IDL with the graphical user interface if possible, otherwise start IDL in command-line mode. If the NO prefix is included, IDL starts in command-line mode.}

\texttt{/ARRAY_MEMORY} \hspace{1cm} \text{Adjust the amount of memory allocated for IDL arrays. See “IDL\_ARRAY\_MEMORY\_SIZE” on page 23 for a more detailed description.}

Additional command line options for the graphical user interface are discussed in “The IDL for Motif Interface” on page 39.

\textbf{Windows:} The Windows version of IDL does not accept command-line switches.

\textbf{Macintosh:} The Macintosh version of IDL does not accept command-line switches.

\section*{Quitting IDL}

To quit IDL, enter the EXIT command at the IDL prompt. If you are running a version of IDL that has a graphical interface, you can also exit by selecting the “Exit” option from the File menu.
Environment Variables Used by IDL

When IDL starts, it checks the values of the following environment variables (or logical names, under VMS) and sets corresponding IDL system variables accordingly. If the environment variables do not exist, IDL uses preference settings (from the IDL Development Environment) or default values. These aspects of IDL's behavior can also be controlled without setting any environment variables.

**Note**  
On Windows and Macintosh platforms, IDL does not check environment variables. All startup values are set via the Preferences dialog.

**IDL_DEVICE**
Set this environment variable equal to the name of the default IDL graphics device. Setting this value is the same as setting the value of the IDL system variable !D.NAME. Note that the concept of a graphics device applies only to IDL Direct Graphics; IDL Object Graphics do not use the current graphics device.

**IDL_DIR**
Set this environment variable equal to the path to the main IDL directory. Setting this value is the same as setting the value of the IDL system variable !DIR.

**IDL_DLM_PATH**
Set this environment variable equal to the path to the directory or directories containing IDL dynamically loadable modules. The corresponding IDL system variable is !DLM_PATH, the value of which cannot be changed. See “!DLM_PATH” on page 42 of the IDL Reference Guide.

**IDL_HELP_PATH**
Set this environment variable equal to the path to the directory or directories containing IDL help files. Setting this value is the same as setting the value of the IDL system variable !HELP_PATH.

**IDL_PATH**
Set this environment variable equal to the path to the directory or directories containing IDL library (.pro and .sav) files. Setting this value is the same as setting the value of the IDL system variable !PATH.

**IDL_STARTUP**
Set this environment variable equal to the path to an IDL batch file that contains a series of IDL statements which are executed each time IDL is run. See “Startup File” on page 33 for further details.

♦ **Unix only:** The following environment variables are used by IDL for Unix.
DISPLAY
IDL uses the DISPLAY environment variable to choose which X display will be used to display graphics.

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

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

♦ VMS only: The following logical name is used only by IDL for VMS.

IDL_ARRAY_MEMORY_SIZE
You can control both the initial size of the memory block allocated to hold IDL arrays and the amount of memory allocated when the array memory block must be extended dynamically. You can control the memory allocation in two ways:

1. If a logical named IDL_ARRAY_MEMORY_SIZE exists when IDL starts, IDL uses its value to determine the initial and extend sizes. If the logical contains a single number, it is taken as the extend size. Two numbers separated by whitespace are taken as the extend and initial sizes, in that order. For example, to set the extend size to 1024 pages, you could put the following line into your LOGIN.COM file:

   $ DEFINE IDL_ARRAY_MEMORY_SIZE 1024

   To also make the initial size be 2048 pages:

   $ DEFINE IDL_ARRAY_MEMORY_SIZE "1024 2048"

2. Use the ARRAY_MEMORY qualifier to specify these same values at startup time. As above, to set the extend size to 1024 pages:

   $ IDL/ARRAY_MEMORY=(EXTEND=1024)

   To set the initial size to 2048 as well:

   $ IDL/ARRAY_MEMORY=(INITIAL=2048, EXTEND=1024)

The choice of qualifier or logical depends on the application. Values specified via the ARRAY_MEMORY qualifier take precedence over those specified by the logical name. Neither are required; IDL will provide defaults. The ability to set these values is provided for those with a deep understanding of VMS memory management and special requirements that the defaults don’t satisfy.
Input to IDL

Commands that are entered at the IDL prompt are usually interpreted as IDL statements to be executed. Other interpretations include executive commands that control execution and compilation of programs, shell commands, etc. Input to the IDL prompt is interpreted according to the first character of the line, as shown in Table 2-1.

<table>
<thead>
<tr>
<th>First Character</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>.</td>
<td>Executive command</td>
</tr>
<tr>
<td>?</td>
<td>Help inquiry</td>
</tr>
<tr>
<td>$</td>
<td>Command to be sent to operating system (UNIX, VMS, Windows)</td>
</tr>
<tr>
<td>@</td>
<td>Batch file initiation.</td>
</tr>
<tr>
<td>Up arrow key</td>
<td>Recall/edit previous command</td>
</tr>
<tr>
<td>Ctrl+D</td>
<td>Under UNIX, exits IDL, closes all files, and returns to operating system.</td>
</tr>
<tr>
<td>Ctrl+Z</td>
<td>Under UNIX, suspends IDL. Under VMS and Windows, exits IDL, closes all files, and returns to operating system.</td>
</tr>
<tr>
<td>All others</td>
<td>IDL statement</td>
</tr>
</tbody>
</table>

Table 2-1: Interpretation of the First Character in an IDL Command

Command Recall and Line Editing

IDL saves the last 20 command lines entered. These command lines can be recalled, edited, and re-entered. The “up arrow” key on the keypad recalls the previous command you entered to IDL. Pressing it again recalls the previous line, etc. When a command is recalled, it is displayed after the IDL prompt and can be edited and/or entered.

The line-editing abilities and the keys that activate them differ somewhat between the different operating systems. Table 2-2 lists the edit functions and the corresponding keys.

The command recall feature is enabled by setting the system variable !EDIT_INPUT to a non-zero value (the default is 1) and is disabled by setting it to 0. See “!EDIT_INPUT” on page 42 of the IDL Reference Guide for details.

Changing the Number of Lines Saved

You can change the number of command lines saved in the recall buffer by setting !EDIT_INPUT equal to a number other than one. (Under Windows, you can set this value in the Preferences dialog as well.) In order for the change to take effect,
Table 2-2: Command Recall and Line Editing Keys

IDL must be able to process the assignment statement before providing a command prompt. This means that you must put the assignment statement in the IDL startup file. (See “Startup File” on page 33 for more information on startup files.)
For example, placing the line

```
!EDIT_INPUT = 50
```

in your IDL startup file changes the number of lines saved in the command recall buffer to 50.

**Special Characters**

Table 2-3 lists characters with special interpretations and states their functions in IDL. See “Special Characters” on page 57 of the IDL Reference Guide for in-depth descriptions of the way IDL interprets these characters.

<table>
<thead>
<tr>
<th>UNIX</th>
<th>VMS</th>
<th>Windows</th>
<th>Macintosh</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>!</td>
<td>!</td>
<td>!</td>
<td>!</td>
<td>First character, system variable names</td>
</tr>
<tr>
<td>'</td>
<td>'</td>
<td>'</td>
<td>'</td>
<td>Delimit string constants or indicate part of octal or hex constant</td>
</tr>
<tr>
<td>;</td>
<td>;</td>
<td>;</td>
<td>;</td>
<td>Begin comment field</td>
</tr>
<tr>
<td>$</td>
<td>$</td>
<td>$</td>
<td>$</td>
<td>Continue current command on the next line, or issue operating system command if entered on a line by itself.</td>
</tr>
<tr>
<td>&quot;</td>
<td>&quot;</td>
<td>&quot;</td>
<td>&quot;</td>
<td>Delimit string constants or precede octal constants</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>Indicate constant is floating point or start executive command</td>
</tr>
<tr>
<td>&amp;</td>
<td>&amp;</td>
<td>&amp;</td>
<td>&amp;</td>
<td>Separate multiple statements on one line</td>
</tr>
<tr>
<td>:</td>
<td>:</td>
<td>:</td>
<td>:</td>
<td>End label identifiers</td>
</tr>
<tr>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>Array subscript range</td>
</tr>
<tr>
<td>@</td>
<td>@</td>
<td>@</td>
<td>@</td>
<td>Include file/Execute IDL batch file</td>
</tr>
<tr>
<td>Control-C</td>
<td>Control-C</td>
<td>Control-break</td>
<td>Command-.</td>
<td>Interrupt</td>
</tr>
<tr>
<td>Control-D</td>
<td>Control-Z</td>
<td>Control-Z</td>
<td>Command-Q</td>
<td>Exit</td>
</tr>
<tr>
<td>Control-\</td>
<td>Control-Y</td>
<td></td>
<td></td>
<td>Abort</td>
</tr>
</tbody>
</table>

Table 2-3: Special Characters
Executive Commands

IDL executive commands compile programs, continue stopped programs, and start previously compiled programs. All of these commands begin with a period and must be entered in response to the IDL prompt. Commands can be entered in either uppercase or lowercase and can be abbreviated. Under UNIX, filenames are case sensitive, while with VM S, Windows, and the Macintosh either case can be used. Note that comments (prefaced by the semicolon character in IDL code) are not allowed within executive commands. Executive commands are summarized in Table 2-4. See “Executive Commands” on page 31 of the IDL Reference Guide for in-depth descriptions of these commands.

<table>
<thead>
<tr>
<th>Command</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>.COMPILE</td>
<td>Compiles text from files or keyboard without executing</td>
</tr>
<tr>
<td>.CONTINUE</td>
<td>Continues execution of a stopped program</td>
</tr>
<tr>
<td>.EDIT</td>
<td>Opens files in editor windows of the IDLDE (Windows and Motif only)</td>
</tr>
<tr>
<td>.GO</td>
<td>Executes previously compiled main program from beginning</td>
</tr>
<tr>
<td>.OUT</td>
<td>Continues program execution until the current routine returns</td>
</tr>
<tr>
<td>.RETURN</td>
<td>Continues execution until encountering a RETURN statement.</td>
</tr>
<tr>
<td>.RNEW</td>
<td>Erases main program variables and then .RUN</td>
</tr>
<tr>
<td>.RUN</td>
<td>Compiles and possibly executes text from files or keyboard</td>
</tr>
<tr>
<td>.SKIP</td>
<td>Skips over the next statement and then single steps</td>
</tr>
<tr>
<td>.STEP</td>
<td>Executes a single statement (abbreviated as .S)</td>
</tr>
<tr>
<td>.STEPOVER</td>
<td>Executes a single statement if the statement does not call a routine (abbreviated as .SO)</td>
</tr>
<tr>
<td>.TRACE</td>
<td>Similar to .CONTINUE, but displays each line of code before execution</td>
</tr>
</tbody>
</table>

Table 2-4: Executive Commands
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 and Macintosh platforms, IDL uses the operating system's built-in printing facilities; on Unix and VMS 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.

Note: IDL does not support tiling or printing multi-page documents.

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 image you wish to print, then use the CLOSE_DOCUMENT keyword to DEVICE to actually initiate the print job and print something from your printer. See “The Printer Device” on page 156 of the IDL Reference Guide for details.

Printing IDL Object Graphics

To print IDL Object Graphics, you must create a printer object to use as a destination for your Draw operations. See Objects and Object Graphics for information about printer objects and their use.

Preparing and Running Programs

To enter a short program or procedure from the keyboard, simply type .RUN or .RNEW. When the final END statement is encountered, execution of the main program will begin if there was an END statement and if no errors were found. If you entered only functions or procedures or if the main program you entered had an error, IDL will display the IDL prompt to show that a program is not running.

Usually, any text editor can be used to prepare programs or procedures of more than a few lines. The GUI front-ends for IDL include built-in text editors, but these need not be used if you prefer to use your own text editor or word processor. Files containing IDL programs, procedures, and functions are assumed to have the extension name .pro. Once the program has been entered into a file from an editor, run IDL and compile one or more program files using .RUN or .RNEW.

Format of Program Files

There are essentially four types of code units in files containing IDL statements:
Procedure
A self-contained code unit with a unique name that is called by other code units to perform a desired function. The calling code unit and the procedure communicate via passed arguments.

Function
A self-contained code unit similar to a procedure. The only difference is that a function returns a value and can therefore be used in expressions.

Main Program
A series of statements that are not preceded by a procedure or function heading. They do, however, require an END statement. Since there is no heading, it cannot be called from other routines and cannot be passed in arguments. When IDL encounters a main program as the result of a .RUN executive command, it compiles it into the special program named $MAIN$ and immediately executes it. Afterwards, it can be executed again by using the .GO executive command.

Include File
A file to be included in other files. The statements contained in an include file are textually inserted into the including file. See “Batch Execution” on page 32. A file can contain any combination of functions, procedures, and/or include files. For example, a file might contain three procedures and two functions and also might be included in another file.

See “Defining Procedures and Functions” on page 109 of Building IDL Applications for more information on creating programs in IDL.

Executing Program Files

Automatic Execution
When a file is specified by typing only the filename at the IDL prompt, IDL searches the current directory for filename.pro (where filename is the file specified) and then for filename.sav. If no file is found in the current directory, IDL searches in the same way in each directory specified by !PATH. If a file is found, IDL automatically compiles the contents and executes any functions or procedures that have the same name as the file specified (excluding the suffix).

Explicit Execution
When a file is specified with the .RUN, .RNEW, .COMPILE, or @ commands, IDL searches the current directory for filename.pro (where filename is the file specified) and then for filename. If no file is found in the current directory,
IDL searches in the same way in each directory specified by !PATH. If a file is found, IDL compiles or runs the file as specified by the executive command used.

**Caution**  If the current directory contains a subdirectory with the same name as filename, IDL will consider the file to have been found and stop searching. To avoid this problem, specify the extension (.pro or .sav, usually) when entering the run, compile, or batch file command.

The details of how !PATH is initialized and used differ between the various operating systems, although the overall concept is the same. See “!PATH” on page 43 of the IDL Reference Guide for more information.

**Interrupting Program Execution**

Programs that are running can be manually stopped by typing Control-C (Unix and VMS), Control-Break (Windows) or Command-. (Macintosh). This action is called a keyboard interrupt. A message indicating the statement number and program unit being executed is issued on the terminal acknowledging the interrupt. The values of variables can be examined, statements can be entered from the keyboard, and variables can be changed. The program can be resumed by typing the executive command .CONTINUE to resume or .S to execute the next statement and stop.

**Variable Context After Interruption**

The variable context after a keyboard interrupt is that of the program unit in which the interrupt occurred. By typing the statement RETURN, the program context will revert to the next higher calling level. The RETALL command returns control to the main program level. If any doubt arises as to which program unit in which the interrupt occurred, the HELP procedure can be used to determine the program context. IDL checks after each statement to see if an interrupt has been typed. Execution does not stop until the statement that was active finishes; thus, a long time can elapse from the time the interrupt is typed to the time the program interrupts.

**Aborting IDL**

If you find it necessary to abort IDL rather than exiting cleanly using the EXIT command, do one of the following:

- **Unix:** As with any UNIX process, IDL can be aborted by typing Control+\. This is a very abrupt exit— all variables are lost, and the state of open files will be uncertain. Thus, although it can be used to exit of IDL in an emergency, its use should be avoided.
Chapter 2: Running IDL

Note After aborting IDL by using Control+`, you may find that your terminal is left in the wrong state. You can restore your terminal to the correct state by issuing one of the following UNIX commands:

```
% reset
or
% stty echo -cbreak
```

♦ VMS: As with any VMS program, IDL can be aborted by typing Control+Y. Aborting IDL with Control+Y should only be used as an emergency measure since all the variables are lost and some output may disappear. It is possible to resume IDL by typing the DCL command:

```
$ CONTINUE
```

However, if any DCL command that causes VMS to run a new program is issued prior to the CONTINUE command, the IDL session will be irreversibly lost.

♦ Windows and Macintosh: There is no abort character for either IDL for Windows or IDL for Macintosh.

Issuing Operating System Commands

Unix and VMS operating system commands can be sent to a subprocess for execution by entering the command preceded by the character $ in response to the IDL prompt. Under Windows, the $ can be used to enter a DOS or Command Shell command at the IDL prompt.

The SPAWN procedure has the same effect and is more flexible because it can be used within an IDL program while $ can only be entered interactively. In addition, the standard output of the command can be saved in an IDL string array by SPAWN. Hence, $ can be thought of as an interactive-only abbreviation for SPAWN. Unlike $, SPAWN can also be used on the Macintosh.

IDL creates and runs a process to execute the command and waits for the command to finish before issuing another IDL prompt. All of the variables, procedures, open files, etc., are saved while the command is executing. (Under Unix, essentially the same result is obtained using Control-Z to suspend IDL.) Output from the command issued is handled different ways under different operating systems:

♦_UNIX_: Output from the command is directed to the window in which IDL is running. Other windows may be created as well.
Batch Execution

IDL can be run in the non-interactive mode (the “batch” mode) by entering the character @ followed by the name of a file containing IDL executive commands and statements. All executive commands and IDL statements that normally come from the keyboard are read from the specified file.

Batch execution can be terminated before the end of the file, with control returning to the interactive mode without exiting IDL, by calling the STOP procedure from the batch file. Calling the EXIT procedure from the batch procedure has the usual effect of terminating IDL.

To enter batch mode from the interactive mode, enter:

```
@filename
```

at the IDL prompt. (Note that the @ symbol must be the first character on the line in order for it to be interpreted properly.) IDL reads commands from the specified file until the end of the file is reached. Batch files can be nested by simply prefacing the name of the new batch file with the @ character. As stated above, the current directory and then all directories in the !PATH system variable are searched (if the file was not found in the current directory). The filename can also include full path specifications (e.g., when the batch file resides in a directory that isn’t included in !PATH).

Interpretation of Batch Statements

Each line of the batch file is interpreted exactly as if it was entered from the keyboard. In the batch mode, IDL compiles and executes each statement before reading the next statement. This differs from the interpretation of programs compiled using .RNEW or .RUN, in which all statements in a program are compiled as a single unit and then executed.

Labels are illegal in the batch mode because each statement is compiled and executed independently.

Multiline statements must be continued on the next line using the $ continuation character, because IDL terminates every interactive mode statement not ending...
with $ by an END statement. A common mistake is to include a multiple-line block statement in a batch file as shown below.

```
FOR I = 1, 10 DO BEGIN
   A = X[I]
   ...
   ...
ENDFOR
```

This will not work in batch mode.

In the batch mode, IDL compiles and executes each line separately, causing syntax errors in the above example because no matching ENDFOR is found on the line containing the BEGIN statement when the line is compiled. The above example could be made to work by writing the block of statements as a single line using the $ (continuation) and & (multiple commands on a single line) characters.

### Batch Examples

An example of an IDL executive command line that initiates batch execution:

```
@myfile
```

This command causes the file myfile.pro to be used for statement and command input. If this file is not in the current directory, the search path, !PATH is also searched.

An example of the contents of a batch file follows:

```
.RUN proga  Run program A.
.RUN progb  Run program B.
PRINT, AVALUE, BVALUE  Print results.
CLOSE, 3   Close unit 3.
<eof>
```

The batch file should not contain complete program units. Complete program units should be compiled and run by using the .RUN and .RNEW commands in the batch files, as illustrated above.

### Startup File

The startup file is an IDL batch file that contains a series of IDL statements which are executed each time IDL is run. Common uses are to compile frequently used procedures or functions, customize default settings, load data, and perform other
useful operations. It contains IDL statements that are individually compiled and executed in the same manner as batch file execution.

- **Unix:** To make IDL execute a startup file under UNIX, set the environment variable IDL_STARTUP to the name of the file to be executed. If IDL_STARTUP is not defined, a startup file is not executed.

- **VMS:** To make IDL execute a startup file under VMS, assign the VM S logical name IDL_STARTUP to the name of the file to be executed. If IDL_STARTUP is not defined, a startup file is not executed.

- **Windows:** To make IDL execute a startup file under Windows, specify the name of the startup file in the “Startup” dialog, found under “Preferences” in the IDL for Windows File menu.

- **Macintosh:** To make IDL execute a startup file on the Macintosh, specify the name of the startup file in the “IDL Startup Settings” dialog, found under “Preferences” in the IDL for Macintosh Edit menu.

The procedure search path, !PATH, is used when searching for the file if it is not in the current directory. Startup command files are executed before the batch file present in the initial command line, if any.

**Noninteractive IDL**

Under Unix and VMS, IDL can be run entirely in the non-interactive mode by including the name of a file containing batch mode commands in the shell command used to invoke IDL. When the end of the file is reached, control reverts to the interactive mode and input is accepted from the keyboard. Call the EXIT procedure from the file to cause IDL to return to the operating system if you do not want to use IDL in the interactive mode. The operating system command:

```
idl startup
```

runs IDL. IDL then executes in batch mode the text in the file startup.pro and reverts to the interactive mode if a call to EXIT is not present in the file.

**SAVE and RESTORE**

The SAVE and RESTORE procedures combine to provide the ability to save the state of variables, system variables, and procedures and functions to restore them at a later time. This ability to “checkpoint” a session and then recover it later can be very convenient. SAVE/RESTORE files can be used for many purposes.
Save files can be used to recover variables that are used from session to session. A startup file can be set up to execute the RESTORE command every time IDL is started. (See “Startup File” on page 33 for details on startup files.)

Save files can be used to distribute IDL code in binary format. If you have a program or programs you wish to distribute, but do not want other to be able to view or edit the source code, use a save file.

The state of an IDL session can be saved quickly and easily, and can be restored to the same point. This feature allows you to stop work, and later resume at a convenient time.

Data can be conveniently stored in SAVE/RESTORE files, relieving the user of the need to remember the dimensions of arrays and other details. It is very convenient to store images this way. For instance, if the three variables $R$, $G$, and $B$ hold the color table vectors, and the variable $I$ holds the image variable, the IDL statement,

\[
\text{SAVE, FILENAME } = \text{ 'image.dat'}, R, G, B, I
\]

will save everything required to display the image properly in a file named image.dat. At a later date, the simple command,

\[
\text{RESTORE, 'image.dat'}
\]

will recover the four variables from the file.

Long iterative jobs can save their partial results in SAVE/RESTORE format to guard against losing data if some unexpected event such as a machine crash should occur.

**Journaling**

Journaling provides a record of an interactive session by saving in a file all text entered from the terminal in response to a prompt. All text entered to the IDL prompt is entered directly into the file, and any text entered from the terminal in response to any other input request (such as with the READ procedure) is entered as a comment. The result is a file that contains a complete description of the IDL session.

\[
\text{JOURNAL[, Argument]}
\]

where Argument is either a filename (if journaling is not currently in progress) or an expression to be written to the file (if journaling is active).

The first call to JOURNAL starts the logging process. If no argument is supplied, a journal file named idlsave.pro is started.
**Caution** Under all operating systems except VMS, creating a new journal file will cause any existing file with the same name to be lost. Supply a filename argument to JOURNAL to avoid destroying desired files.

When journaling is not in progress, the value of the system variable !JOURNAL is zero. When the journal file is opened, the value of this system variable is set to the number of the logical file unit on which the file is opened. This allows IDL routines to check if journaling is active. You can send any arbitrary data to this file using the normal IDL output routines. In addition, calling JOURNAL with an argument while journaling is in progress results in the argument being written to the journal file as if the PRINT procedure had been used. In other words, the statement,

```
JOURNAL,
```

is equivalent to

```
PRINTF, !JOURNAL, Argument
```

with one significant difference—the JOURNAL statement is not logged to the file, only its output; while the PRINTF statement will be logged to the file in addition to its output.

Journaling ends when the JOURNAL procedure is called again without an argument or when IDL is exited. The resulting file serves as a record of the interactive session that went on while journaling was active. It can be used later as an IDL batch input file to repeat the session, and it can be edited with any text editor if changes are necessary.

As an example, consider the following IDL statements:

```
JOURNAL, 'demo.pro'
PRINT, 'Enter a number:'
READ, Z
JOURNAL, '; This was inserted with JOURNAL.'
PRINTF, !JOURNAL, '; This was inserted with PRINTF.'
```

If these statements are executed by a user named Doug on a Sun workstation named quixote, the resulting journal file demo.pro will look like the following:
; IDL Version 5.0 (sunos sparc)
; Journal File for doug@quixote
; Working directory: /home/doug/IDL
; Date: Mon Oct 7 14:38:24 1996

PRINT, 'Enter a number:'
;Enter a number:
READ, Z
; 87
; This was inserted with JOURNAL.
; This was inserted with PRINTF.
PRINTF, !JOURNAL, '; This was inserted with PRINTF.'

Note that the input data to the READ statement is shown as a comment. In addition, the statement to insert the text using JOURNAL does not appear, while the statement using PRINTF does appear.
# Chapter 3

## The IDL for Motif Interface

The following topics are covered in this chapter:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>The Main IDL Window</td>
<td>40</td>
</tr>
<tr>
<td>IDLDE Windows</td>
<td>42</td>
</tr>
<tr>
<td>The Menu Items</td>
<td>43</td>
</tr>
<tr>
<td>Keyboard Shortcuts</td>
<td>55</td>
</tr>
<tr>
<td>Using Preferences to Customize IDLDE</td>
<td>57</td>
</tr>
<tr>
<td>Using Resources to Customize IDL</td>
<td>66</td>
</tr>
<tr>
<td>Command Line Options</td>
<td>68</td>
</tr>
<tr>
<td>Modifying the Control Panel</td>
<td>72</td>
</tr>
<tr>
<td>Action Routines</td>
<td>74</td>
</tr>
<tr>
<td>Modifying the Macros Menu</td>
<td>79</td>
</tr>
<tr>
<td>Using External Editors</td>
<td>80</td>
</tr>
<tr>
<td>Using the IDL Editor</td>
<td>81</td>
</tr>
</tbody>
</table>
IDL for Unix and VMS platforms can be used with one of two different interfaces. Starting IDL with the command `idl` begins a traditional IDL session using a simple tty (text) command line interface. If you are running the X Window system, however, IDL can also be started with the command `idlide` (or `idl/de` under VMS), which invokes a convenient multiple-document interface called the IDL Development Environment (IDLDE). This chapter describes the IDLDE.

See “Starting IDL” on page 20 and “Environment Variables Used by IDL” on page 22 for details on running IDL with its command-line interface.

**The Main IDL Window**

Figure 3-1 shows the default appearance of the IDLDE window. The seven sections of this window are described below.

![Figure 3-1: The Main IDL Window.](image)
Menu Bar

The menu bar, located at the top of the main IDLDE window, is used to control various IDLDE features.

Control Panel

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; see “Toolbar” on page 54 for more information. The buttons displayed as well as the commands they issue are completely configurable (see “Modifying the Control Panel” on page 72). When you position the mouse pointer over a Control Panel button, the Status Bar displays a brief description.

Multiple Document Panel

The top section of the main IDL window is where IDL Editor windows are displayed. The Multiple Document Panel can be sized by moving the “thumb” located above the Output Log slider. The Multiple Document Panel can be hidden by selecting “Hide Panel” item from the Windows menu.

Output Log

Output from IDL is displayed in the Output Log window, which appears by default when IDLDE is first started. The Output Log area can be sized by moving the thumb located above the Output Log slider and can be made invisible (along with the Command Input Line) by selecting “Hide Command” from the Windows menu. If you click the right mouse button while positioned over the Output Log, a popup menu appears allowing you to move to a specified error or clear the contents of the Output Log.

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. For more information about the Variable Watch Window, see “The Variable Watch Window” on page 145.

Command Input Line

The Command Input Line is a single IDL prompt where you can enter IDL commands. The text output by IDL commands is displayed in the Output Log window. The Command Input Line can be made invisible (along with the Output Log) by selecting “Hide Command” from the Windows menu.
If you click the right mouse button while positioned over the Command Input Line, a popup menu appears displaying the command history, with a maximum buffer of 20 entries. You can specify the number of lines in the recall buffer with the General Preferences tab from the File menu. If you enter \texttt{HELP \slash \textsc{RECALL\_COMMANDS}} at the Command Input Line, you will see the same results, except that the number of saved lines are changed by specifying the environment variable \texttt{EDIT\_INPUT} in the IDL startup file.

You can also open and compile files from the Command Input Line. See “Open [Ctrl+O]” on page 43 and “Compile filename.pro [Ctrl+F5]” on page 48 for more information.

\textbf{Status Bar}

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

\textbf{IDLDE Windows}

Two types of windows can be created and manipulated with IDLDE: IDL Editor and IDL Graphics windows.

\textbf{IDL 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 “New” or “Open” from the File menu.

The “Multiple Windows/Single Window” toggle option under the Windows menu allows you to either display one file at a time inside the IDLDE main window, or to work with multiple Editor windows outside the main window. See “Using the IDL Editor” on page 81 for more information on the IDL Editor.

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.

If a program error or breakpoint is encountered, IDLDE displays the relevant file, opening it if necessary. The line at which the breakpoint or error occurred is marked. See Chapter 6, “Programming Tools” for more on IDL’s debugging commands.
IDL Graphics Windows

IDL Graphics windows appear when you use IDL to plot or display an image.

When an IDL Graphics window is minimized (iconized), the icon for that window consists of the X motif icon titled with the name of the IDL window iconized. This icon appears on the desktop, not in the Multiple Document Panel.

Caution If a window is iconized, it will not be refreshed upon return if system or IDL backing store is not enabled.

The Menu Items

Seven menus (File, Edit, Search, Run, Macros, Window, and Help) allow you to control the operation of the IDLDE. These menus are described below. You can cause each menu to float on the desktop by clicking on the dotted line at the top of each menu listing. Each menu becomes a tear-off.

Keyboard accelerators are shown in brackets.

File Menu

New [CTRL+N]
Select this option to create a new, empty IDL Editor window. Each window is titled “Untitled” until saved. This option is also accessible by clicking the “New Document” button from the Toolbar (first button).

Open [CTRL+O]
Select this option to open a text file for editing. The Open dialog appears. Use the filter to search a specific directory. To open a file, either double-click on the file you want to open or type the file name in the Selection field and click “OK”. If the Multiple Windows option is in effect, a new IDL Editor window is created outside the main window to contain each text file. If the Single Window option is in effect, the new file is displayed and all others are listed in the Windows menu.

You can also open files from the Command Input Line. Enter the following at the IDL prompt:

```
.EDIT file1 [file2 ... file_n]
```

where file is the name of the file you want to open. If the path is not specified in the Paths Preference from the File menu, you must enter the full path for file. See “.EDIT” on page 33 of the IDL Reference Guide 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.

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 saving and re-compiling simply re-runs the previously-compiled version. Select “Compile” from the Run menu to return to the main program level and re-compile the routine, without incorporating recent changes. Select “Compile from Memory” from the Run menu to compile the last-saved version of the file without saving or implementing recent changes.

Save As... [CTRL+W]
Select this option to save the contents of an IDL Editor window to a specified filename. The Save As File Selection dialog box appears.

Revert to Saved
Select this option to reload the last saved version of the document.

Caution Unsaved changes are lost without warning.

Print... [CTRL+P]
Select this option to print the contents of the currently-selected window to the currently-active printer. The Print dialog appears.

Print Setup
Select this option to change the printer and printing options. The Printer Setup dialog appears. For further information on Printer Setup, select “Printer Help” from the Help menu of the IDL Online Help.

Recent Files
Select this option to view or open recently opened files. This menu item lists the last ten opened files. To open a file on this list, select it.

To change the maximum number of files displayed from ten to another number, modify the idlde.numRecentFiles resource in your resource file called .idlde, which is located in your home directory.
Preferences
Select this option to display a dialog box containing five tab selections with which you can customize your interaction with the IDLDE environment. The five categories are: General, Layout, Graphics, Edit, Startup, Fonts, and Paths.

• General
This tab allows you to set the look of the IDLDE interface.

• Layout
This tab allows you to specify the location and size of the main window on the screen. You can also designate the appearance of the IDLDE’s components.

• Graphics
This tab allows you to specify graphics window dimensions and also to select how to handle IDL’s backing store.

• Edit
This tab allows you to specify how to compile files in IDL.

• Startup
This tab allows you to specify IDL’s main directory, working directory, and the startup file. The startup file specifies the startup path of IDLDE for the next session. It is also possible to disable the use of the startup file.

• Fonts
This tab allows you to specify the fonts used in document windows.

• Paths
This tab allows you to specify the IDL Files Search Path. Your entries are appended to the system variable \PATH .

Exit [CTRL+Q]
Select this option to exit IDLDE. All IDL Editor windows are closed before exiting. If text in an Editor window has changed, you are asked if you want to save it before exiting.

Edit Menu

Undo [ALT+Z]
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 [ALT+Y]**
Select this option to redo previously undone editing actions. Multiple “redo” operations are supported; the first reverses the most recent undo, etc.

**Cut [ALT+X]**
Select this option to remove currently-selected text from an IDL Editor window to the Motif clipboard.

**Copy [ALT+C]**
Select this option to copy currently-selected text from an IDL Editor window to the Motif clipboard.

**Paste [ALT+V]**
Select this option to paste the contents of the Motif clipboard at the current insertion point. The insertion point can only be placed in an IDL Editor window.

**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 select all of the text in an IDL Editor window. The entire contents of the window are highlighted.

**Clear All**
Use this option to erase all of the contents in the current IDL Editor window.

**Clear Log [CTRL+Y]**
Use this option to erase the contents of the Output Log window.

**Search Menu**

**Find... [ALT+F]**
Select this option to find text in the currently-active IDL Editor window. The Find/Replace dialog appears. The attributes available for the Find dialog are described below:

- **Find**: Enter text to search for in this field.
- **Case**: Sensitive” or “Non-sensitive” — Specify if the search should reflect the case of the text entered in the “Find” field.
• Search: Forward or Backward: Specify the direction in which you would like to begin the search.

• Start at: Top, Current or Bottom — Specify where to begin the search. For Top and Bottom, the Search automatically moves Forward or Backward, respectively. After a word is found, the Search begins at Current.

• Whole words only: Select this checkbox so that the search applies only to the entire designated word, instead of finding the word within other words as a substring.

• Files: You can specify an open file in which to search or that all open files be searched. By default, the search will take place in the currently-selected window. You can also create a Tear-off from the pulldown menu (click on the dashed line at the top), which remains open as long as the Find/Replace dialog is open.

To replace text, use the Replace dialog [ALT+R].

Find Again [ALT+G]
Select this option to repeat the most recent text search.

Find Selection [ALT+H]
Select this option to find the next occurrence of the currently-selected text.

Enter Selection [ALT+T]
Select this option to enter selected text in the Find field of the Find/Replace dialog.

Replace... [ALT+R]
Select this option to find text in an IDL Editor window and replace it with new text. The Find/Replace dialog appears. See “Find... [Alt+F]” on page 46 for a description of most of the attributes for the Replace dialog; the differing attributes available for the Replace dialog are described below:

• Replace: To replace an occurrence of the text specified in the Find field, enter the replacement text in this field. Click “Replace” to change the found text. Click “Replace&Find” to change the found text and find the next occurrence of the text specified. You can only click “Replace” or “Replace&Find” if a word has been found, i.e. if it is highlighted in the relevant Editor window.

• Replace all: Click on this checkbox to specify that all occurrences of the word in the “Find” field be replaced by the word in the “Replace” field. Click “Replace” to change all the words in the specified file(s).

Replace & Find [ALT+P]
Select this option to repeat the most recent search-and-replace operation.
Go To Line [CTRL+G]
Use this option to jump directly to a specified line number in an IDL Editor window. The Go To Line dialog box appears. Enter the line number in the “Goto Line:” field.

Go To Definition [CTRL+T]
Use this option to display the definition of the currently-selected procedure or function, which must have been compiled during the current IDLDE session.

Run Menu
Run Menu items are enabled when an IDL program is loaded into an IDL Editor window. 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 6, “Programming Tools” for more detailed information.

Compile filename.pro [CTRL+F5]
Select this option to compile filename.pro. 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 Input Line, with the appropriate Editor window selected in the Multiple Document Panel.

You can also compile files from the Command Input Line. Enter the following at the IDL prompt:

```
.COMPILE file_1 [file_2 ... file_n]
```

where file_1 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 Paths Preference from the File menu, you must enter the full path for file.

See “.COMPILE” on page 32 of the IDL Reference Guide for more detail.

Compile from Memory filename.pro [CTRL+F6]
Select this option to compile filename.pro from the last-saved version of the file, without saving or implementing recent changes. Selecting this option is the same as entering .COMPILE -f at the Command Input Line. See “.COMPILE” on page 32 of the IDL Reference Guide 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, filename, contained in the currently-active editor window. Selecting this option is the same as entering the procedure name at the Command Input Line or using the .GO executive command at the Command Input Line. If the file is interrupted while running, selecting this option resumes execution; it is the same as entering .CONTINUE at the Command Input Line. For more information, see .CONTINUE and“.GO” on page 33 of the IDL Reference Guide.

Caution 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]
Select this option to iteratively compile all uncompiled 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 Input Line. The QUIET keyword suppresses informational messages. See “RESOLVE_ALL” on page 952 of the IDL Reference Guide for a more detailed explanation.

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” on page 141.

Break [CTRL+C]
Select this option to interrupt program execution. IDL inserts a marker to the left of the line at which program execution was interrupted.

Reset [CTRL+R]
Select this option to reset the IDL environment. Selecting this item is the same as entering the following at the Command Input Line:

```
RETAIL
WIDGET_CONTROL,/RESET
CLOSE,/ALL
HEAP_GC,/VERBOSE
```
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 “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 Input Line. See “.STEP” on page 35 of the IDL Reference Guide 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 “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 Input Line. See “.STEPOVER” on page 36 of the IDL Reference Guide for a more detailed explanation.

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 Input Line. See “.OUT” on page 33 of the IDL Reference Guide for a more detailed explanation.

Trace ...
Select this option to access the Trace dialog. You can modify the interval between successive .STEP or .STEPOVER commands, depending on whether the “Step Over” option is enabled. 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” on page 36 of the IDL Reference Guide 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” on page 229 of the IDL Reference Guide for a more detailed explanation.

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” on page 33 of the IDL Reference Guide for a more detailed explanation.
Set Breakpoint [F9]
Select this option to set a breakpoint on the current line. Selecting this item is the same as entering the following at the IDL Command Input Line:

```
BREAKPOINT, ['file',] line
```

where “file” is the file to set a breakpoint within, and `index` designates the line number at which the breakpoint is set.

See “BREAKPOINT” on page 229 of the IDL Reference Guide for a more detailed explanation.

Set Complex Breakpoint
Select this option to access the Complex Breakpoint dialog to set a complex breakpoint. Complex breakpoints may function only once, or may function only after being “hit” a specified number of times. Selecting this item is the same as setting the AFTER and ONCE keywords for the BREAKPOINT procedure at the IDL Command Input Line.

Enter the source file in which to set a breakpoint in the “File field. The default field is the one in which the cursor is positioned. Click “File...”, at the bottom of the dialog, to search through available directories. Enter the line number at which to place the breakpoint in the “Line” field. The default is the line at which the cursor is currently positioned. You can specify how many times the line must be hit in order to interrupt execution. Click “Once” to interrupt execution after encountering the line for the first time or click “Break After” and enter the number of hits after which execution should be interrupted into the given field. Click on “Condition” to specify an expression that will be evaluated each time the breakpoint is encountered. If and when the condition is true, program execution is interrupted.

See “BREAKPOINT” on page 229 of the IDL Reference Guide for a more detailed explanation.

Clear All Breakpoints
Select this option to clear all breakpoints.

List Breakpoints
Select this option to list all breakpoints currently set, in all compiled programs. Selecting this item is the same as entering `HELP, /BREAKPOINTS` at the IDL Command Input Line.

Up Stack [CTRL+Up]
Select this option to move up the call stack by one.
**Down Stack [CTRL+Down]**
Select this option to move down the current 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 Input Line. See “HELP” on page 537 of the IDL Reference Guide for a more detailed explanation.

**Macros Menu**
Macros allow you to access frequently-used IDL commands from a menu. You can add your own macros to the macros menu by editing your `.idlde` resource file. See “Modifying the Macros Menu” on page 79 for more information. The following macros are installed by default. (Unix syntax is shown; similar macros are installed under OpenVMS.)

**Edit...**
Select this option to access the Edit Macros dialog. The Edit Macros dialog is a convenient GUI with which you can modify existing macros or create new ones. The macros can be applied as either a menu item or a toolbar button. Click on a menu or toolbar macro to view its attributes. You can specify different attributes for a macro, some of which are required. You can also rearrange the order of the menu or toolbar macros with the up and down arrows located at the bottom of the Macro Attributes section.

**Print Variable**
Select this option to print the selected variable. Selecting this item is the same as entering **PRINT, x** at the IDL Command Input Line, where x is the selected variable.

**Help On Variable**
Select this option to list attributes of the selected variable. Selecting this item is the same as entering **HELP, /STRUCTURE** at the IDL Command Input Line, where x is the selected variable.

**Insight**
Select this option to start IDL's Insight application. Insight allows you to visualize and interactively manipulate your data.

**IDL Demo**
Select this option to start the IDL Demo application.
File in XEmacs
Select this option to edit the currently selected file with the XEmacs editor. Selecting this item is the same as typing `SPAWN, 'xemacs +index file &'` at the Command Input Line, where file is the full pathname of the file to be edited and index is the line number at which the cursor is positioned.

File in running XEmacs
Select this option to edit the currently selected file with a currently-running XEmacs editor. Selecting this item is the same as typing `SPAWN, 'gnuclient +index file &'` at the Command Input Line, where file is the full pathname of the file to be edited and index is the line number at which the cursor is positioned.

File in Emacs (X Window)
Select this option to edit the currently selected file with the Emacs (X Window) editor. Selecting this item is the same as typing `SPAWN, 'emacs +index file &'` at the Command Input Line, where file is the full pathname of the file to be edited and index is the line number at which the cursor is positioned.

File in Emacs (Xterm)
Select this option to edit the currently selected file with the Emacs (Xterm) editor. Selecting this item is the same as typing `SPAWN, 'xterm -e emacs -nw +index file &'` at the Command Input Line, where file is the full pathname of the file to be edited and index is the line number at which the cursor is positioned.

File in running Emacs
Select this option to edit the currently selected file with a currently-running Emacs editor. Selecting this item is the same as typing `SPAWN, 'emacsclient +index file &'` at the Command Input Line, where file is the full pathname of the file to be edited and index is the line number at which the cursor is positioned.

File in vi (Xterm)
Select this option to edit the currently selected file with the vi editor. Selecting this item is the same as typing `SPAWN, 'xterm -e vi +index file &'` at the Command Input Line, where file is the full pathname of the file to be edited and index is the line number at which the cursor is positioned.

Windows Menu
Read 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 [F11]
Select this option to shift IDL’s focus to the next numbered editor window.

Previous [ALT+F11]
Select this option to shift IDL’s focus to the previous numbered editor window.

Cascade
Select this option to arrange all open windows in a staggered, overlapping fashion.

Tile
Select this option to arrange all open windows in a non-overlapping fashion.

Close All
Select this option to close all open files. If a file has not yet been saved, you are asked if you would like to save changes.

Configure
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 “The Main IDL Window” on page 40.

• 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)

Toolbar
Select this option to access a pulldown menu with the three Windows toolbars: Hide Standard Tools (Show Standard Tools), Hide Run & Debug Tools (Show Run & Debug Tools), and Hide Macros (Show Macros).

Multiple Windows (Single Window)
Select this option to toggle between two available window arrangements. Selecting “Multiple Windows” opens windows outside the IDLDE interface. By default, all windows are staggered. Selecting “Single Window” displays the most recent window within the main window and lists the others as menu items in the Window menu.
Open Windows
The 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. If the Single Window menu item is active, the selected file will be displayed in the main window. If the Multiple Windows menu item is active, the selected item's window will be brought to the foreground.

Help Menu

Help on IDL...
Select this option to display the IDL Online Help Viewer.

Find Topic...
Select this option to access the Search dialog for IDL Online Help.

Help on IDE...
Select this option to display this chapter of Using IDL.

Help on Help
Select this option to learn about how to use Help.

About IDL
Select this option to display information on the IDL version in use.

Keyboard Shortcuts
Many of the menu options can be accessed from the keyboard as well as by selecting from the menus. Table 3-1 lists all of the available keyboard equivalents. Note that these equivalents are also shown to the right of each menu item in the menus themselves.

<table>
<thead>
<tr>
<th>Keyboard Shortcut</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alt+C</td>
<td>Copy selected text</td>
</tr>
<tr>
<td>Alt+F</td>
<td>Find</td>
</tr>
<tr>
<td>Alt+G</td>
<td>Find Again</td>
</tr>
<tr>
<td>Alt+H</td>
<td>Find Selection</td>
</tr>
<tr>
<td>Alt+P</td>
<td>Replace and Find</td>
</tr>
<tr>
<td>Alt+R</td>
<td>Replace</td>
</tr>
<tr>
<td>Alt+T</td>
<td>Enter Selection</td>
</tr>
</tbody>
</table>

Table 3-1: Keyboard Shortcuts
## Keyboard Shortcuts

<table>
<thead>
<tr>
<th>Keyboard Shortcut</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alt+V</td>
<td>Paste</td>
</tr>
<tr>
<td>Alt+X</td>
<td>Cut selected text</td>
</tr>
<tr>
<td>Alt+Y</td>
<td>Redo</td>
</tr>
<tr>
<td>Alt+Z</td>
<td>Undo</td>
</tr>
<tr>
<td>Alt+F5</td>
<td>Resolve dependencies</td>
</tr>
<tr>
<td>Alt+F11</td>
<td>Previous numbered editor window</td>
</tr>
<tr>
<td>Ctrl+C</td>
<td>Interrupt program execution / Break</td>
</tr>
<tr>
<td>Ctrl+G</td>
<td>Go To Line</td>
</tr>
<tr>
<td>Ctrl+N</td>
<td>New editor window</td>
</tr>
<tr>
<td>Ctrl+O</td>
<td>Open IDL Editor window</td>
</tr>
<tr>
<td>Ctrl+P</td>
<td>Print contents of editor window</td>
</tr>
<tr>
<td>Ctrl+Q</td>
<td>Exit IDL</td>
</tr>
<tr>
<td>Ctrl+R</td>
<td>Reset</td>
</tr>
<tr>
<td>Ctrl+S</td>
<td>Save contents of editor window</td>
</tr>
<tr>
<td>Ctrl+T</td>
<td>Go To Definition</td>
</tr>
<tr>
<td>Ctrl+W</td>
<td>Save contents of editor window to another file name</td>
</tr>
<tr>
<td>Ctrl+Y</td>
<td>Erase contents of Output Log</td>
</tr>
<tr>
<td>Ctrl+F5</td>
<td>Compile program in current window</td>
</tr>
<tr>
<td>Ctrl+F6</td>
<td>Compile program from memory</td>
</tr>
<tr>
<td>Ctrl+F7</td>
<td>Run To Return</td>
</tr>
<tr>
<td>Ctrl+F8</td>
<td>Step Out</td>
</tr>
<tr>
<td>Ctrl+↑ (Up arrow)</td>
<td>Up stack</td>
</tr>
<tr>
<td>Ctrl+↓ (Down arrow)</td>
<td>Down stack</td>
</tr>
<tr>
<td>Delete</td>
<td>Deletes selection</td>
</tr>
<tr>
<td>F5</td>
<td>Run</td>
</tr>
<tr>
<td>F6</td>
<td>Continue stopped program in current window</td>
</tr>
<tr>
<td>F7</td>
<td>Run to cursor</td>
</tr>
<tr>
<td>F8</td>
<td>Step Into</td>
</tr>
<tr>
<td>F9</td>
<td>Set/Clear Breakpoint</td>
</tr>
<tr>
<td>F10</td>
<td>Step Over</td>
</tr>
<tr>
<td>F11</td>
<td>Next numbered editor window</td>
</tr>
</tbody>
</table>

Table 3-1: Keyboard Shortcuts
Using Preferences to Customize IDLDE

IDLDE can be customized in two ways. By editing the resource files or by selecting “Preferences” from the IDL File menu. The Control Panel buttons and the Menu items are two common areas of customization. For further information about editing resource files, see “Using Resources to Customize IDL” on page 66.

The IDL Preferences dialog box contains five tab selections with which you can customize your interaction with the IDLDE environment. The seven categories are: General, Layout, Graphics, Edit, Startup, Fonts, and Paths.

Note It is important to understand the distinctions in application that occur throughout the Preferences Dialog, as described in Table 3-2

<table>
<thead>
<tr>
<th>Button</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>OK</td>
<td>Changes are applied to the current session and the Preferences dialog is dismissed.</td>
</tr>
<tr>
<td>Apply</td>
<td>Changes are applied to the current session but not saved. The Preferences dialog remains visible.</td>
</tr>
<tr>
<td>Save</td>
<td>Changes are applied to the current session and saved. If the option has an asterisk next to it, you must save and restart the IDLDE for the change to take effect.</td>
</tr>
<tr>
<td>Cancel</td>
<td>Any unapplied changes are ignored and the Preferences dialog is dismissed.</td>
</tr>
</tbody>
</table>

Table 3-2: Application Button Distinctions
General Preferences

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

- **Show Splash Screen**
  Select this option to show IDL’s splash screen on startup. IDL must be restarted for this option to take effect.

- **Save Preferences on Exit**
  Select this option to save all the settings—as specified in the seven Preference tabs—from the current IDL session to be applied to future IDL sessions.

- **Confirm Exit**

Log and Command Window
The performance of IDL can depend upon the number of saved lines. The amount of memory required for greater numbers of saved lines can affect the speed at which IDL will run. Click in the field next to each description and enter your adjusted value.

Figure 3-2: The General Preferences and Layout Preferences dialogs
• Lines to Save
   This field controls the minimum number of lines retained by the Output Log window. The default is 500 lines. IDL must be restarted for this option to take effect.

• Delete on limit
   This field controls the number of lines to delete at a time until the limit is reached again. The default is 125. IDL must be restarted for this option to take effect.

• Lines saved in the recall buffer:
   This field controls the maximum number of lines saved in the recall buffer. There are three ways to access the contents of the recall buffer, all of which are limited by this field. After locating the cursor in the Command Input Line, you can press your up arrow key to scroll through your last entries. You can also enter \texttt{HELP, /RECALL} in the Command Input Line or click on your right mouse button while positioned over the Command Input Line to display your entries up to the limit specified by the recall buffer. The default is 20.

Files
You can specify how files are opened within the IDLDE:

• Change Directory on Open
   Click on this checkbox to change the working directory to the directory of the most recently opened file.

• Open Files Read Only
   Click on this checkbox to open files so that they are not writable.

Layout Preferences
   This tab allows you to control the look of the main IDL window.

Main window
   By default the size of the window is 1/4 of the screen size (i.e., 1/2 the screen width and 1/2 the screen height). The window is positioned such that the upper-left corner of the window is at the upper-left corner of the screen. Click on “Default” to use these settings.

   To change the layout, click on “Specify”, which allows you to adjust the positioning of the window with the “Left” and “Top” fields and to adjust the size of the window with the “Width” and “Height” fields.
• **Left**
  The horizontal location of the upper-left corner of the main IDL window (in pixels) relative to the left side of the screen. The default is 75.

• **Top**
  The vertical location of the upper-left corner of the main IDL window (in pixels) relative to the top of the screen. The default is 25.

• **Width**
  The width of the main IDL window in pixels. The default is 1/2 the screen width. The value in this entry reflects the current width of the main IDL window.

• **Height**
  The height of the main IDL window in pixels. The default is 1/2 the screen height. The value in this entry reflects the current height of the main IDL window.

Click on “Remember” to apply the settings to future IDL sessions. This option—as indicated by the asterisk before “Main Window”—is unavailable until IDL has been restarted.

**Windows**
These options are also contained in the Windows menu of the IDLDE. The difference between the Windows section of the Layout Preferences and the Windows menu is that any changes to the Preferences are applies to future IDL sessions.

• **Editor Layout**
  Click on **Multiple** to display open Editor windows separately from the main IDLDE window. The Editor Layout is listed as **Multiple Window/Single Window** in the Windows menu.

• **Hide**
  Click on any of the sections of the IDLDE window to hide them from view. If the checkbox is marked, the section is hidden. By default, none of the sections are hidden. The Hide options are found in the pulldown menu accessed with the Configure option from the Windows menu.

• **Separate**
Click on the available sections to separate them from the main IDLDE window. If the checkbox is marked, the section can be found on the desktop in a separate window. If you dismiss a window, the Hide option for that section, as described above, is enabled. To view a dismissed window, un-hide it and click “OK” or “Apply”.

**Note** If the Multiple Windows option is enabled, the choice to hide or view the Editor windows is not available.

**Control Panel**
You can specify how you would like to display the Control Panel buttons:

- **Hide Tools**

  Click on any of the available toolbars: Standard, Run&Debug, and User to change their visibility. If a box is checked (it will appear darker), the toolbar with which it is associated is hidden on the main IDLDE window.

- **Number of Rows**

  Enter the number of rows to use in displaying any visible toolbars. You can select from 1 to 3 rows. If the window has been separated, as described in “Separate” above, number entered is reflected in the separated window.

- **Vertical**

  If the Control Window has been separated, you can specify if the Toolbars should be displayed horizontally or vertically. If the “Vertical” checkbox is marked, the toolbars are displayed vertically in a separate window. By default, separated toolbars are displayed horizontally. This option is available only when the Control Panel has been separated.

**Graphics Preferences**

**Windows Size**
This section of the Graphics Preferences tab allows you to specify the appearance of an IDL graphics window.

- **Default Width**

  The width of IDL graphics windows, in pixels. The default is 1/4 of the total screen width.

- **Default Height**

  The height of IDL graphics windows, in pixels. The default is 1/4 of the total screen height.
• Use 1/4 the screen size

If this box is checked, the Graphics windows are set to 1/2 the screen size in both width and height. If this box is unchecked, the Graphics windows are sized according to content.

**Backing Store**

When backing store is enabled, a copy of each Graphics window is kept in memory. The copy of the window is used to refresh the window when it has been covered and uncovered. IDL's performance increases for no backing store, since the amount of memory required to save files can affect the speed at which IDL will run.

See “Backing Store” on page 144 of the IDL Reference Guide for more information.

• None, RETAIN = 0

Click None to disable backing store. This option does not keep a copy of the window, allowing the highest performance in terms of speed.

• System, RETAIN = 1

Click this option to request backing store from the server.

•Pixmap, RETAIN = 2
Click this option (the default) to have IDL maintain backing store. Most users should keep this value set to 2.

**Graphics Attributes**

- **Size of TrueType Font Cache (in glyphs)**
  Enter the number of TrueType characters to save triangulation information for. 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.

- **Object Graphics Renderer**
  Select either Hardware Rendering (OpenGL) or Software Rendering. See “Window Objects” in Chapter 14 (“Using Destination Objects”) of Objects and Object Graphics for information about the differences between the two rendering systems.

**Edit Preferences**

This tab allows you to set the way in which IDL compiles files. By default, IDLDE asks if you would like to save changes. You can also set IDLDE to make a backup copy of the source file.
Startup Preferences

This tab allows you to specify the location of a file to be run when IDLDE starts.

Select IDL Main Dir ...
Click on this button to start the Select IDL Main Dir dialog, which shows you where the main IDL directory is located. The default is the location you specified when you installed IDL. There is no reason to change this entry. The location of the home IDL directory is shown primarily for informational purposes. You must restart IDL for any changes to take effect.

Select Working Directory
Click on this button to start the Select Working Directory dialog. You can specify the initial working directory for future IDL sessions. The General Preferences tab contains an option, described in “Change Directory on Open” on page 59, which also affects the Working Directory.

Select Startup File
Click on this button to start the Select Startup File dialog. You can specify the name of an IDL batch file to be executed automatically each time IDL is run. For example, to execute the commands in a batch file named MYBATCH.PRO, located in the /home/user directory, use:

/home/user/MYBATCH.PRO

Disable the use of the startup file by selecting the “Don’t Use Startup File” button.
Caution  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. See “Startup File” on page 33 for more information.

Font Preferences
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 - Control Panel
- Edit - Editor windows
- Log - Output Log
- Command - Command Input Line.

Path Preferences
This tab allows you to control where IDL looks for procedures and functions. Entries into the IDL Files Search Path are appended to the system variable !PATH.

Note  You must restart IDLDE for changes to take effect.

Figure 3-5: The Fonts Preferences and Paths Preferences dialogs
**IDL Files Search Path**

This field tells IDL where to look for procedures and functions.

Select one of the paths by clicking on it; it becomes highlighted. You can select more than one path at a time by clicking once with your left mouse button on the first path, and dragging the mouse down to the last path you want to select.

- **Subdirectory checkboxes**
  
  To specify a directory tree that includes all of that directory's subdirectories, click on the checkbox to the left of a path, placing an “x” in front of the entry.

- **Up and Down Arrow buttons**
  
  You can move the selected path up or down through the list by clicking on the up or down buttons located directly below the IDL Files Search Path list. A second click on a selected path causes it to become outlined, but not selected. You can scroll through the list by pressing the up and down arrows on your keyboard after either selecting or outlining one of the paths.

- **Insert...**
  
  To add a path to the IDL Files Search Path list, click on “Insert...” to start the Select Path 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.

- **Remove**
  
  Click on “Remove” to delete the selected path.

- **Expand**
  
  Click on “Expand” to list a selected path’s subdirectories directly beneath the path. The expanded path is then unselected and any subdirectories are selected so you can cancel the expansion by immediately clicking “Remove”. The initial path and any expanded subdirectories are automatically unchecked to prevent subdirectory searching.

See “Executing Program Files” on page 29 for more information.

---

**Using Resources to Customize IDL**

**X Resources in Brief**

The component widgets of an X Window 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.

**Editing Resource Files**

There are two resource files used to customize IDLDE. 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 will be lost with a new installation.

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 preferences; it is modified when preferences are saved and shouldn’t be modified externally.

**Reserving Colors**

When IDL starts, it attempts to secure entries in the shared system colormap for use when drawing graphics. If the entry `Idl.colors` exists in the `Idl` resource file, IDL will 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 \texttt{Idl.colors} resource 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:

\begin{verbatim}
Idl.colors = -10
\end{verbatim}

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.

If you experience colormap flashing when using the IDLDE, but not when you use the plain tty interface, try adjusting the number of colors used by the IDLDE interactively, using the \texttt{-colors} startup flag. For example,

\begin{verbatim}
idle -colors -15
\end{verbatim}

starts the IDLDE and allocates all but 15 of the currently available colors. When you find an appropriate number of colors to reserve, you can set the \texttt{idlde.colors} resource in the \texttt{Idl} resource file or in your personal \texttt{.idlde} file accordingly.

\section*{Command Line Options}

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 (\texttt{Idl}) and user resource files (\texttt{.idlde}). For more information about resources, see "Using Resources to Customize IDL" on page 66. Under VMS, command line switches are preceded by a / rather than a -.

\textbf{Example} Type the following at the operating system command line to start IDLDE using separate main-level windows to display files:

\begin{verbatim}
idlde -multi
\end{verbatim}

\texttt{Unix}

\begin{verbatim}
IDLDE/MULTI
\end{verbatim}

\texttt{VMS}

The available command line flags follow:

\begin{itemize}
\item \texttt{-e file [-e file\textsubscript{1} -e file\textsubscript{2}...]} \\
  \texttt{/EDIT=(file [, file\textsubscript{1}, file\textsubscript{2}...])}
\end{itemize}

Opens specified files at startup.
-colors \text{n}  
/COLORS=\text{n}

If specified, IDL attempts to allocate \text{n} colors specified from the shared colormap. If there aren't enough colors available, a private colormap with \text{n} colors is used instead.

Specifying a negative value for the colors flag causes IDL to attempt to use the shared colormap, allocating all but the specified number of colors. For example:

```
idlde -colors -15
```

allocates all but 15 of the currently available colors for the IDLDE. This allows other applications that might need their own colors to run in tandem with IDL.

The related resource is `idlde.colors`.

-nocommand  
/NOCOMMAND

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

-command  
/COMMAND

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

-nocontrol  
/NOCONTROL

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

-control  
/CONTROL

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

-nolog  
/NOLOG

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

-log  
/LOG

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

**Using IDL**

- **-nostartup**  
  /NOSTARTUP  
  Does not execute startup file on startup (including IDL_STARTUP). The related resource is `Idl*idlde.noStartupFile: True`.

- **-startup**  
  /STARTUP  
  Executes startup file on startup (including IDL_STARTUP). The related resource is `Idl*idlde.noStartupFile: False`.

- **-startupfile “file”**  
  /STARTUPFILE="file"  
  Executes “file” at startup (overrides IDL_STARTUP environment variable). If Startupfile is not specified, the environment variable IDL_STARTUP is used as the startup file (if defined). The related resource is `Idl*idlde.startupFile: file` where `file` is the full path name of the startup file.

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

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

- **-path "path"**  
  /PATH="path"  
  Append “path” to the IDL path (defined using IDL_PATH environment variable). The related resource is `Idl*idlde.path: path` where `path` is the full path to be appended.

- **-quiet**  
  /QUIET  
  Inhibits display of the IDL startup announcement and message of the day (motd) file.

- **-readonly**  
  /READONLY  
  Opens files as read-only. The related resource is `Idl*idlde.readOnly: True`.
Chapter 3: The IDL for Motif Interface

- **readwrite**
  
  Open files as read-writeable. The related resource is `Idl*idlde.readOnly: 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`.

- **VAX_FLOAT**
  
  This option is available only in IDL for VMS. Set this qualifier to change the default value of the VAX_FLOAT keyword of the CALL_EXTERNAL and OPEN procedures to be TRUE. Starting IDL with this qualifier allows old code that is written to assume IDL reads and writes VAX format floating-point numbers to continue reading and writing that format without requiring changes to the IDL code. There are three caveats:

  1. Internally, IDL is still using IEEE floating-point numbers.
  2. This option should be used as a transitional aid prior to converting the code to work with IEEE math. It is not a good long term strategy to use IDL in this mode.
  3. There is no such support for LINKIMAGE routines, which must be rebuilt to use the IEEE floating-point standard.
You can also change this default at runtime using the VAX_FLOAT function.

**Note** You should read the warnings on this topic found on the OPEN and CALL_EXTERNAL reference manual pages.

### Modifying the Control Panel

The Control Panel, with the resource name `control`, is located below the 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 with either the `idlButtonsUser` resource, or, for the Macros toolbar only, by clicking Edit in the Macros menu.

The `idlButtonsUser` resource supplies each button's resource name. The resource name details button attributes, such as its label or pixmap, its associated IDL command, and its status bar message.

To add a Control Panel button, make the following modifications to the `.idlde` file:

- Add a new name to the `idlButtonsUser` list. The buttons are created in the order specified.
- Add `labelString` or `labelPixmap` resources. These resources define the button text or image.
- Add an `idlCommand` resource. This is the text of the IDL command to execute.
- Add a `statusString` resource. This is the text that appears in the Status Bar when the cursor is positioned over the new button.

### Bitmaps for Control Panel Buttons

Bitmaps for control panel buttons must conform to the following:

1. The bitmap must be in either XBM (X11 bitmap file) or XPM (X11 system pixmap file) format, with the file extension `.xbm` or `.xpm`.
2. The bitmap must be located in one of the following directories:
Chapter 3: The IDL for Motif Interface

Using IDL

Modifying the Control Panel

Under Unix:

• $IDL_DIR/resource/X11/lib/app_defaults
• $IDL_DIR/resource/X11/lib/app_defaults/bitmaps
• $HOME
• $HOME/bitmaps

Under VM S:

• IDL_DIR:[RESOURCE.X11.LIB.X11.APP-DEFAULTS.BITMAPS]
• SYS$LOGIN

3. The bitmap must be defined in the resource file (Idl, .idlde), for example:

    idlde*control*mybutton*labelPixmap: mybutton

Examples

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

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

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

    Idl*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:

    Idl*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`.

Command Stream Substitutions

The `idlCommand` resource specifies the IDL command that is entered into the input command stream when the respective button is clicked. You can use `%` substitutions to include certain types of information into the command:

<table>
<thead>
<tr>
<th>% Symbol</th>
<th>Substitution</th>
</tr>
</thead>
<tbody>
<tr>
<td>%S</td>
<td>The text of the current selection.</td>
</tr>
<tr>
<td>%F or %P</td>
<td>The filename associated with the current IDL Editor.</td>
</tr>
<tr>
<td>%N</td>
<td>The base name of the filename (without path and suffix).</td>
</tr>
<tr>
<td>%B</td>
<td>The base name of the filename (without path, but with a suffix)</td>
</tr>
<tr>
<td>%L</td>
<td>The line number with the current insertion point.</td>
</tr>
<tr>
<td>%%</td>
<td>Inserts “%”.</td>
</tr>
</tbody>
</table>

Table 3-3: Command Stream Substitutions

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 a 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*button name*idlAction: action
```

where `button name` is the name of the button and `action` is the name of the action routine.

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 Table 3-4 must be set:

<table>
<thead>
<tr>
<th>Action Routines</th>
<th>Using IDL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Chapter 3: The IDL for Motif Interface

Using IDL

Action Routines

<table>
<thead>
<tr>
<th>Argument</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>SET</td>
<td>Set a breakpoint on the current line.</td>
</tr>
<tr>
<td>CLEAR</td>
<td>Clear the breakpoint on the current line.</td>
</tr>
<tr>
<td>TOGGLE</td>
<td>Toggle (SET or CLEAR) the state of the breakpoint on the current line.</td>
</tr>
<tr>
<td>COMPLEX</td>
<td>Display breakpoint dialog to set a complex breakpoint.</td>
</tr>
<tr>
<td>LIST</td>
<td>List all currently set breakpoints</td>
</tr>
</tbody>
</table>

Table 3-4: Breakpoint Arguments

**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 Input 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 Table 3-5 must be set:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>FILE</td>
<td>Compiles the currently-active file.</td>
</tr>
<tr>
<td>TEMPORARY</td>
<td>Compiles the currently-active file into a temporary file</td>
</tr>
<tr>
<td>RESOLVE</td>
<td>Resolves all referenced and uncompiled IDL routines</td>
</tr>
</tbody>
</table>

Table 3-5: 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 Table 3-6 must be set:

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

Table 3-6: 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 from Table 3-7 must be set:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>NEW</td>
<td>Creates a new editor window.</td>
</tr>
<tr>
<td>OPEN</td>
<td>Opens an existing file.</td>
</tr>
</tbody>
</table>

Table 3-7: Editor Window Arguments
IdlFileReadOnly

Use `IdlFileReadOnly` to specify the read/write status of the currently-active editor window. One of the arguments from Table 3-8 must be set:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAVE</td>
<td>Saves the contents of the currently-selected editor window.</td>
</tr>
<tr>
<td>PRINT</td>
<td>Prints the contents of the currently-selected editor window.</td>
</tr>
</tbody>
</table>

Table 3-7: Editor Window 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:

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

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”.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>READONLY</td>
<td>Disable editing of the currently-selected editor window.</td>
</tr>
<tr>
<td>READWRITE</td>
<td>Enables editing of the currently-selected window.</td>
</tr>
</tbody>
</table>

Table 3-8: Read/Write Arguments
IdlRecallCommand

Use IdlRecallCommand to recall 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 Table 3-9 may be used:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIND</td>
<td>Displays a search dialog (default).</td>
</tr>
<tr>
<td>FINDAGAIN</td>
<td>Finds the next occurrence of the specified string.</td>
</tr>
<tr>
<td>FINDSELECTION</td>
<td>Finds next occurrence of the current selection.</td>
</tr>
<tr>
<td>ENTERSELECTION</td>
<td>Enters the current selection as the search string in the Find dialog.</td>
</tr>
<tr>
<td>REPLACE</td>
<td>Replaces the search string, with a specified replacement string.</td>
</tr>
<tr>
<td>REPLACEFIND</td>
<td>Finds the next occurrence of the search string, and replaces it with the specified replacement string.</td>
</tr>
</tbody>
</table>

Table 3-9: 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 Table 3-10 must be set.

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 Table 3-11 must be set:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>CASCADE</td>
<td>Arrange open windows in a staggered, overlapping fashion.</td>
</tr>
<tr>
<td>TILE</td>
<td>Arrange all windows in a non-overlapping fashion.</td>
</tr>
<tr>
<td>MULTI</td>
<td>Open windows outside the IDLDE interface.</td>
</tr>
<tr>
<td>SINGLE</td>
<td>Display the most recent window on the Multiple Document Panel.</td>
</tr>
</tbody>
</table>

Table 3-11: Editor Window Display Arguments

### Modifying the Macros Menu

You can adjust the Macros menu. IDLDE looks for a resource named `macrosList`. If `macrosList` is found, its value supplies the resource names of the additional buttons to be added to the Macros menu. This allows system-
dependent commands to be added to IDLDE, which simplifies the process of calling external editors such as emacs or vi.

**Example** To add the menu item “File in Big Vi” to the Macros menu add the following resources to .idlde:

```
Idl*idlde*menubar*macrosMenu*macrosListUser: bigViXterm
  Define a new menu item.

Idl*idlde*menubar*macrosMenu*bigViXterm*labelString: 
  File in Big Vi
  Assign text to the defined menu item.

Idl*idlde*menubar*macrosMenu*bigViXterm*idlCommand: 
  SPAWN,’xterm -geometry 80x50 -e vi -c %L %F &
  Define a procedure to call up the vi editor.

Idl*idlde*menubar*macrosMenu*bigViXterm*statusString: 
  Run vi in the Big Xterm window
  Assign text for the status string.
```

**Modifying other resources:**

You can modify other resources in your user resource file. Check the `Idl` resource file for available resources.

**Example** To set your own IDLDE default font:

```
Idl*idlde*fontList: -*-Prestige-Medium-R-*-*-*-110-100-100-*-*-ISO8859-1
```

**Using External Editors**

Files in Editor windows are used mainly for rudimentary editing and debugging, and do not offer any sophisticated editing features. If you wish to use more sophisticated editing features, choose one of the external editors offered in the Macros menu.

The Macros menu default items are the Unix standard editors vi and emacs (or XEmacs, a highly sophisticated editor from the Free Software Foundation). Emacs also supports an IDL language mode (idl-mode), which offers chroma (or font) highlighting of the IDL language construct, and many other editing features. Please consult the IDL FAQ for the latest version of idl-mode (idl.el)
Using the IDL Editor

The IDL Editor provides a simple text file editing facility and is the default editing environment provided at startup.

Table 3-12 lists some shortcuts for maneuvering in the IDL Editor window:

<table>
<thead>
<tr>
<th>Key</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+A</td>
<td>Move cursor to beginning of line.</td>
</tr>
<tr>
<td>Ctrl+B</td>
<td>Move cursor back one word.</td>
</tr>
<tr>
<td>Ctrl+D</td>
<td>Delete next character.</td>
</tr>
<tr>
<td>Ctrl+E</td>
<td>Move cursor to end of line.</td>
</tr>
<tr>
<td>Ctrl+F</td>
<td>Move cursor forward one word.</td>
</tr>
<tr>
<td>Ctrl+K</td>
<td>Delete everything in the current line to the right of the cursor.</td>
</tr>
<tr>
<td>Ctrl+U</td>
<td>Delete everything in the current line to the left of the cursor.</td>
</tr>
<tr>
<td>Ctrl+V</td>
<td>Delete word to the left of the cursor.</td>
</tr>
<tr>
<td>Ctrl+End</td>
<td>Move to end of file</td>
</tr>
<tr>
<td>Ctrl+Home</td>
<td>Move to beginning of file</td>
</tr>
<tr>
<td>Ctrl+→</td>
<td>Move right one word</td>
</tr>
<tr>
<td>Ctrl+←</td>
<td>Move left one word</td>
</tr>
<tr>
<td>End</td>
<td>Move to end of current line</td>
</tr>
<tr>
<td>Home</td>
<td>Move to beginning of current line</td>
</tr>
<tr>
<td>PgUp</td>
<td>Move to previous screen</td>
</tr>
<tr>
<td>PgDn</td>
<td>Move to next screen</td>
</tr>
<tr>
<td>Tab</td>
<td>Indent text lines one tab-stop right</td>
</tr>
<tr>
<td>←→↑↓</td>
<td>Move cursor left or right one character, up or down one line</td>
</tr>
</tbody>
</table>

Table 3-12: Editor Window Key Definitions
Chapter 4

The IDL for Windows Interface

The following topics are covered in this chapter:

- The Main IDL Window ................. page 84
- IDLDE Windows........................ page 86
- The Menu Items........................ page 88
- Keyboard Shortcuts.................... page 100
- Customizing IDL ...................... page 102
- Using the IDL Editor............... page 110
- Windows IDL Differences ........... page 112
IDL for Windows has a convenient multiple-document interface called the IDL Development Environment (IDLDE) that includes built-in editing and debugging tools. This chapter describes the IDLDE.

See “Environment Variables Used by IDL” on page 22 for additional details on the IDL environment.

The Main IDL Window

When you start IDL, the main IDL window appears (shown in Figure 4-1.) The seven sections of this window are described below.

Figure 4-1: The Main IDL Window with Multiple Documents.
Docking/Undocking

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 Input Line. Click on the border and drag the left mouse button. You will notice the outline of the section you have chosen moving with your mouse. When you are satisfied with a location, let go of 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 Input Line will remain between the Menu Bar and the Status Bar when docked. They can be docked in any order against 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 on the desktop instead of docking to the nearest available side of the IDLDE.

Menu Items

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

Toolbars

You can choose from three toolbars: Standard, Run & Debug, and Macros. To change the toolbars displayed, select any combination from the Toolbars pulldown menu in the Windows menu. In addition, when you open a GUIBuilder window, 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.

Multiple Document Panel

The top section of the main IDL window is where IDL Editor windows are displayed. The Multiple Document Panel can be sized or made invisible by moving the separator at the top of the Output Log or the Variable Watch Window, depending on your IDLDE Preferences setup.

Output Log

Output from IDL is displayed in the Output Log window, which appears by default when IDLDE is first started. The Output Log area can be sized by moving the separator attached to the top of the Output Log. You can hide/display the Output Log by clicking the “Output Log” toggle item in the Windows menu, by pressing [CTRL+L], or by changing the Layout tab from Preferences in the File...
Menu. If you click the right mouse button while positioned over the Output Log, a popup menu appears allowing you to move to a specified error, clear the contents of the Output Log, or 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. Size the Variable Watch Window by moving the separator attached to the top. You can hide/display the Variable Watch Window by clicking the “Variable Watch” toggle item in the Windows menu, by pressing [CTRL+A], or by changing the Layout tab from Preferences in the File Menu. For more information about the Variable Watch Window, see “The Variable Watch Window” on page 145.

**Command Input Line**

The Command Input Line is a single IDL prompt where you can enter IDL commands. The text output by IDL commands is displayed in the Output Log window. The Command Input Line can be made invisible by clicking the “Command Input” toggle item in the Windows menu, by pressing [CTRL+I] or by changing the Layout tab from Preferences in the File Menu.

If you click the right mouse button while positioned over the Command Input Line, a popup menu appears displaying the command history, with a maximum buffer of 20 entries. If you enter HELP, /RETURN at the Command Input Line, you will see the same results, except that you can specify the number of lines in the recall buffer with the General Preferences tab from the File menu.

You can also open and compile files from the Command Input Line. See “Open... [Ctrl+O]” on page 88 and “Compile filename.pro [Ctrl+F5]” on page 94 for more information.

**Status Bar**

When you position the mouse pointer over a Toolbar button or select an option from a menu in IDLDE, the Status Bar displays a brief description. The Status Bar can be made invisible by clicking the “Status Bar” toggle item in the Windows menu or by changing the Layout tab from Preferences in the File Menu.

**IDLDE Windows**

Three types of windows can be created within the IDLDE: IDL Editor windows, IDL GUIBuilder windows, and IDL Graphics windows.
IDL 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 “New” then File from the File menu, or by selecting “Open” from the File menu. You can also open windows using the toolbar buttons. When you minimize an Editor window, a Windows title bar with the name of the file appears in the Multiple Document Panel.

You can access different files from the Windows menu by clicking on the appropriate numbered file. See “Using the IDL Editor” on page 110 for more information on the IDL Editor.

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.

If a program error or breakpoint is encountered, IDLDE displays the relevant file, opening it if necessary. The line at which the breakpoint or error occurred is marked. See Chapter 6, “Programming Tools”, for more on IDL's debugging commands.

IDL GUIBuilder Windows

IDL GUIBuilder windows allow you to interactively create user interfaces. Then, you can generate the IDL code that defines that interface and code that will contain the event-handling routines. You can modify the code and compile and run the application in the IDLDE.

You can have any number of GUIBuilder windows open simultaneously.

To open a GUIBuilder window, you can select New then GUI from the File menu, or you can select Open from the File menu. You can also open GUIBuilder windows using the toolbar buttons.

When you minimize a GUIBuilder window, a Windows title bar with the name of the file appears in the Multiple Document Panel.

For information about the IDL GUIBuilder, see “Using the IDL GUIBuilder” on page 149.

IDL Graphics Windows

IDL Graphics windows appear when you use IDL to plot or display an image.
You can copy the contents of a Graphics window—Direct or Object—directly to
the operating system clipboard in a bitmap format using Command-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.

Caution  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 106.

The Menu Items

Six menus (File, Edit, Search, Run, Macros, Window, Help) allow you to control
the operation of IDLDE. These menus are described below.

File Menu

The File menu accesses and manipulates files.

New

From this option you can select Editor [Ctrl+N] or GUI. If you select Editor, a
new IDL Editor window is opened. If you select GUI, a new IDL GUIBuilder is
opened. Each window is titled Untitledn or UntitledPrcn until saved with another
name, n being the numerical order of the new window opened.

For information about the IDL GUIBuilder, see “Using the IDL GUIBuilder” on
page 149.

Open... [CTRL+O]

Select this option to open a text file or a GUIBuilder *.prc portable resource file
for editing. The Open dialog appears. Select the file you want to open and click
“Open”. You can select a continuous range of files by holding down the Shift key
after selecting the first file. You can also select multiple separated files by selecting
each file while holding down the Control key. A new IDL Editor window is
created to contain each text file.

You can also open text files from the Command Input Line. To open text files,
enter the following at the IDL prompt:

.EDIT file1 [file2 ... file_n]

where file is the name of the text file you want to open. If the path is not specified
in the Path Preferences from the File menu, you must enter the full path for file.
See “.EDIT” on page 33 of the IDL Reference Guide 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.

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 saving and re-compiling simply re-runs the previously-compiled version. Select the “Compile” option in the Run menu to return to the main program level and re-compile the routine, without incorporating recent changes. Select “Compile from Memory” in the Run menu to save and compile recent changes to a temporary file.

Save As...
Select this option to save the contents of an IDL Editor window to a specified filename. The Save As dialog appears.

Revert to Saved
Select this option to reload the last saved version of the document.

Caution Unsaved changes are lost without warning.

Generate .pro
From this option, select Widget, Event, or Both 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 your options:

- Widget: Generates the widget definition code to a *.pro file.
- Event: Generates the event-handler callback code to a *_eventcb.pro file.
- Both: Generates both the widget definition code and the event-handler callback code. The event code filename is based on the widget code file you specify. For example, if you enter app1.pro for File name and click Save, the event code file will be named app1_eventcb.pro.

Note If you save both files at the same time, IDL puts the RESOLVE_ROUTINE procedure in the generated widget code, which contains the name of the related event * _eventcb.pro file, which should be compiled and loaded with the widget file. If you generate the files separately, or generate only the widget
file, you must compile the files yourself, or add the routine to the *.pro widget code (which is the only modification you should ever make to the widget code).

For information about the IDL GUIBuilder generated code, see “Generating Files” on page 179.

Print... [CTRL+P]
Select this option to print the contents of the currently-selected window to the currently-active printer. The Print dialog appears. Use the Printers icon in the Microsoft Windows Control Panel (found in the Main program group) to change the currently-selected printer.

Print Setup...
Select this option to change the printer and printing options.

Recent Files
Select this option to view or open recently opened files. This menu item lists the last ten opened files, and it includes both text and GUIBuilder portable resource files. To open a file on this list, select it.

To change the maximum number of files displayed from ten to another number, modify the Current User on Local Machine setting in the registry in the following resource location: IDL\IDLDE\RecentFiles\NumRecentFiles.

Preferences...
Select this option to display a dialog box containing seven tab selections with which you can customize your interaction with the IDLDE environment. The seven categories are: General, Layout, Graphics, Editor, Startup, Fonts and Path. For more information about the Preferences, see “Customizing IDL” on page 102.

• General
This tab allows you to specify how the IDLDE 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
This tab allows you to specify the location and size of the main window on the screen. You can also designate which components of the IDLDE will be visible.

• Graphics
This tab allows you to set the layout of the graphics window and to specify the backing store.
Chapter 4: The IDL for Windows Interface

- **Editor**
  This tab allows you to customize the built-in IDL editor and also offers several compiling options.

- **Startup**
  This tab allows you to specify the main directory, the working directory, and the startup file.

- **Fonts**
  This tab allows you to specify different fonts, styles, and sizes for the Editor, Command Input Line and Output Log.

- **Path**
  This tab allows you to specify the IDL Files Search Path. Your entries are appended to the system variable !PATH.

**Exit**
Select this option to exit IDL for Windows. All IDL Editor windows are closed before exiting. If text in an Editor window has changed, you are asked if you want to save it before exiting.

**Edit Menu**

**Undo [CTRL+Z]**
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]**
Select this option to redo previously undone editing actions. Multiple “redo” operations are supported; the first redo reverses the most recent undo, etc.

**Cut [CTRL+X]**
Select this option to remove currently-selected text from an IDL Editor window or the Command Input Line to the Windows clipboard.

**Copy [CTRL+C]**
Select this option to copy the currently-selected text in an IDL Editor window, Output Log window, or command input 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]
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.

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]
Use this option to clear the entire contents of an IDL Editor window.

Clear Log
Use this option to clear the entire contents of the Output Log.

Properties
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” on page 166.

Menu
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” on page 171.

Search Menu
Find... [CTRL+F]
Select this option to find text in an IDL Editor window or windows. The Search dialog appears.

Enter the text to find in the field marked “Search for:” or choose an entry from the pulldown list of recent search terms. To replace the found text with new text, check the “Replace with” checkbox. Enter replacement text in the field or choose an entry from the pulldown list of recent replacement terms.

Click “Find next” to highlight the search text in your file. 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.

**Find Again [F3]**
Select this option to repeat the previous Find.

**Find Selection [CTRL+E]**
Select this option to find the next occurrence of the selected text in an IDL Editor window.

**Replace... [CTRL+H]**
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 Again [SHIFT+F3]**
Select this option to repeat the previous Replace.

**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]**
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.

**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 6, “Programming Tools” for more information.
Compile filename.pro [CTRL+F5]
Select this option to compile filename.pro. 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 Input Line, with the appropriate Editor window selected in the Multiple Document Panel.

You can also compile files from the Command Input 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” on page 32 of the IDL Reference Guide for a more detailed explanation.

Compile filename.pro from Memory [CTRL+F6]
Select this option to compile filename.pro from the last-saved version of the file, without saving or implementing recent changes. Selecting this option is the same as entering .COMPILE -f at the Command Input Line. See “.COMPILE” on page 32 of the IDL Reference Guide for a more detailed explanation.

Compile All
Select this option to compile all currently open *.pro files.

Run filename [SHIFT+F5]
Select this option to execute the file, filename, contained in the currently-active editor window. Selecting this option is the same as entering the procedure name at the Command Input Line or using the .GO executive command at the Command Input Line. If the file is interrupted while running, selecting this option resumes execution; it is the same as entering .CONTINUE at the Command Input Line. For more information, see .CONTINUE and “.GO” on page 33 of the IDL Reference Guide

Caution 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
Select this option to iteratively compile all uncompiled IDL routines that are referenced in any open and compiled files. Selecting this option is the same as
Chapter 4: The IDL for Windows Interface

...entering RESOLVE_ALL, /QUIET at the Command Input Line. The QUIET keyword suppresses informational messages. See “RESOLVE_ALL” on page 952 of the IDL Reference Guide for a more detailed explanation.

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” on page 141.

Test GUI [CTRL+T]
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 close X in the upper-right corner of the application window of the running test application.

Break [CTRL+BREAK]
Select this option to interrupt program execution. IDL inserts a marker to the left of the line at which program execution was interrupted.

Reset [CTRL+R]
Select this option to reset the IDL environment. Selecting this item is the same as entering the following at the Command Input Line:

RETALL
WIDGET_CONTROL, /RESET
CLOSE, /ALL
HEAP_GC, /VERBOSE


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 “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 Input Line. See “.STEP” on page 35 of the IDL Reference Guide 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 “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 .STEP OVER at the IDL Command Input Line. See “.STEP OVER” on page 36 of the IDL Reference Guide for a more detailed explanation.

**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 Input Line. See “.OUT” on page 33 of the IDL Reference Guide for a more detailed explanation.

**Trace...**

Select this option to access the Trace Execution dialog. You can modify the interval between successive .STEP or .STEP OVER 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” on page 36 of the IDL Reference Guide 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” on page 229 of the IDL Reference Guide for a more detailed explanation.

**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” on page 33 of the IDL Reference Guide for a more detailed explanation.

**Set Breakpoint [F9]**

Select this option to set a breakpoint on the current line. Selecting this item is the same as entering the following at the IDL Command Input Line:
BREAKPOINT, ['file',] line

where “file” is the file to set a breakpoint within, and line designates the line number at which the breakpoint is set.

See “BREAKPOINT” on page 229 of the IDL Reference Guide for a more detailed explanation.

Set Complex Breakpoint...

Select this option to access the Set Complex Breakpoint dialog. Complex breakpoints may function only once, or may function only after being “hit” a specified number of times. Selecting this item is the same as setting the AFTER and ONCE keywords for the BREAKPOINT procedure at the IDL Command Input Line.

Enter the source file in which to set a breakpoint in the “Source file:” field. The default field is the one in which the cursor is positioned. Click “Browse” to search through available directories. Enter the line number at which to place the breakpoint in the “Line number:” field. The default is the line at which the cursor is currently positioned. You can also specify how many times the line must be hit in order to interrupt execution. Click “Break once only” to interrupt execution after encountering the line for the first time or click “Break after being hit __ times” and enter the number of hits after which execution should be interrupted into the given field.

See “BREAKPOINT” on page 229 of the IDL Reference Guide for a more detailed explanation.

Clear All Breakpoints

Select this item to clear all breakpoints.

List Breakpoints

Select this item to list all breakpoints currently set, in all compiled programs. Selecting this item is the same as entering HELP, /BREAKPOINTS at the IDL command prompt.

Up Stack [CTRL+Up]

Select this option to move up the call stack by one.

Down Stack [CTRL+Down]

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 Input Line. See “HELP” on page 537 of the IDL Reference Guide for a more detailed explanation.

Macros Menu

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”.

Print Var
Select this option to print the selected variable. Selecting this item is the same as entering PRINT, x at the IDL Command Input Line, where x is the selected variable.

Help On Var
Select this option to list attributes of the selected variable. Selecting this item is the same as entering HELP, /STRUCTURE at the IDL Command Input Line, where x is the selected variable.

Insight
Select this option to access IDL’s Insight application.

Demo
Select this option to access IDL’s Demo application.
Window Menu

Next [F6]
Select this option to shift IDL’s focus to the next numbered editor window.

Previous [SHIFT+F6]
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
Select this option to tile all the IDL Editor windows on top of one another within the main window.

Tile Vertically
Select this option to tile all the IDL Editor windows side-by-side within the main window.

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.

Command Input [CTRL+I]
If this menu item has a check mark by it, the IDL Command Input Line is visible in the main IDL window. If this item does not have a check mark next to it, the IDL command input line is not visible. Click on this menu item to toggle between the two states.

Output Log [CTRL+L]
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. Click on this menu item to toggle between the two states.

Variable Watch [CTRL+A]
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 next to it, the Variable Watch Window is not visible. Click on this menu item to toggle between the two states.
Keyboard Shortcuts

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 Items.

Status Bar
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.

Help Menu

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.

Help on the IDL Dev Env...
Select this menu item to display this chapter of Using IDL.

Help on the IDL Language...
Select this menu item to display information on the IDL language.

Help on Help...
Select this menu item to learn about how to use Help.

About IDL...
Select this option to display information on the IDL version in use.

Keyboard Shortcuts
Most of the menu options can be accessed from the keyboard instead of clicking on the menus. Table 4-1 lists all of the available keyboard equivalents. Note that these equivalents are also shown to the right of each menu item in the menus themselves.
Chapter 4: The IDL for Windows Interface

<table>
<thead>
<tr>
<th>Keyboard Shortcut</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+A</td>
<td>Toggle Variable Watch Window</td>
</tr>
<tr>
<td>Ctrl+C</td>
<td>Copy selection to clipboard</td>
</tr>
<tr>
<td>Ctrl+D</td>
<td>Go to definition</td>
</tr>
<tr>
<td>Ctrl+E</td>
<td>Find highlighted selection</td>
</tr>
<tr>
<td>Ctrl+F</td>
<td>Start Find dialog</td>
</tr>
<tr>
<td>Ctrl+G</td>
<td>Start Go To Line dialog</td>
</tr>
<tr>
<td>Ctrl+H</td>
<td>Start Replace dialog</td>
</tr>
<tr>
<td>Ctrl+H</td>
<td>Toggle Command Input Line</td>
</tr>
<tr>
<td>Ctrl+L</td>
<td>Toggle Output Log</td>
</tr>
<tr>
<td>Ctrl+N</td>
<td>Open new file</td>
</tr>
<tr>
<td>Ctrl+O</td>
<td>Open file</td>
</tr>
<tr>
<td>Ctrl+P</td>
<td>Print currently-active file</td>
</tr>
<tr>
<td>Ctrl+Q</td>
<td>Exit IDL</td>
</tr>
<tr>
<td>Ctrl+R</td>
<td>Reset IDL environment</td>
</tr>
<tr>
<td>Ctrl+S</td>
<td>Save currently-active file</td>
</tr>
<tr>
<td>Ctrl+V</td>
<td>Paste selection from clipboard at insertion point</td>
</tr>
<tr>
<td>Ctrl+X</td>
<td>Cut selection to clipboard</td>
</tr>
<tr>
<td>Ctrl+Y</td>
<td>Redo last undo</td>
</tr>
<tr>
<td>Ctrl+Z</td>
<td>Undo previous editing action</td>
</tr>
<tr>
<td>Ctrl+Break</td>
<td>Interrupt execution</td>
</tr>
<tr>
<td>Ctrl+Del</td>
<td>Clear current Editor window</td>
</tr>
<tr>
<td>Ctrl+F1</td>
<td>Start Contents of Online Help</td>
</tr>
<tr>
<td>Ctrl+F5</td>
<td>Compile currently-selected file</td>
</tr>
<tr>
<td>Ctrl+F7</td>
<td>Execute file to return</td>
</tr>
<tr>
<td>Ctrl+F8</td>
<td>Continue processing until program returns: .OUT</td>
</tr>
<tr>
<td>Ctrl+Up arrow</td>
<td>Move up call stack</td>
</tr>
<tr>
<td>Ctrl+Down arrow</td>
<td>Move down call stack</td>
</tr>
<tr>
<td>Delete</td>
<td>Delete selection</td>
</tr>
<tr>
<td>F1</td>
<td>Start Find Topic in Online Help</td>
</tr>
<tr>
<td>F3</td>
<td>Repeat last Find entry</td>
</tr>
<tr>
<td>F5</td>
<td>Run / Continue stopped program: .CONTINUE</td>
</tr>
</tbody>
</table>

Table 4-1: Keyboard Shortcuts, Windows IDLDE
Customizing IDL

Various defaults for IDL can be customized using the IDL Preferences dialog box. Select “Preferences” from the IDL File menu to display a cascading list of preferences. The IDLDE Preferences dialog box contains seven tab selections with which you can customize your interaction with the IDLDE environment. The seven categories are: General, Layout, Graphics, Editor, Startup, Fonts, and Path. Click “Reset” to restore the settings to the values from the start of the current IDL session.

**General Preferences**

This tab allows you to specify how the IDLDE 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.

**Program**

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

- **Show Splash Screen**
  Select this option to show IDL’s splash screen on startup.

- **Save Settings on Exit**
  Select this option to save all the preferences settings from the current IDL session to be applied to future IDL sessions.

- **Confirm Exit**

<table>
<thead>
<tr>
<th>Keyboard Shortcut</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>F6'</td>
<td>Display next-numbered Editor window</td>
</tr>
<tr>
<td>F7</td>
<td>Execute file to cursor</td>
</tr>
<tr>
<td>F8</td>
<td>Execute a single statement: .STEP</td>
</tr>
<tr>
<td>F9</td>
<td>Toggle breakpoint</td>
</tr>
<tr>
<td>F10</td>
<td>Execute a single statement: .STEPOVER</td>
</tr>
<tr>
<td>Shift+F3</td>
<td>Repeat last Replace entry</td>
</tr>
<tr>
<td>Shift+F5</td>
<td>Execute currently-selected file</td>
</tr>
<tr>
<td>Shift+F6</td>
<td>Display previously-numbered Editor window</td>
</tr>
</tbody>
</table>

Table 4-1: Keyboard Shortcuts, Windows IDLDE
• Users share preferences and macros

If this checkbox is selected, all users are able to use and edit the same set of preferences and macros. If it is not checked, each user has their own set and will not be able to affect other users’ preferences and macros.

Log and Command windows

The performance of IDL can depend upon the number of saved lines. The amount of memory required for greater numbers of saved lines can affect the speed at which IDL will run. Click in the field next to each description and enter your adjusted value.

• Number of lines saved in the recall

This field controls the maximum number of lines saved in the recall buffer. There are three ways to access the contents of the recall buffer, all of which are limited by this field. After locating the cursor in the Command Input Line, you can press your up arrow key to scroll through your last entries. You can also enter HELP, /RECALL in the Command Input Line or click on your right mouse button while positioned over the Command Input Line to display your entries up to the limit specified by the recall buffer. The default is 20.

• Number of lines to display in the log

Figure 4-2: General Preferences Dialog.
This field controls the minimum number of lines retained by the Output Log window. The default is 1000 lines.

- **Number of log lines to delete at limit**
  This field controls the number of lines to delete at a time until the limit is reached again. The default is 100.

**Files**

You can change the way in which IDL handles opening files. Click on the following checkboxes to apply or disable the options:

- **Change Directory on Open**
  Select this checkbox to change the working directory upon opening a file to the opened file’s directory.

- **Open Files Read Only**

- **Clip long filenames**
  Select this checkbox to truncate filenames so that they conform to the Windows 8.3 filename format. By default, a file is opened without changing the filename. See “！WARN” on page 41 of the IDL Reference Guide for more information.

**Layout Preferences**

**Main window**

By default the size of the window is 1/4 of the screen size (i.e., 1/2 the screen width and 1/2 the screen height). The window is positioned such that the lower-left corner of the window is at the lower-left corner of the screen. Click on “Default Layout” to use these settings.

To change the layout, click on “Specify layout”, which allows you to adjust the positioning of the window with the “Left” and “Top” fields and to adjust the size of the window with the “Width” and “Height” fields.

- **Left**
  The horizontal location of the upper-left corner of the main IDL window (in pixels) relative to the left side of the screen. The default is 0.

- **Top**
  The vertical location of the upper-left corner of the main IDL window (in pixels) relative to the top of the screen. The default is 1/2 the screen height.
• Width
The width of the main IDL window in pixels. The default is 1/2 the screen width. The value in this entry reflects the current width of the main IDL window.

• Height
The height of the main IDL window in pixels. The default is 1/2 the screen height. The value in this entry reflects the current height of the main IDL window.

Click on “Remember Layout” to apply the settings to future IDL sessions.

Show Window
By default, all the listed options are checked, signifying that they are all visible in the IDLDE main window. Click on the checkboxes to show or hide the sections.

For more information about the above window sections, see “The Main IDL Window” on page 84.
Graphics Preferences

This tab allows you to control the layout and size of the open Graphics windows in the Main Document Panel and also to control the backing store applied to all Graphics windows.

Window layout

All open Graphics windows can be arranged in either a tiled or cascading fashion.

- **Tile**
  The Tile option arranges all Graphics windows on the desktop side-by-side, without any overlap.

- **Cascade**
  The Cascade option arranges all Graphics windows on the desktop so that they overlap.

- **Width**
  This field specifies the width of IDL graphics windows, in pixels. The default is 1/2 of the total screen width.

- **Height**
  This field specifies the height of IDL graphics windows, in pixels. The default is 1/2 of the total screen height.

- **Use 1/4 the screen size**
  Click this check box to fill in the Height and Width fields with 1/2 of the height and width of your display.

Backing Store

When backing store is enabled, a copy of each Graphics window is kept in memory. The copy of the window is used to refresh the window when it has been covered and uncovered. IDL’s performance increases for no backing store, since the amount of memory required to save files can affect the speed at which IDL will run.

See “Backing Store” on page 144 of the IDL Reference Guide for more information.

- **None (direct-draw), RETAIN = 0**
  Click None to disable backing store. This option does not keep a copy of the window, allowing the highest performance in terms of speed.
• System buffered, RETAIN = 1
  Click this option to request backing store from the Windows server. This option is the default.

• Bitmap buffered, RETAIN = 2
  Click this option (the default) to have IDL maintain backing store. Most users should keep this value set to 2.

**True Type Fonts**
Enter the number of TrueType characters to save triangulation information for. 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.

**Default object graphics renderer**
• Hardware (Open GL)
• Software

See “Window Objects” in Chapter 14 of Objects and Object Graphics for information about the differences between the two rendering systems.

**Editor Preferences**
This tab allows you to control the appearance and performance of the built-in IDL Editor, and also to set the way in which IDL compiles files.
For more information, see “Using the IDL Editor” on page 110.

**Startup Preferences**
This tab allows you to control the location of the main IDL directory and any startup file to be run.

**IDL Main Directory**
This field shows where the main IDL directory is located. The default is the location you specified when you installed IDL. There is no reason to change this entry. The location of the home IDL directory is shown primarily for informational purposes. Click “Browse” next to the “Home directory:” field to access the Select Directory dialog.
Working Directory
This field allows you to set the initial working directory for future IDL sessions. The General Preferences tab contains an option, described in “Change Directory on Open” on page 104, which also affects the Working Directory.

Startup file
Use this field to specify the name of an IDL batch file to be executed automatically each time IDL is run. The startup file specifies the startup path for the next session of the IDLDE. Your entries are appended to the system variable !PATH. Click “Browse” next to the “Startup file:” field to access the Select File dialog.

For example, to execute the commands in a batch file named `MYBATCH.PRO`, located in the `C:\DATA` directory, use:

```
C:\DATA\MYBATCH.PRO
```

Note: 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.

See “Startup File” on page 33 of Using IDL for more information.

---

Figure 4-4: Editor and Startup Preferences Dialogs.
**Fonts Preferences**
This tab allows you to specify individual font descriptions for the Editor window, the Command Input Line and the Output Log.

**Path Preferences**
This tab allows you to control where IDL looks for procedures, functions, and help files.

**Search Path**
This field tells IDL where to look for procedures and functions. The search path specifies a list of directories to search.

- **Subdirectory checkboxes**
  To specify a directory tree that includes all of that directory's subdirectories, add a check to the box in front of the entry. Clicking on a checked box, thereby unchecking it, specifies that the subdirectories of the directory will not be searched.

- **Add**
  To add a path to the Search Path, click on “Add” to start the Select Directory dialog. The new path is inserted before a selected path. If none of the paths are selected, the new path is appended to the end of the list.

---

![Font and Path Preferences Dialogs](image)

*Figure 4-5: Font and Path Preferences Dialogs.*

---
• Remove

Click on “Remove” to delete the selected path.

• Expand

Click on “Expand” to list a selected path’s subdirectories directly beneath the path. The expanded path is then unselected and any newly listed subdirectories are selected so you can cancel the expansion by immediately clicking “Remove”. The initial path and any expanded subdirectories are automatically unchecked to prevent subdirectory searching.

• Move Up and Move Down

You can move the selected path up or down through the list by clicking on “Move Up” or “Move Down”. You can scroll through the list by pressing the up and down arrows on your keyboard after selecting one of the paths.

The default path is the IDL directory and all of its subdirectories. See “Executing Program Files” on page 29 for more information.

Message-of-the-Day File

A “message-of-the-day” file can be used to display the contents of an ASCII text file each time IDL is run. To create a message-of-the-day file for IDL for Windows, simply name the desired text file MOTD.TXT or WIN32.TXT and place it in the MOTD subdirectory of the HELP subdirectory of the main IDL directory. Note that the MOTD file is simply an ASCII text file—not an IDL program or batch file. To execute a series of IDL commands, select a startup file as described in “Startup file” on page 108.

If you don’t wish to see the message-of-the-day file each time you start IDL, simply remove or rename the WIN32.TXT or MOTD.TXT file.

Using the IDL Editor

The IDL Editor is a programmer’s-style editor—if you indent a line using the Tab key, the following lines will be indented as well. Use the Shift-Tab key to move left one tab stop. You can move the cursor position within an IDL Editor window using either the mouse or the keyboard. For example, holding down the Control key and clicking the left mouse button moves the cursor to the end of a text line; so does pressing the “End” key. To change the way the tabs are configured, see the Editor tab from Preferences in the File menu.

IDL Editor window key definitions are listed in Table 4-2.
Using IDL

Using the IDL Editor

### Text Selection Modes

IDL Editor windows provide three ways of selecting text.

<table>
<thead>
<tr>
<th>Key</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>←→↑↓</td>
<td>Move cursor left or right one character, up or down one line</td>
</tr>
<tr>
<td>Ctrl+←</td>
<td>Move left one word</td>
</tr>
<tr>
<td>Ctrl+→</td>
<td>Move right one word</td>
</tr>
<tr>
<td>Ctrl+]</td>
<td>Find matching (, {, or [ character</td>
</tr>
<tr>
<td>Ctrl+K</td>
<td>Delete word to the right of the cursor</td>
</tr>
<tr>
<td>Ctrl+U</td>
<td>Make selected text (or the character to the right of the cursor) lower-case</td>
</tr>
<tr>
<td>Ctrl+Y</td>
<td>Redo last undone action</td>
</tr>
<tr>
<td>Ctrl+Z</td>
<td>Undo last action</td>
</tr>
<tr>
<td>Ctrl+Home</td>
<td>Move to beginning of file</td>
</tr>
<tr>
<td>Ctrl+End</td>
<td>Move to end of file</td>
</tr>
<tr>
<td>Ctrl+Shift+U</td>
<td>Make selected text (or the character to the right of the cursor) upper-case</td>
</tr>
<tr>
<td>Ctrl+Shift+Y</td>
<td>Cut Selection (or line containing cursor) to clipboard</td>
</tr>
<tr>
<td>End</td>
<td>Move to end of current line</td>
</tr>
<tr>
<td>Home</td>
<td>Move to beginning of current line</td>
</tr>
<tr>
<td>PageDown</td>
<td>Move to next screen</td>
</tr>
<tr>
<td>PageUp</td>
<td>Move to previous screen</td>
</tr>
<tr>
<td>Shift+Tab</td>
<td>Move cursor one tab-stop left</td>
</tr>
<tr>
<td>Tab</td>
<td>Indent text lines one tab-stop right</td>
</tr>
</tbody>
</table>

Table 4-2: IDL Editor window key definitions
- Stream mode selects text in a stream, beginning with the first character selected and ending with the last character, just as if you were reading the text.

- Line mode selects full lines of text.

- Column mode selects text from one screen column to the next. Selecting text in column mode is similar to drawing a rectangle around the text you wish to select.

Switch between the three modes by clicking the right mouse button while positioned over an Editor window. Select the Selection Mode option to access a pulldown menu with the three text selection modes. The option with a check mark by it is the currently selected text selection mode. If you have text already selected, the selected area will change to reflect the new mode.

**Chromacoded editor**

IDL Editor windows support chromacoding—different types of IDL statements appear in different colors. By default, the IDL Editor uses chroma-coding. The Editor tab from Preferences in the File menu displays the colors used for different words recognized by IDL. Change the Foreground color to change the color of the word itself. Highlight the word by specifying the Background color.

**Turning Chromacoding Off**

By default, the IDL Editor uses chroma-coding. Set the editor to simple black-and-white by unselecting the “Enable chroma-coding (colored syntax)” checkbox in the IDL Editor Preferences tab from the File menu.

**Windows IDL Differences**

The Windows version of IDL implements most of the functionality of other versions. There are a number of differences, however, as described below.
A Note about Microsoft Windows Displays

We recommend that you use a graphics driver that provides at least 800 by 600 pixel resolution with 256 colors. This mode is supported by most VGA (Video Graphics Array) cards that have 512K of memory. VGA cards with 1 Megabyte of memory support 1024 by 768 pixel resolution with 256 colors. Note that EGA (Enhanced Graphics Adapter) cards provide only 16 colors no matter what resolution they support.

Getting Information About Your Graphics Device

Under Windows 95 and Windows NT 4.0, click on “Display” in “Control Panel” under “My Computer” to access the “Settings” tab.

Using a Two-Button Mouse with IDL

IDL supports the use of mice with up to three buttons. However, many mice used with Microsoft Windows systems have only two buttons. Hold down the Control key while pressing the left mouse button to simulate a “middle” mouse button press.

File Manipulation

Reading and Writing Files

Under Windows, a file is read or written as an uninterpreted stream of bytes — there is no record structure at the operating system level. Files are processed as binary or text. Binary files are processed using no translation of characters. Text files are processed by translating the characters that terminate a line. Lines are terminated by the character sequence CR LF (carriage return, line feed). During read operations, if a CR character precedes a LF character, the CR is removed. During write operations, all LF characters are prepended with a CR character.

Text files transferred to or from other operating systems may need to be translated before they will work properly with IDL. The ASSOC, READU, and WRITEU routines operate in binary mode. The PRINT, PRINTF, READ, and READF routines operate in text mode. See “OPEN” on page 783 of the IDL Reference Guide for details on explicitly opening files in text or binary mode.

Note

It is possible, although not recommended, to create a file where portions of the file are written in binary mode and other portions are written in text mode. To properly port such a file to other operating systems, special processing would be required.

Filenames

Under Windows 95 and Windows NT, long filenames are supported by IDL. Names can be up to 255 characters long including extensions. Names can contain
any uppercase or lowercase characters (including spaces) except those shown above. Windows 95 and Windows NT preserve the case of filenames, but the names are not case sensitive (that is, FileName is the same as filename).

While the names of IDL Library files have been truncated to 8 characters, the names of the actual routines remain unchanged.

Save/Restore Files
SAVE/RESTORE files generated with the Windows version of IDL are saved in the XDR format. This format allows data files saved under UNIX, VMS, Windows, and MacOS systems to be easily exchanged.

Positioning File Pointers
Under Windows, the current file pointer can be positioned arbitrarily. Moving the file pointer to a position beyond the current end-of-file causes the file to grow out to that point. Under Windows, the file is padded with arbitrary data.

Running IDL with Fewer than 256 Colors
We recommend that you use a graphics card and driver that support high-resolution and 256 colors. Support for fewer than 256 colors is provided mostly for portable computers. Portables often have LCD displays that can display only between 16 and 64 shades of gray.

If your graphics card and Windows driver support fewer than 256 colors, IDL will run but the results may not be acceptable.

The Windows Palette
Windows reserves the first 20 colors out of all the available colors for its own use. These colors are the ones used for title bars, window frames, window backgrounds, scroll bars, etc. If your graphics driver supports fewer than 20 colors, any windows application that you run, including IDL, must use those reserved colors. This type of color map is called a static color map. IDL can still display graphics, but when it requests a color, Windows supplies the closest available system color. Often, this color choice is not very close to the one you want.

If your driver supports more than 20 colors, the quality of graphics output from IDL improves. Any colors beyond the 20 that Windows needs to reserve can be customized by IDL to be the exact color requested. If you have a 256 color driver, IDL has (by default) 236 colors to work with.

You can display the Windows system colors by opening the “Control Panel”, located in the “Main” program group. Double click on the “Color” icon then click on the “Color Palette >>” button when the Color menu appears. For Win95 and NT 4.0, click on “Display” in “Control Panel” under “My Computer” to access...
the “Appearance” tab. The “Color” dropdown list (combobox) shows the 20 colors reserved by Windows.
Chapter 5

The IDL for Macintosh Interface

The following topics are covered in this chapter:

- The Main IDL Windows .................. 118
- IDL Document Windows ................. 119
- The Menus .................................. 120
- Customizing IDL .......................... 128
- Macintosh IDL Differences ............. 134
IDL for Macintosh includes a built-in editing and debugging environment called the IDL Development Environment (IDLDE). This chapter describes the IDLDE.

The Main IDL Windows

When you start IDL, the IDL Command Output Log, the Command Input Line and the Variable Watch Window appear.

Output Log

The Output Log window displays output from IDL and echoes commands input to IDL. Only one Output Log window can exist at a time.

Command Input Line

The Command Input window is either anchored at the top or bottom of your screen (depending on the setting in the General Preferences dialog) or is free-floating and movable like any other window. An unanchored Command Input window can be moved, resized, or hidden.

The Command Input window contains a single IDL prompt; this is where you enter IDL commands. The commands you type and any output from IDL are displayed in the Output Log.
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. For more information about the Variable Watch Window, see “The Variable Watch Window” on page 145.

IDL Document Windows

IDL 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. If you started IDL by double-clicking on a .pro file, that file will appear in an Editor window. Editor windows can also be created by selecting “New” or “Open” from the File menu.

The line number button box at the bottom left of an IDL Editor window displays the number of the line on which the cursor is located. To relocate the cursor on another line, click in the box and specify the line number in the Go To Line field. Clicking in this box is a shortcut for the Go To Line option from the Search menu.

Debug Windows

If the debugger is turned on (see “Use Debugger” on page 129) when IDL encounters a program error or breakpoint, the IDL Editor window containing the routine in question is brought to the front. If the file containing the error is not already open, a new Editor window is opened to contain it. A current-line
indicator is placed at the line at which the breakpoint or error occurred. You can use standard Macintosh editing commands to edit and save the program file.

**IDL Graphics Windows**
IDL graphics windows appear when you use IDL to plot or display an image.
You can copy the contents of a Graphics window—Direct or Object—directly to the operating system clipboard in a bitmap format using Command-C.

**The Menus**
Six menus (File, Edit, Search, Run, Macros, Window, and Help) allow you to control the operation of IDL for Macintosh. These menus are described below. Note that many menu items have “command-key equivalents” displayed to the right of the menu option.

**File Menu**

*New*
Select this option to create a new, empty IDL Editor window.

*Open*
Select this option to open a text file for editing. The standard file selection dialog box appears. Select the file you want to open and click OK. A new IDL Editor window is created to display the text file.

*Open Selection*
Select this option to use whatever text is selected as an argument to the Open command. If the selected text is not the name of a file in the current folder or a valid path, no file is opened.

*Close / Hide*
Select this option to close the currently-selected IDL window. If you have made changes in the window, you are asked if you want to save the changes before closing it. If the currently-selected window is the Output Log, this options changes to “Hide Output Log.” If the currently-selected window is the Output Log, this options changes to “Hide Variable Watch Window.” You can re-display a hidden window by selecting “Output Log” or “Variable Watch” from the Window menu.
Save
Select this option to save the contents of an IDL Editor window. If the window is untitled, you are prompted for a filename for the new file. If the window is already associated with a filename, the contents of the window are saved over the old file.

Note Changes made to a previously-compiled program or function are not noticed by the IDL session until that file is re-compiled. Calling the routine simply re-runs the currently-compiled version. Select the “Save and Compile” menu option, described below, to re-compile the routine.

Save As
Select this option to save the contents of an IDL Editor window to a specified filename. A file selection dialog box appears.

Revert to Saved
Select this option to discard any changes made in the current window and restore the last saved version of the file.

Page Setup
Select this option to define page orientation and other print characteristics for the currently-selected window.

Print
Select this option to print the contents of the currently-selected IDL window, text widget, or graphics widget to the currently-active printer.

Recent Files
Select this option to view or open recently opened files. This menu item lists the last ten opened files. To open a file on this list, select it.

Preferences
Select this menu item to display a cascading menu of preference options: General, Graphics, Edit, Startup, Path, and Syntax Coloring. See “Customizing IDL” on page 128 for more information.

Working Folder
Select this option to modify the current folder for reading and writing files. Note that the current folder is searched first when IDL looks for program files.

Quit
Select this option to exit IDL for Macintosh. All IDL Editor windows are closed before exiting. If text in an Editor window has changed, you are asked if you want to save it before exiting.
Edit Menu

Undo
Select this option to undo the most recent editing action. If the most recent action is not undo-able, this option will be shown as “Can’t Undo”.

Cut
Select this option to cut the currently-selected text from an IDL Editor window or the IDL command input line and place it on the clipboard.

Copy
Select this option to copy the currently-selected text in an IDL Editor window, Output Log window, or command input line to the clipboard. “Copy” also allows you to copy graphics from an IDL graphics window or draw widget to the clipboard.

Paste
Select this option to paste the contents of the clipboard at the current insertion point. You can paste into the IDL command line and IDL Editor windows, but not the Output Log window.

Clear
Select this option to delete the currently-selected text. The deleted text is not placed on the clipboard.

Select All
Use this option to select all of the text in an IDL Editor or Output Log window. The entire contents of the window are highlighted.

Shift Left
Select this menu item to shift the selected text one tab stop to the left.

Shift Right
Select this menu item to shift the selected text one tab stop to the right.

Search Menu

Find
Select this menu item to search for a text string in the currently-selected IDL Editor window, Output Log window, or text widget.

Find Again
Select this option to repeat the most recent text search.
Chapter 5: The IDL for Macintosh Interface

Find Selection
Select this menu item to search for occurrences of the currently-selected text in the currently-selected window.

Enter Selection
Select this menu item to enter the currently-selected text into the Find dialog as the search string. For example, you can enter the selection and then select “Find Again” to find the next occurrence.

Replace
Select this menu item to search for a text string in the currently-selected IDL Editor window or text widget and replace it with another text string you specify.

Replace & Find Again
Select this option to repeat the most recent search and replace operation.

Go To Routine Definition
Select this menu item to find and display the definition of the selected IDL library or user-written routine. The routine must already be compiled.

Go To Line
Select this option to specify a line on which to locate the cursor in the currently-selected IDL Editor window or text widget.

You can also use the line number button box at the bottom left of an IDL Editor window, which displays the number of the line on which the cursor is located. To relocate the cursor on another line, click in the box and specify the line number in the Go To Line field.

Run Menu
Run Menu items are enabled when an IDL program is loaded into an IDL Editor window and compiled. See Chapter 6, “Programming Tools” for more detailed information.

Note You must have the “Use Debugger” option in the IDL General Preferences dialog checked for the Debug menu to appear.

Compile
Select this option to compile the current editor window from memory. 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 Input Line, with the appropriate Editor window selected. See “.COMPILE” on page 32 of the IDL Reference Guide for a more detailed explanation.
Compile from Memory
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 Input Line. See "COMPILE" on page 32 of the IDL Reference Guide for a more detailed explanation.

Compile All
Select this option to compile all currently open *.pro files.

Run
Select this option to execute the file contained in the currently-active editor window. Selecting this option is the same as entering the procedure name at the Command Input Line.

Resolve Dependencies
Select this option to iteratively compile all uncompiled 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 Input Line. The QUIET keyword suppresses informational messages. See "RESOLVE_ALL" on page 952 of the IDL Reference Guide for a more detailed explanation.

Profile...
Select this option to start the IDL Code Profiler, which helps you analyze the performance of your applications. For more information about the Profiler, see "The IDL Code Profiler" on page 141.

Continue
Select this option to continue a stopped program or start a main-level program from the beginning. Selecting this option is the same as entering .CONTINUE at the Command Input Line. See ".CONTINUE" on page 33 of the IDL Reference Guide for a more detailed explanation.

Break
Select this option to interrupt program execution. IDL inserts a marker to the left of the line at which program execution was interrupted.

Reset
Select this option to reset the IDL environment. Selecting this item is the same as entering the following at the Command Input Line:

RETALL
WIDGET_CONTROL, /RESET
CLOSE, /ALL
HEAP_GC, /VERBOSE


**Step Over**
Select this option to execute a single statement in the current program. The current-line indicator advances one statement. If the statement “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 Input Line. See “.STEPOVER” on page 36 of the IDL Reference Guide for a more detailed explanation.

**Step Into**
Select this option to execute a single statement in the current program. The current-line indicator advances one statement. If the statement “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 Input Line. See “.STEP” on page 35 of the IDL Reference Guide for a more detailed explanation.

**Step Out**
Select this option to continue processing until the current program returns. Selecting this item is the same as entering `.OUT` at the IDL Command Input Line. See “.OUT” on page 33 of the IDL Reference Guide for a more detailed explanation.

**Trace**
Select this option to point to program lines as they are executed. Selecting this item is the same as entering `.TRACE` at the IDL command prompt. See “.TRACE” on page 36 of the IDL Reference Guide for a more detailed explanation.

**Run to Cursor**
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” on page 229 of the IDL Reference Guide for a more detailed explanation.

**Run to Return**
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” on page 33 of the IDL Reference Guide for a more detailed explanation.

**Set Breakpoint**
Select this option to set a breakpoint on the current line. Selecting this item is the same as entering the following at the IDL Command Input Line:

```idl
BREAKPOINT, ['file',] line
```

where “file” is the file to set a breakpoint within, and _index_ designates the line number at which the breakpoint is set.

See “BREAKPOINT” on page 229 of the IDL Reference Guide for a more detailed explanation.

**Set Complex Breakpoint ...**
Select this option to access the Complex Breakpoint dialog to set a complex breakpoint. Complex breakpoints may function only once, or may function only after being “hit” a specified number of times. Selecting this item is the same as setting the AFTER and ONCE keywords for the BREAKPOINT procedure at the IDL Command Input Line.

Enter the source file in which to set a breakpoint in the “File:” field. The default file is the one in which the cursor is positioned. Click “Choose File ...” to search through available directories. Enter the line number at which to place the breakpoint in the “Line:” field. The default is the line at which the cursor is currently positioned. You can also specify how many times the line must be hit in order to interrupt execution. Click “One-Time Breakpoint” to interrupt execution after encountering the line for the first time or click “Break After:” and enter the number of hits after which execution should be interrupted into the given field.

See “BREAKPOINT” on page 229 of the IDL Reference Guide for a more detailed explanation.

**Clear All Breakpoints**
Select this option to clear all breakpoints.

**List Breakpoints**
Select this option to list all breakpoints currently set, in all compiled programs. Selecting this item is the same as entering HELP, /BREAKPOINTS at the IDL Command Input Line.

**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 Input Line. See “HELP” on page 537 of the IDL Reference Guide for a more detailed explanation.

**Macros Menu**

The following macros are installed by default.

- **Print**
  Select this option to print the selected variable. Selecting this item is the same as entering `PRINT, x` at the IDL Command Input Line, where `x` is the selected variable.

- **Help**
  Select this option to access help on the selected text.

**Window Menu**

- **Stagger**
  Select this option to stagger all the Output Log and IDL Editor windows on the desktop.

- **Tile**
  Select this option to tile all the Output Log and IDL Editor windows side-by-side on the desktop.

- **Command Input Anchored**
  Select this option to anchor or unanchor the Command Input window. If the window is anchored, a check will appear next to this menu item.

- **Command Input**
  Select this menu item to give the Command Input window the input focus.

- **Output Log**
  Select this menu item to bring the Output Log to the front.

- **Variable Watch**
  Select this option to give the Variable Watch window the input focus.

- **Profile**
  Select this option to bring the Profile dialog to the front. For more information about the Profiler, see “The IDL Code Profiler” on page 141.
Profile Results
Select this option to bring the Profile Report dialog to the front. For more information about the Profiler, see “The IDL Code Profiler” on page 141.

Open Editor Windows
If any files are open, they are listed at the bottom of the Window menu. Select any of these menu items to make that window the current window and give it the input focus.

Help Menu
The Help menu is located at the far right of the menu bar.

About Balloon Help . . .
IDL for Macintosh supports balloon help. Select this menu item to learn more about balloon help.

Show Balloons
Select “Show Balloons” to activate help balloons. This menu item changes to “Hide Balloons” when balloons are enabled.

IDL Online Help
Select this menu item to display the IDL Online Help Viewer.

Help on Selection
Select this menu item to display the help topic in IDL Online Help for the highlighted item. The Index dialog appears if the topic is ambiguous or does not exist.

Customizing IDL
Various defaults for IDL can be customized using the IDL Preferences dialog box. Select “Preferences” from the IDL File menu to display a cascading list of preferences.

Select an item from the list to display a preferences dialog box. Enter new values and click on “OK” to use the preferences in the current IDL session. Select the “Save Settings on Exit” option (a checkmark appears by the item the next time you select “Preferences”) to save the preferences for use with future IDL sessions.

General Preferences
These preferences control the general appearance and behavior of IDL.
Lines to Save in Log Window
Enter the number of lines you wish to save in the IDL Output Log window. By default, 200 lines are saved.

Anchor Command Window
Use this option to select whether you want the IDL Command Input window to be anchored at the top or bottom of your screen when it is anchored. You can unanchor the Command Input window by unchecking “Command Input Anchored” in the Window menu.

Default Text Formats
These three buttons allow you to set the default font, font size, and tab size to be used in:
- IDL Editor windows,
- Text and List widgets,
- Buttons, menus, titles, and other widget objects.

Note To set the text formats for the current window only, select “Format” from the Edit Preferences option.

Use Debugger
If this box is checked, the Debug menu appears in the menu bar and the IDL for Macintosh debugger automatically opens an edit window containing the program module in question when an error occurs in an IDL program.
Graphics Window Settings

These preferences control defaults for IDL graphics windows.

Number of Colors Used
The number of colors allocated for IDL graphics windows. The default is 220. If your display supports 256 colors, a maximum of 254 colors can be reserved for IDL (black and white are reserved for the System).

Default Window Width
The width of IDL graphics windows, in pixels. The default is 1/4 of the total screen width.

Default Window Height
The height of IDL graphics windows, in pixels. The default is 1/4 of the total screen height.

Backing Store
This field controls the default for how IDL handles backing store. When backing store is enabled, IDL keeps a copy of each window in memory. This data is used to refresh the window when it has been covered and uncovered.

Change this field to “None” to disable backing store. Set this field to “Pixmap” (the default) to have IDL maintain backing store.

See “Backing Store” on page 144 of the IDL Reference Guide for more information.

Startup Depth
This popup menu allows you to specify the color pixel depth you wish to work in, regardless of the actual depth of your monitor. All operations (saving, printing, copying to the clipboard, etc.) will be carried out in the depth you select, even if your monitor does not support that depth.

For example, if you wish to work with 24-bit images but your computer only supports 8-bit video, select “24-bit” from the Startup Depth menu. (If you do not select “Dither to Lower Depth Screens,” images will not be displayed properly on your monitor.) Similarly, if you wish to use 8-bit, pseudo-color video even though your machine supports 24-bit true-color, choose “8-bit.” Select “Screen Depth” to match the pixel depth IDL works with to your monitor.

Note  This setting takes effect when IDL is restarted.

Dither to Lower Depth Screens
Check this box to display high pixel-depth images on a lower pixel-depth monitor. Floyd-Steinberg dithering is used to display, for example, true-color images on an 8-bit monitor.
Using IDL

Customizing IDL

Chapter 5: The IDL for Macintosh Interface

images on an 8-bit-deep screen. This setting takes effect immediately when you click “OK”—you need not close and restart IDL.

Size of TrueType Font Cache (in Glyphs)
Enter the number of TrueType characters to save triangulation information for. 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.

Object Graphics Renderer
Select either Hardware Rending (Open GL) or Software Rendering. See “Window Objects” in Chapter 14 (“Using Destination Objects”) of Objects and Object Graphics for information about the differences between the two rendering systems.

Hardware Font
Click on this button to bring up the Graphics Hardware Font dialog, which allows you to specify the font, font size, and style to be used when hardware fonts are specified for use in IDL graphics windows.

Note The !P.FONT system variable field must be set equal to zero to use hardware fonts rather than the default vector fonts.

Edit Preferences
These preferences control the look of the IDL windows.

Window Format
The current window—Active Window, Command Input, or Output Log—is reflected in this dialog. You can set the font, the font size, and the tab size. The area to the right of the pulldown menus shows an example of the settings.
AutoIndent
Check this box to activate auto-indentation as applicable in the current window.

Startup Settings
These preferences control the location of the IDL Home Folder and the IDL Startup File.

Select IDL Main Dir...
The IDL Main Dir is the folder in which IDL was installed. It only needs to be changed if the executable is moved somewhere else (e.g., to a special applications folder). Clicking on this button displays a dialog for selecting the folder. The IDL main directory is displayed below.

Select Startup File
Click this button to specify the name of an IDL batch file to be executed automatically each time IDL is run. The startup file is displayed below.

Note  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.

Use No Startup File
Click this button to have no startup file executed.

Path Specifications
This dialog specifies where IDL will look for procedures and functions. To specify a folder that includes all of that folder’s subfolders, select the entry in the list and click the “Search Subfolders” button. A “+” will be shown in front of the path,
indicating that the folder is to be searched recursively. To change the path specification list, click on either “Add” or “Remove”. The default path is the IDL folder and all of its subfolders.

**Syntax Coloring**

This dialog specifies if IDL should use syntax coloring within open editor windows and in the Command Input Line. To change the color associated with a context word, click on the color box next to each type of word. To disable syntax coloring altogether, un-check the “Use Syntax Coloring” checkbox. To disable syntax coloring only within the Command Input Line, un-check the “Use Syntax Coloring on Command Line” checkbox.

**Setting IDL’s Memory Partition**

Running large IDL programs or displaying large images may require more RAM than is automatically allocated for IDL by the installation program. In general, the more memory you allocate for IDL, the faster complex programs will run. Of course, you must balance the size of IDL’s memory partition with the memory requirements of other the applications you use.

To change the memory allocation, first exit IDL. Using the Finder, click on the IDL icon. Select “Get Info” from the Macintosh system File menu to bring up the file information dialog. In the “Memory Requirements” section of the file information dialog is a field that reads “Preferred Size”. Change the value in this field to reflect the amount of memory you wish to allocate to IDL, in kilobytes.
Close the IDL Info window when you are finished. When you restart IDL, the new memory allocation will be in effect.

**Message-of-the-Day File**

A “message-of-the-day” file can be used to display the contents of an ASCII text file each time IDL is run. To create a message-of-the-day file for IDL for Macintosh, simply name the desired text file `MOTD.txt` or `MacOS.txt` and place it in the `motd` folder in the `help` folder in the main IDL folder. Note that the MOTD file is simply an ASCII text file—not an IDL program or batch file. To execute a series of IDL commands, select a startup file as described under “IDL Startup Settings,” above.

If you don’t wish to see the message-of-the-day file each time you start IDL, simply remove or rename the `MacOS.txt` or `MOTD.txt` file.

**Macintosh IDL Differences**

The Macintosh version of IDL implements most of the functionality of other IDL versions. There are a number of differences, however, as described below.

**Using the Macintosh Mouse with IDL**

IDL supports the use of mice with up to three buttons. However, the Macintosh mouse has only one button. When pressed, the Macintosh mouse button is interpreted by IDL as the “left” mouse button. Hold down the Option key while pressing the mouse button to simulate a “middle” mouse button press. Hold down the Command key while pressing the mouse button to simulate a “right” mouse button press.

**Specifying Paths**

Many IDL commands accept a partially- or fully-qualified filename (i.e., a filename and the directory path to that file) as an argument. However, the Macintosh uses a graphical representation of file folders instead of a directory tree. To solve this problem, the following syntax is used to specify the location of files and folders:

- Partially- or fully-qualified filenames are specified as a colon-separated list of drive names and folders.
- Folder and file names can contain spaces and/or commas.

For example, the string to specify the fully-qualified filename of the file `myprogram.pro`, located in the folder named `Programs` which resides on the drive named `Macintosh HD` would be:
Partially-qualified filenames—filenames specified relative to the current directory—begin with ":" (the colon character). For example, to specify the filename of `myprogram.pro`, located in the folder `test`, which is located in the current working folder (whatever that may be), use the following string:

`':test:myprogram.pro'`

### Operating System Commands

**Changing the Current Working Directory**

To change the current working directory, specify a valid Macintosh path to a folder (as described above under “Specifying Paths”) with the IDL CD command. For example, to change the current directory (folder) to the folder `Programs`, which resides on a disk called `Macintosh HD`, enter the command:

`IDL> CD, 'Macintosh HD:Programs:'`

Other Macintosh operations can change the current directory as well:

- Opening a file with the “Open” command or by double-clicking on it in the Finder changes the current directory to the folder where that file resides.
- Saving a file with the “Save As” command changes the current directory to the folder where the saved file resides.
- Choosing a new folder using the “Working Folder” command in the File menu.

**Note** The IDL routine `DIALOG_PICKFILE` does not change the current directory.

### File Manipulation

**Compiling Programs**

Because Macintosh filenames allow spaces, the `.RUN`, `.RNEW`, and `.COMPILE` executive commands cannot be used to compile multiple programs with a single command.

For example, on most IDL platforms, the following line compiles three IDL program files named `test`, `demo`, and `program`:

`IDL> .RUN test demo program`

However, since Macintosh filenames can have spaces in them, the filenames shown above would be interpreted as a single filename. In IDL for Macintosh, you can only specify one filename per `.RUN` command. For example:

`IDL> .RUN Macintosh HD:Programs:test`
Save/Restore Files
SAVE/RESTORE files generated with the Macintosh version of IDL are saved in the XDR format. This format allows data files saved under UNIX, VMS, Macintosh and DOS systems to be easily exchanged.

Logical Unit Numbers
The three special file units, 0, -1, and -2, are tied to stdin (the command line), stdout (the output log) and stderr (the output log), respectively.

Positioning File Pointers
Under Macintosh, the current file pointer cannot be positioned past the end of the file.

Math Error Handling
Integer divide by zero is always trapped. Integer overflow and underflow are not detected. Improper floating-point operations are trapped.

Macintosh-Specific File Information
When a file is saved on the Macintosh, it is assigned a “file type”. Text files saved from IDL are assigned the type TEXT. Binary files saved from IDL are assigned the type “BIN “. (Note that the type code is always four characters long, so the BIN type code includes an ASCII space character at the end.)
Chapter 6

Programming Tools

The following topics are covered in this chapter:

- Debugging in IDL ......................... 138
- Debugging Commands ................. 138
- The IDL Code Profiler ............... 141
- The Variable Watch Window .......... 145
IDL's built-in debugging and executive commands are explored in this chapter, illustrated with a short example using the IDLDE interface to debug a file.

**Debugging in IDL**

There are several tools you can use to help you find errors in your IDL code. The Run menu item in the IDL Development Environment provides several ways to access IDL's built-in debugging and executive commands. The IDL Profiler provides useful information about the routines used in the program being executed. The Variable Watch Window helps you keep track of the variables used in your program.

This chapter explains the debugging commands and contains short examples using the IDLDE interface to debug a file.

**Debugging Commands**

When a file displayed in an IDL editor window has been compiled (by selecting “Compile” or “Memory Compile” from the Run menu, or by entering `.COMPILE`, `.COMPILE -f`, or `.RUN` at the IDL command prompt), a number of debugging commands become available for selection. For the IDLDE for Macintosh, see “Run Menu” on page 123. For the IDLDE for Motif, see “Run Menu” on page 48. For the IDLDE for Windows, see “Run Menu” on page 93.

When execution is interrupted, a current-line indicator is placed next to the line that will be executed when processing resumes. The routine being compiled need not already be shown in an editor window. If a routine compiled with the `.RUN`, `.RNEW`, or `.COMPILE` executive commands contains an error, IDLDE will display the file automatically.

**A Simple Example**

A simple procedure, called BROKEN, has been included in the IDL distribution. An error occurs when BROKEN is executed.

Start the IDLDE. Call the BROKEN procedure by entering:

```
BROKEN
```

at the IDL command line. An error is reported in the Output Log window and an editor window containing the file `BROKEN.PRO` appears.

A “Variable is undefined” error has occurred. Since execution stopped at line 4, that line is highlighted with an arrow.

Click on the Output Log window to see the error:
% Compiled module: BROKEN.
% PRINT: Variable is undefined: I.
% Execution halted at BROKEN 4
    /user/local/rsi/idl50/examples/general/broken.pro
% $MAIN$

There are several ways of fixing this error. We could edit the program file to
explicitly define the variable i, or we could change the program so that it accepts
a parameter at the command line. We can also define the variable i “on the fly”
and continue execution of the program without making any changes to the
program file. We'll do this first, then go back and edit the program to accept a
command-line parameter.

To define the variable i and assign it the value 10, click in the IDL command line
and enter:

    i = 10

**Step Through the Program**

Select “Step Into” from the Run menu to execute line 4 with the new value of i
and step to the next program line.

The Output Log reports:

    10

and the current-line pointer advances to the next line in the window containing
the file BROKEN.pro. You could continue stepping through the program by
choosing “Step Into” repeatedly (or by entering .STEP at the IDL command
prompt).

The Trace dialog offers an opportunity to automatically step through the
program. Select “Trace...” from the Run menu. The Trace dialog appears. Click
“Run” to continue issuing the .STEP command until the END statement is
encountered. Click “Cancel” to dismiss the Trace dialog.

You can also continue execution of the program without stepping through. Select
“Run” from the Run menu, noting that the Output Log shows that IDL calls
broken. Define the variable i in the Command Input Line. Select “Run” again.
The Output Log now shows that IDL calls .CONTINUE. IDL prints the resulting
output to the Output Log window:

    10
    20
    30
    40
When stepping through a main program, if the next line calls another IDL procedure or function, you have three options with which to handle execution of the nested program. Selecting “Step Into” executes statements in order by successive “Step” commands. Selecting “Step Over” executes statements to the end of the called function, without interactive capability. Select “Step Out” if you would like to continue processing until the main program returns.

Fix the Program

To fix the program permanently, edit the first line of the program to be:

```
PRO BROKEN, i
```

Select “Save” from the File menu and “Compile” from the Run menu. IDL saves the modified text file over the old version and compiles the modified routine. To call this new version of BROKEN with an input argument of 10, enter:

```
BROKEN, 10
```

The Output Log window prints the result:

```
10
20
30
40
```

Breakpoints

You can suspend execution of a program temporarily by setting breakpoints in the code. Set a breakpoint at the fifth line of BROKEN.PRO by placing the cursor in the line that reads:

```
PRINT, i*2
```

and selecting “Set Breakpoint” from the Run menu. A breakpoint dot appears next to the line. Now enter:

```
BROKEN, 10
```

The Output Log window displays the following:

```
10
% Breakpoint at: BROKEN              5
    user/local/rsi/idl40/examples/general/broken.pro
```

and a current-line indicator marks line 5. Select “Run” to allow execution to resume. To list the breakpoints, select “List Breakpoints” from the Run menu. This is the same as entering `HELP,/BREAKPOINT` at the Command Input Line. You can remove a breakpoint by selecting “Clear All Breakpoints” from the Run menu.
Setting a breakpoint allows you to inspect (or change) variable definitions as the program executes. Since our example does not set any variables, setting a breakpoint in `BROKEN.PRO` is not very informative. Breakpoints can be extremely helpful, though, when debugging complex programs, or programs that call other routines.

You can also set a one-time breakpoint either to run to where the cursor is positioned or to where a `RETURN` statement is located in the code. See “Run to Cursor” or “Run to Return” from the Run menu in your platform’s description of the IDL Development Environment for more information.

### The IDL Code Profiler

The IDL Code Profiler helps you analyze the performance of your applications. You can easily monitor the calling frequency and execution time for procedures and functions. The Profiler can be used with programs entered from the command line as well as programs run from within a file.

You can start the IDL Code Profiler by selecting “Profile” from the Run menu of the IDLDE or by entering `PROFILER` at the Command Input Line. For more information about the `PROFILER` procedure, see “PROFILER” on page 846 of the IDL Reference Guide.

**Note** Calling the Profiler from the Command Input Line does not start the Profiler dialog.

### The Profile Dialog

Select “Profile” from the Run menu. The Profile dialog appears.
User Modules
User modules include user-written procedures as well as library procedures and functions provided with IDL. By default, none of the User Modules are selected for profiling. To select a module, click on the checkbox next to it. All user modules must be compiled before opening the Profile dialog in order to be available for profiling.

All User Modules
Select this checkbox to select all the user modules for profiling.

System Modules
This field includes all IDL system procedures and functions.

All System Modules
Select this checkbox to select all the system modules for profiling.

Buttons
Click “Profile All” to enable profiling for all the available modules—System and User. Click “Clear All” to disable profiling for all the available modules—System and User. Click “Reset” to clear the report shown in the “Profile Report” dialog. The “Profile Report” dialog is dismissed, as it no longer contains any information. Click “Report” to generate a profile of the selected modules. The Profile Report dialog appears. Click “Cancel” to dismiss the Profile dialog. Click “Help” to display this chapter of Using IDL.

The Profile Report Dialog
Click “Report” from the Profile dialog in the Run menu of the IDLDE. The Profile Report dialog appears.

Fields in the Profiler Report Dialog
The fields in the Profiler Report dialog show the following attributes of the modules selected for profiling from the Profile dialog. You can sort the values in each column in both ascending and descending order by clicking anywhere within the column. By default, the Modules column is sorted alphabetically.

Note
Whether you enter a program at the command line, or run a program contained in a file, the PROFILER procedure will report the status of all the specified modules compiled and executed either since profiling was first set or since the PROFILER was reset.

Modules
The name of the library, user, or system procedure or function.
Typ
The type of module. System procedures or functions are associated with an “S”. User or library functions or procedures are associated with a “U”.

Count
The number of times the procedure or function has been called.

Only(sec)
The time required, in seconds, for IDL to execute the given function or procedure, not including any calls to other functions or procedures (children).

Only Avg
Average of the Only(sec) field above.

+Children(sec)
The time required, in seconds, for IDL to execute the given function or procedure including any calls to other functions or procedures.

+Child Avg
Average of the +Children(sec) field above.

Buttons
Click “Print” to print the report. The Print dialog appears. You can also select “Print” from the File menu of the IDLDE. Click “Save” to save the report as a text file. The Save Profile Report dialog appears. Click “Cancel” to dismiss the Profile Report dialog. The contents remain available after cancelling. Click “Help” to display this chapter of Using IDL.

Using the IDL Code Profiler
Open a new editor file by selecting “New” from the File menu.
Enter the following lines in the editor:

```
pro prof_test
openr, 1, filepath('nyny.dat', subdir=['examples', 'data'])
a=assoc(1, bytarr(768,512))
b=a[0]
close, 1
TV, b
end
```

Save the file as prof_test.pro by selecting “Save” from the File menu. The Save As dialog appears.

To use the IDL Code Profiler, you must first compile the routines you would like to profile. For more involved programs, you can use RESOLVE_ALL to compile
all uncompiled functions or procedures that are called in any already-compiled procedure or function.

Select “Profile...” from the Run menu. The Profile dialog appears; it will remain visible until dismissed. Select “Profile All” to profile all the available modules.

Run the application by selecting “Run” from the File menu. After the application is finished, return to the Profile dialog and click “Report”. The Profile Report dialog appears, as shown below.

For more information about the capabilities of either dialog, see “The Profile Dialog” on page 141 and “The Profile Report Dialog” on page 142.

Profiling with Command Line Modules

We will demonstrate how the Profiler handles newly compiled modules. The above example set profiling for all system files, plus the user module, prof_test, and the library function, FILEPATH. If you have altered the above results, reset the report and run prof_test again.

Enter the following lines at the Command Input Line:

```
A = DIST(500)  
TV, A
```

Create a dataset using the library function DIST. Note that DIST is immediately compiled.

Display the image.

Return to the Profile dialog. You will note that the DIST function has been appended to the User Module field, but that it remains unselected. The Profiler will not include any uncompiled modules by default. Click “Report” in the Profile dialog.
dialog to refresh the Profile Report dialog's results. The figure below shows the new results. Note that TV is counted twice, and that more system modules have been appended to the Modules column. The DIST function, although it is not itself included, calls system routines which were previously selected for profiling.

If you select DIST in the User Modules field in the Profile dialog and then re-enter only the statement calling TV at the Command Input Line, you will notice that only the count for TV increases in the profiler report. You must re-enter the statement calling DIST at the Command Input Line; the already-compiled library function is executed again, making it available for profiling.

The Variable Watch Window

The Variable Watch window displays current variable values after IDL has completed execution. If the calling context changes during execution — as when stepping into a procedure or function — the variable table is replaced with a table
appropriate to the new context. While IDL is at the main program level, the Watch window remains active and displays any variables created.

**Customizing Variable Watch Window Layout**

To hide the Variable Watch window, select “Hide Variable Watch” from the Configure option in the Window menu. Select “Show Variable Watch” to make it reappear. Changing the Window menu will only affect the current IDL session. To apply your changes to future sessions, go to the Layout tab from the Preferences option of the File menu. In the section labeled Windows, you can use the Hide field to make any of the available options disappear. Click “Save” to apply any changes to future IDL sessions.

**Note** The Configure option from the Window menu reflects changes in the Layout Preferences and vice versa.

You can also choose to separate the Variable Watch window from the main IDLDE window. Use the Separate field in the Layout Preferences tab from the File menu.

**The Variable Watch Interface Description**

The Variable Watch window is refreshed after the IDLDE has completed execution. Each Variable Watch window contains the following folders:

- **Locals**
  This tab contains descriptions of local variables. Local variables are created from IDL's main program level. For example, entering `a=1` at the Command Input Line lists the integer `a` in the Locals tab.

- **Params**
  This tab contains descriptions of parameters. The variables and expressions passed to a function or procedure are parameters. For more information, see “Parameters” on page 110 of Building IDL Applications.

- **Commons**
  This tab contains descriptions of variables contained in common blocks. The name of each common block is shown in parentheses next to the variable contained within it. For more information, see “Common Blocks” on page 93 of Building IDL Applications.

- **System**
  This tab contains descriptions of system variables. System variables are a special class of predefined variables available to all program units. For more
information about system variables, see “System Variables” in the IDL Reference Guide

Each tab contains a table listing the variables included in the category. You can size the columns by clicking on the line to the right of the title of the column you wish to expand or shrink. Drag the mouse either left or right until you are satisfied with the width of the column. For example, to change the width of the Name column, click and drag on the line separating the Name field from the Type field.

The following fields describe variable attributes:

- **Name**
  This field shows the name of the variable. This field is read-only, except for array subscript descriptions (see example in Using the Variable Watch Window below).
  
  For compound variables such as arrays, structures, pointers, and objects, click the “+” symbol to the left of the name to show the variables included in the compound variable. Click the “-” symbol to collapse the description.

- **Type**
  This field shows the type of the variable. This field is read-only.

- **Value**
  This field shows the value of the variable. To edit a value, highlight the cell by clicking on it, press the function key $F2$ to enter editing mode, and enter the new value.

The Name, Type, and Value fields are displayed as when using the procedure “HELP” on page 537 of the IDL Reference Guide. For more information about variables, see “Variables” in Chapter 2 of Building IDL Applications.

**Note** The Variable Watch Window and Objects

Object references are expanded only if they reference non-null objects. Object data are expanded only if the object method has finished running. Object data are read-only and cannot be changed with the Variable Watch window.

**Using the Variable Watch Window**

Arrays are expanded to show one array element. Click on the “+” symbol next to the name of the array to display the initial array subscript. You can change this field to display the characteristics of any other array element. To edit the subscript, highlight the cell by clicking on it, press the function key $F2$ to enter editing.
mode, and modify the name using the arrow keys to maneuver. For example,
enter the following:

```idl
d=MAKE_ARRAY(2,3)  # Create an array with 2 columns
                  # and 3 rows.
PRINT, A          # Show the values of array A in the
                  # Output Log. They will all be zero.
A(2)=5            # Assign the value of 5 to the value in
                  # the array subscripted as 2. This is
                  # the same as entering A(0,1)=5.
PRINT, A          # Show the new values of array A.
```

IDL prints:

```
0.00000      0.00000
5.00000      0.00000
0.00000      0.00000
```

It is easy to manipulate variables within the Watch window. Click on the “+”
expansion bitmap next to the array A. The subscript [0,0] will be revealed
beneath the description of A. Enter editing mode and change [0,0] to [0,1].

**Note** To enter editing mode in Motif, press F2 after clicking on the cell to be edited.
In Windows, double click on the cell. On the Macintosh, click on the cell.

Press Return [Enter] to effect the change. Notice that the value of the subscript is
displayed as 5.00000, as you entered from the Command Input Line. Press the
“Tab” key twice to highlight the value of the subscript [0,1]. You can change it
to another number. Enter [1,0] in the subscript name field. You can change the
value from 0.00000 to another number.

For more information about arrays, see “Arrays” in Chapter 9 of Using IDL.
Chapter 7
Using the IDL GUIBuilder

The following topics are covered in this chapter:

- Starting the IDL GUIBuilder ........ page 151
- Creating an Example Application page 153
- IDL GUIBuilder Tools ................ page 164
- Widget Operations.................. page 176
- Generating Files.................... page 179
- IDL GUIBuilder Examples .......... page 181
- Widget Properties................... page 196
- Common Widget Properties....... page 196
- Base Widget Properties .......... page 202
- Button Widget Properties ......... page 212
- Text Widget Properties .......... page 216
- Label Widget Properties .......... page 221
- Slider Widget Properties .......... page 222
- Droplist Widget Properties ...... page 224
- Listbox Widget Properties ....... page 226
- Draw Widget Properties .......... page 229
- Table Widget Properties .......... page 234
The IDL GUIBuilder is part of the IDLDE for Windows. The IDL GUIBuilder supplies you with a way to interactively create user interfaces and then generate the IDL source code that defines that interface and contains the event-handling routine place holders.

**Note** The IDL GUIBuilder is supported on Windows only. However, the code it generates is portable and runs on all IDL 5.2 supported platforms. Since applications built with IDL GUIBuilder require functionality added to IDL 5.2, generated code only runs on IDL 5.2 or greater.

The IDL GUIBuilder has several tools that simplify application development. These tools allow you to create the widgets that make up user interfaces, define the behavior of those widgets, define menus, and create and edit color bitmaps for use in buttons.

**Note** When using code generated by the IDL GUIBuilder on other non-Windows platforms, more consistent results are obtained by using a row or column layout for your bases instead of a bulletin board layout. By using a row or column layout, problems caused by differences in the default spacing and decorations (e.g., beveling) of widgets on each platform can be avoided.

These are the basic steps you will follow when building an application interface using the IDL GUIBuilder:

1. Interactively design and create a user interface using the components, or widgets, supplied in the IDL GUIBuilder. Widgets are simple graphical objects supported by IDL, such as sliders or buttons.

2. Set attribute properties for each widget. The attributes control the display, initial state, and behavior of the widget.

3. Set event properties for each widget. Each widget has a set of events to which it can respond. When you design and create an application, it is up to you to decide if and how a widget will respond to the events it can generate. The first step to having a widget respond to an event is to supply an event procedure name for that event.

4. Save the interface design to an IDL resource file, *.prc file, and generate the portable IDL source code files. There are two types of generated IDL source code: widget definition code (*.pro files) and event-handling code (*.eventcb.pro files).

5. Modify the generated *.eventcb.pro event-handling code file using the IDLDE, then compile and run the code. This code can run on any IDL-supported platform.
The *_eventcb.pro file contains place holders for all of the event procedures you defined for the widgets, and you complete the file by filling in the necessary event callback routines for each procedure.

**Caution**  Once you have generated the widget definition code (*.pro files) and event-handling code (*_eventcb.pro files), you should not modify these files manually. If you decide to change your interface definition or your event handlers, you will need to regenerate the interface code or the event-handling code, and will therefore overwrite the *.pro files. If you do wish to make manual changes to these files, save them using a different filename, and paste the changes into the automatically-generated *.pro files.

“Widgets” in Chapter 15 of Building IDL Applications contains complete information about IDL widgets, and it describes how to create user interfaces programmatically (without the IDL GUIBuilder).

**Starting the IDL GUIBuilder**

To open a new IDL GUIBuilder window:

1. From the IDLDE File menu, choose New, then choose GUI.
2. Or
   - Click the New GUI button on the IDLDE toolbar.

Each of these actions opens a new IDL GUIBuilder window and displays the IDL GUIBuilder toolbar. The IDL GUIBuilder window contains a top-level base widget, as shown in Figure 7-1. This top-level base holds all of the widgets for an individual interface; it is the top-level parent in the widget hierarchy being created.

**Opening Existing Interface Definitions**

To open an existing interface design in the IDL GUIBuilder:

1. Do one of the following to launch the Open dialog:
   - From the IDLDE File menu, choose Open.
   - Or
     - Click on the Open button on the IDLDE toolbar.
2. In the Open dialog, locate and select the appropriate *.prc file, and click Open.

The *.prc portable resource file contains the widget definitions that make up the widget hierarchy and define your interface design. When you click Open, the
existing definition is displayed in a IDL GUIBuilder window. You can modify the interface then save it, and you can generate new IDL source code for the modified definition.

**Note** The next section, Creating an Example Application, describes how to build a simple application using the IDL GUIBuilder. The other sections in this chapter describe the basic functionality and usage of the IDL GUIBuilder and the properties you can set for each widget.

![Figure 7-1: IDLDE with New IDL GUIBuilder Window](image)
Creating an Example Application

The following example takes you through the process of creating your first application with the IDL GUIBuilder and the IDLDE. You will create the user interface and write the event callback routines.

The simple example application contains a menu and a draw widget. When complete, the running application allows the user to open and display a graphics file in GIF format, change the color table for the image display, and perform smooth operations on the displayed image.

This example introduces you to some of the basic procedures you will use to create applications with the IDL GUIBuilder; it shows you how to define menus, create widgets, set widget properties, and write IDL code to handle events.

Defining Menus for the Top-level Base

To define the menu, follow these steps:

1. Open a new IDL GUIBuilder window by clicking on the New GUI button (window icon) on the IDLDE toolbar.

2. Drag out the window then the top-level base to a reasonable size for displaying an image.

   For example, the base in Figure 7-3 has an X Size property value of 500 and a Y Size property value of 400. To view the property values, right-click on the base, and choose Properties from menu. In the Properties dialog, scroll down to view the X Size and Y Size property values.

3. Right-click on the top-level base in the IDL GUIBuilder window, then choose Edit Menu. This action opens the Menu Editor, as shown in Figure 7-2.

4. In the Menu Editor Menu Caption field, enter “File” and click Insert. Clicking Insert sets the entered value and adds a new line after the currently selected line, and the new line becomes the selected line.

5. To define the File menu items, do the following:
   - With the new line selected, click on the right arrow in the Menu Editor, which indents the line and makes it a menu item.
   - Click in the Menu Caption field and enter “Open...”.
   - Click in the Event Procedure field and enter “OpenFile”. The OpenFile routine will be called when the user selects this menu.
   - To create a separator after the Open menu, click the line button at the right side of the dialog (above the arrow buttons).
Creating an Example Application

6. To define the Tools menu and its one item, do the following:
   • With the new line selected, click the left arrow to make the line a top-level menu.
   • In the Menu Caption field, enter “Tools”, then click Insert.
   • Click the right arrow to make the new line a menu item.
   • In the Menu Caption field, enter “Load Color Table”.
   • In the Event Procedure field, enter “OnColor”.
   • To set the values and move to a new line, click Insert.

7. To define the Analyze menu and its one menu item, do the following:
   • With the new line selected, click the left arrow to make the line a top-level menu.
   • In the Menu Caption field, type “Analyze”, then press Enter.
   • Click the right arrow to make the new line a menu item.
   • In the Menu Caption field, enter “Smooth”.
   • In the Event Procedure field, enter “DoSmooth”.
   Your entries should look like those shown in Figure 7-2.

8. Save your menu definitions by clicking OK in the Menu Editor.

   Note For more information about using the Menu Editor, see “Using the Menu Editor” on page 171.
9. At this time you can click on the menus to test them. Your interface should look similar to the one in Figure 7-3.

10. From the IDLDE File menu, choose Save, which opens the Save As dialog.

11. In the Save As dialog, select a location, enter “example.prc” in the File name field, and click Save. This action writes the portable resource code to the specified file.
Creating a Draw Widget

To create a draw area that will display GIF image files, follow these steps:

1. Click on the Draw Widget tool button (the dark square icon), then drag out an area that fills the top-level base display area. Leave a small margin around the edge of the draw area when you drag it out.

For more information about creating and operating on widgets, see “Using the IDL GUIBuilder Toolbar” on page 164, and see “Widget Operations” on page 176.
2. Right click on the draw area, and choose Properties. This action opens the Properties dialog for the draw area; the draw widget properties are displayed in the dialog.

3. In the Properties dialog, click the push pin button so the dialog will stay open and on top.

4. In the Properties dialog, change the draw widget Name attribute value to “Draw”. Later, you will write code to handle the display of the image in this draw area widget. Renaming the widget now will make it easier to write the code later; the “Draw” name is easy to remember and to type.

   Note: The Name property must be unique to the widget hierarchy.

5. In the IDL GUIBuilder window, click on the top-level base widget to select it. When you do so, the Properties dialog will update and display the attributes for this base widget.

6. In the Properties dialog, locate the Component Sizing property, and select Default from the droplist values. This action sizes the base to the draw widget size you created. The application should look like the one shown in Figure 7-4.

   When you first dragged out the size of the base, the Component Sizing property changed from Default to Explicit—you explicitly sized the widget. Now that the base widget contains items, you can return it to Default sizing, and IDL will handle the sizing of this top-level base.

7. From the File menu, choose Save, which saves your new modifications to the example.prc file.
You can run the application in test mode, which allows you to test the display of widgets and menus.

To run your application in test mode:

- From the Run menu, choose Test GUI.
- Or
- Press Control+t.

Both these actions display the interface as it will look when it runs. You can click on the menus, but there is no active event handling in test mode.
To exit test mode:

Press the Esc key.

Or

Click the close X in the upper-right corner of the test application window.

**Generating the IDL Code**

To generate the code for the example application, follow these steps:

1. From the File menu, choose Generate .pro, then choose Both. This action opens the Save As dialog.

2. In the Save As dialog, find the location where you want the files saved, enter “example.pro” in the File name field, and click Save.

This action generates an example.pro widget definition file and an example_eventcb.pro event-handling file.

The example.pro file contains the widget definition code, and you should never modify this file. If you decide later to change your interface, you will need to regenerate this interface code, and thus overwrite the widget code file.

The example_eventcb.pro contains place holders for all the event procedures you defined in the IDL GUIBuilder Menu Editor and Properties dialog. You must complete these event procedures by filling in event callback routines. If you generate code after you have modified this file, be careful to ensure that you do not overwrite the file. For information on ways to handle regenerating the *.eventcb.pro file, see “Notes on Generating Code a Second Time” on page 180.

For more information on interface definitions and generated code, see “Generating Files” on page 179.

**Note** You should modify only the generated event-handling file (*_eventcb.pro); you should never modify the generated interface code (the *.pro file).

**Handling the Open File Event**

You can now modify the generated example_eventcb.pro file to handle the events for the application. First, you will modify the OpenFile routine.

When the user selects Open from the File menu of the example application, the appropriate event structure is sent, and the OpenFile routine handles the event. For this application, the Open menu item will launch an Open dialog to allow the user to choose a GIF file, and then the routine will check the selected file’s type, read the image, and display it in the draw area.
To open the file and add the code to handle the OpenFile event, follow these steps:

1. From the File menu in the IDLDE, choose Open, which launches the Open dialog.

2. In the Open dialog, locate and select the example_eventcb.pro file, and click Open. This file contains the event handling routine placeholders, which you will now complete.

3. In the example_eventcb.pro file, locate the OpenFile routine calling sequence, which looks like this:

   ```idl
   PRO OpenFile, Event
   END
   ```

4. Add the following code to handle the event (the comments describe the added code):

   ```idl
   PRO OpenFile, Event
   
   ; If there is a file, draw it to the draw widget.
   sFile = Dialog_Pickfile(filter="*.gif")
   if(sFile ne ")then begin
   
   ; Find the draw widget, which is named Draw.
   wDraw = Widget_Info(Event.top, find_by_uname="Draw");
   ; Make sure something was found.
   if(wDraw gt 0)then begin
   ; Make the draw widget the current, active window.
   widget_control, wDraw, get_value=idDraw
   wset,idDraw
   
   ; Read in the image.
   read_gif,sFile, im
   ; Size the image to fill the draw area.
   im = congrid(im, !d.x_size, !d.y_size)
   ; Display the image.
   tv, im
   ; Save the image in the uvalue of the top-level base.
   widget_control, Event.top, set_uvalue=im, /no_copy
   endif
   endif
   END
   ```

   **Note**  In the added code, you used the FIND_BY_UNAME keyword to find the draw widget using its name property. In this example, the widget name, "Draw", is the one you gave the widget in the IDL GUIBuilder Properties dialog. The widget name is case-sensitive.
Handling the Exit Event
To add the code that causes the example application to close when the user chooses Exit from the File menu, follow these steps:
1. Locate the OnExit routine place holder, which looks like this:
   ```idl```
   PRO OnExit, Event
   END
   ```idl```
2. Add the following statement to handle the destruction of the application:
   ```idl```
   PRO OnExit, Event
   widget_control, Event.top, /destroy
   END
   ```idl```

Handling the Load Color Table Event
To add the code that causes the example application to open the IDL color table dialog when the user chooses Load Color Table from the Tools menu, follow these steps:
1. Locate the OnColor routine place holder, which looks like this:
   ```idl```
   PRO OnColor, Event
   END
   ```idl```
2. Add the following procedure to open the IDL XLoadct color table dialog:
   ```idl```
   PRO OnColor, Event
   xloadct
   END
   ```idl```
This procedure opens a dialog from which the user can select from a set of predefined color tables. When the user clicks the name of a color table, it is loaded and the displayed GIF file changes appropriately.

**Note** The IDL XLoadct color table dialog affects only 8-bit display devices.

Handling the Smooth Event
When the user selects Smooth from the Analyze menu, a smooth operation is performed on the displayed GIF image. The smooth operation displays a smoothed image with a boxcar average of the specified width, which in the example code is 5.
To add the callback routines to handle the smooth operation, follow these steps:

1. Locate the DoSmooth routine place holder, which looks like this:

   ```idl
   PRO DoSmooth, Event
   END
   ```

2. Add the following code to handle the smooth operation:

   ```idl
   PRO DoSmooth, Event

   ; Get the image stored in the uvalue of the button widget.
   widget_control, Event.top, get_uvalue=image, /no_copy
   ; Make sure the image exists.
   if(n_elements(image) gt 0)then begin

   ; Smooth the image.
   image = smooth(image, 5)
   ; Display the smoothed image.
   tv, image
   ; Place the new image in the uvalue of the button widget.
   widget_control, Event.top, set_uvalue=image, /no_copy
   endif

   END
   ```

3. From the File menu, choose Save, which saves all your changes to the example_eventcb.pro file.

**Compiling and Running the Example Application**

To compile and run your example application, follow these steps:

1. At the IDL> command prompt, type the following:

   ```
   example
   ```

2. This action compiles and runs the example application. Figure 7-5 shows the example application and the IDL color table dialog.

   In the running application, you can open and display a GIF file. Then, you can open the IDL XLoadct dialog and change the color table used in displaying the image, or you can perform the smooth procedure on the image.
Figure 7-5: Running Example Application and XLoadct Dialog
IDL GUIBuilder Tools

You will use the following tools to design and construct a graphical interface using the IDL GUIBuilder:

- The IDL GUIBuilder Toolbar, which you use to create the widgets that make up your interface.
- Widget Properties dialog, which you use to set widget attributes and event properties.
- Widget Browser, which you can use to see the widget hierarchy and to modify certain aspects of the widgets in your application.
- The Menu Editor, which you use to define menus to top-level bases and buttons.
- The Bitmap Editor, which you use to create or modify bitmap images to be displayed on button widgets.
- The IDLDE to modify, compile, and run the generated code.

Using the IDL GUIBuilder Toolbar

The IDL GUIBuilder has its own toolbar in the IDE, which you use to create the widgets for your user interface. The toolbar is shown in Figure 7-6.
These are the widget types you can create using the IDL GUIBuilder toolbar:

<table>
<thead>
<tr>
<th>Widget</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base</td>
<td>Creates a container for a group of widgets within a top-level base container. A top-level base is contained in the IDL GUIBuilder window, and you build your interface in it. Use base widgets within the top-level base to set up the widget hierarchy, layout, and to organize the application. For example, you can use a base widget to group a set of buttons. For information on base properties, see “Base Widget Properties” on page 202.</td>
</tr>
<tr>
<td>Button</td>
<td>Creates a push button. The easiest way to allow a user to interact with your application is through a button click. You can have button widgets display labels, menus, or bitmaps. For information on button properties, see “Button Widget Properties” on page 212.</td>
</tr>
<tr>
<td>Radio Button</td>
<td>Creates a toggle button that is always grouped within a base container. Use radio buttons to present a set of choices from which the user can pick only one. For information on radio button properties, see “Button Widget Properties” on page 212.</td>
</tr>
<tr>
<td>Checkbox</td>
<td>Creates a checkbox, which you can use either as a single toggle button to indicate a particular state is on or off or as a list of choices from which the user can select none to all choices. Checkboxes are created within a base container. For information on checkbox properties, see “Button Widget Properties” on page 212.</td>
</tr>
<tr>
<td>Text</td>
<td>Creates a text widget. Use text widgets to get input from users or to display multiple lines of text. For information on text widget properties, see “Text Widget Properties” on page 216.</td>
</tr>
<tr>
<td>Label</td>
<td>Creates a label. Use label widgets to identify areas of your application or to label widgets that do not have their own label property. Use labels when you have only a single line of text and you do not want the user to be able to change the text. For information on label widget properties, see “Label Widget Properties” on page 221.</td>
</tr>
<tr>
<td>Horizontal and Vertical Sliders</td>
<td>Creates a slider with a horizontal or vertical layout. Use slider widgets to allow the user to control program input, such as adjust the speed of movement for a rotating image. For information on slider properties, see “Slider Widget Properties” on page 222.</td>
</tr>
<tr>
<td>Droplist</td>
<td>Creates a droplist widget, which you can use to present a scrollable list of items for the user to select from. The droplist is an effective way to present a lot of choices without using too much interface space. For information on droplist properties, see “Droplist Widget Properties” on page 224.</td>
</tr>
<tr>
<td>Listbox</td>
<td>Creates a list widget, which you can use to present a scrollable list of items for the user to select from. For information on listbox properties, see “Listbox Widget Properties” on page 226.</td>
</tr>
</tbody>
</table>
Note  The Select Cursor button returns the cursor to its standard state, and it indicates that the cursor is in that state. After you click on another button and create the selected widget, the cursor returns to the selection state.

Creating Widgets
All widgets for a user interface must be descendents of a top-level base; in the IDL GUIBuilder window, all widgets must be contained in a top-level base widget. When you open a IDL GUIBuilder window, it contains a top-level base. You can add base widgets to that top-level widget to form a widget hierarchy. The added bases can act as containers for groups of widgets.

To create a widget:

Click on the appropriate button on the toolbar, then drag out an area within the top-level base widget. When you release the mouse button, a widget the size of the dragged-out area is created.

Or

Click on the appropriate button on the toolbar, then click within the top-level base area. This action creates a widget of the default size.

After you add widgets to a top-level base, you can resize, move, and delete them, and you can change their parent base. You can also set properties for each widget. For information on how to operate on widgets, see “Widget Operations” on page 176, and for information on setting properties, see “Using the Properties Dialog” on page 166.

Using the Properties Dialog
For each widget, you can define attribute and event procedure properties. A widget’s attributes define how it will display on the screen and its basic behaviors. The attributes you can set for a selected widget are displayed on the Attributes tab of the Properties dialog. These attributes are initially set to default values.

<table>
<thead>
<tr>
<th>Widget</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Draw Area</td>
<td>Creates a draw area, which you can use to display graphics in your application. The draw area can display IDL Direct Graphics or IDL Object Graphics, depending on how you set its properties. For information on the draw area properties, see “Draw Widget Properties” on page 229.</td>
</tr>
<tr>
<td>Table</td>
<td>Creates a table widget, which you can use to display data in a row and column format. You can allow users to edit the contents of the table. For information on the table widget properties, see “Table Widget Properties” on page 234.</td>
</tr>
</tbody>
</table>
Event procedures are the predefined set of events a widget can recognize. When you write an application, you decide if and how the widget will respond to each of the possible events. The events that a selected widget recognizes are displayed on the Events tab of the Properties dialog. The event Values are initially undefined. Supply event routine names for only those events to which you want the application to respond.

**Opening the Properties dialog**

To open the Properties dialog for a widget:

Right-click on the widget in the IDL GUIBuilder window, and choose Properties from the menu.

Or

Select the widget, and choose Properties from the Edit menu.

These actions open a Properties dialog similar to the one shown in Figure 7-7.

The status area at the bottom of the Properties dialog, contains a description of the currently selected attribute or event. In addition, for each property that maps directly to an IDL keyword, there is a tool-tip that provides the name of the IDL keyword.

To display a tool-tip:

Place the cursor over the property name. The tool-tips are displayed only for properties that map to IDL keywords.

**Note** If you have multiple widgets selected in the IDL GUIBuilder window, the Properties dialog displays the properties for the primary selection, which is indicated by the darker, filled-in sizing handles around the widget. When you select multiple widgets, only one is marked as the primary selection.
To keep the Properties dialog on top:

Click the push pin button.

The Properties dialog will close as soon as it loses focus, unless you click the push pin button. If you click the push pin button, the Properties dialog stays on top and updates to reflect the properties of the currently selected widget.

To close the Properties dialog when the push pin is being used:

Click the push pin again, and the dialog will close when it loses focus.

Or

Press Escape while the dialog has focus.

Or

Click the close X in the upper right corner of the dialog.

Any changes you make to values in the Properties dialog are automatic; you will see the results of all visual changes immediately. For example, any changes you make to the alignment or column setting will change the layout position of the widget immediately.

All widgets share a common set of properties, and each widget has its own specific properties. These properties are arranged in the following order in the Properties dialog Attributes tab:
Chapter 7: Using the IDL GUIBuilder

- The Name property
- An alphabetical list of common and widget-specific properties, combined

On the Properties dialog Events tab, the properties are displayed in alphabetical order with common and widget-specific events combined.

For information on the properties you can set for each widget, see “Widget Properties” on page 196.

**Entering Multiple Strings for a Property**

There are several widget properties that you can set to multiple string values. The attribute's Value field contains a popup edit control in which you can enter multiple strings.

To enter more than one string in the edit control, do one of the following:

- Type in a string, then press Control+Enter, at the end of each line.
- Or
- Type in a string, then press Control+j, at the end of each line.

These actions move you to the next line. When you have entered the necessary string, press Enter to set the values.

**Using the Widget Browser**

The Widget Browser of the IDL GUIBuilder is a dialog window that presents the current GUI in a tree control. This presents the user with a different view into the GUI they are designing.

To start the Widget Browser:

- Right-click on any component in a IDL GUIBuilder window, then choose Browse from the menu.

This action opens the Widget Browser, like the one shown in Figure 7-8.

The Widget Browser is helpful when you want to see your widget hierarchy and when you need to operate on overlapping widgets in your interface layout, which can happen when you design an interface to show or hide widgets on specific events. For an example that uses the Widget Browser for this purpose, see “Controlling Widget Display” on page 191.

**Note** In the Widget Browser, there is no indication of defined menus.
You can expand the widget tree by clicking on the plus sign, or contract it by clicking on the minus sign.

When you select a widget in the hierarchy by clicking on it, the widget is selected in the IDL GUIBuilder window, and the Properties dialog updates to display the selected widget’s properties.

Right-click on a component to display a context menu from which you can cut, copy, paste, or delete the widget. From the context menu, you can also open the Properties dialog and the Menu Editor, when appropriate. To delete a widget from the Widget Browser, use the context menu, or select a widget and press the Delete key.

To change a widget’s Name property in the Widget Browser:

Select the widget name with two single clicks on the name. This action changes the name into an editable text box in which you can enter the new name. The Name property must be unique to the widget hierarchy.

You can also move a widget from one parent base to another, simply by dragging and dropping the component. (This action is equivalent to pressing Alt and dragging the widget in the IDL GUIBuilder window.)

For more information on other ways to operate on widgets, see “Widget Operations” on page 176.
Using the Menu Editor

You can add menus to top-level bases or to buttons that have the Type property set to Menu. To define menus for your interface, use the Menu Editor, which is shown in Figure 7-9 with defined menus. This dialog allows you to define menus, menu items, submenu titles, and submenus, and all their associated event procedures.

For the instructions on how to define the menus shown in Figure 7-9, see “Defining Menus for the Top-level Base” on page 153.

To define basic menus, menu items, submenu titles, and submenus and their associated event procedures, to top-level bases, follow these general steps:

1. Open the Menu Editor by doing one of the following:
   - From the Edit menu, choose Menu.
   - Or
   - Right-click on a top-level base, then choose Edit Menu.

2. To define a top-level menu in the Menu Editor, enter a Menu Caption, and click Insert. When you are defining menus for a top-level base, the top-level menus are aligned along the left edge of the menu list, and following the indentation indicates the nesting in the menu.

   ![Figure 7-9: Menu Editor Dialog](image)
Note  The Menu Caption is the name that appears on the menubar. If you are defining a top-level menu for a base, you do not need to supply an Event Procedure. On button menus however, the button’s Label property acts as the top-level menu, and the first level of menus in the editor serve as menu items, and thus need defined Event Procedures.

3. To define a menu item on a new line in the editor, click the right arrow, enter a Menu Caption and its associated Event Procedure, and then click Insert.

The Menu Caption is the name you want to appear on the menu. The Event Procedure is the name of the routine that will be called when the menu item is selected.

Note  For top-level bases, you must indent a line to make it a menu item and enable the Event Procedure field.

4. To define a submenu title, enter the Menu Caption, and click Insert. It is not necessary to define Event Procedures for submenu titles.

5. To define submenus to a submenu title, enter the Menu Caption and the Event Procedure, indent the item another level by using the right arrow, and click Insert. Enter the submenus you want at this level of indentation.

6. To define another top-level menu or menu item, enter the information, click the left arrow until the indentation is appropriate, and click Insert.

7. To define a separator, select a blank line, or select the line you want the separator after, then click the separator button (which has a line on it and is above the arrow buttons).

8. To save your defined menus, Click OK in the Menu Editor. When you do so, the menu items will appear on the top-level base. To test the display of the menus, click on them.

Note  Under Microsoft Windows, including the ampersand character (&) in the Menu Caption causes the window manager to underline the character following the ampersand, which is the keyboard accelerator. This functionality is supported in the Menu Editor. If you are designing an application to run on other platforms however, avoid the use of the ampersand in the Menu Caption.

To move a menu item to a new position:

Select the menu item, and click the up or down arrow on the right side of the dialog until the menu item is in the desired position. Then, click OK.
To add a menu item in the middle of existing menu items:

Select the line you want the new item to follow, then click Insert. This action adds a new line, for which you can enter a Menu Caption and Event Procedure.

To make a menu item display disabled initially:

Click the Enabled checkbox (to uncheck it). All menu items are enabled by default.

To delete a menu item:

Select the item, then click Delete.

To delete a menu:

Delete each contained menu item, then delete the top-level menu.

**Adding Menus to Buttons**

You can also create buttons that contain menus. To add a menu to a button, follow these basic steps:

1. Click on the Button widget tool on the toolbar, then click on the top-level base area. This action creates a button of the default size.

2. Right-click on the button and choose Properties, which opens the Properties dialog.

3. In the Properties dialog, click on the Type attribute Value arrow, then choose Menu from the droplist.

4. Right-click on the button, then choose Edit Menu, which opens the Menu Editor. You can define the menu items and submenus with the Menu Editor, using the general steps described above.

**Note**  
For buttons, the button Label property acts as the top-level menu, and the first level of menus in the Menu Editor serve as menu items, and therefore require defined Event Procedures (unlike top-level menu items defined to bases).

5. After you have defined all the necessary menus, click OK. When you do so, the menus are saved, and, in the IDL GUIBuilder, the button Label property is displayed as the top-level menu.
To view menus on buttons:

Immediately after creating the menu (after clicking OK in the Menu Editor), click on the button, and the menus will display.

Or

At any other time, right-click on the button, and then choose Show Menu. After you do this, you can click on the button to view the menu items. To view the menus at any other time, choose Show Menu again.

Using the Bitmap Editor

Use the Bitmap Editor to create 16 color bitmaps to be displayed on push buttons. The Bitmap Editor can read and write bitmap files (*.bmp). Using the editor, you can create your own bitmaps, or you can open existing bitmap files and modify them.

IDL supplies a set of bitmap files you can use in the buttons of your applications. The files are always available for loading. The bitmaps are located in the following directory:

`IDL_DIR\resource\bitmaps`

Placing a Color Bitmap on a Button

To display a bitmap on a button, follow these steps:

1. Right-click on the button widget, and choose Properties from the menu, which opens the Properties dialog for this button.

2. In the Type Value field, select Bitmap from the droplist.

3. In the Properties dialog, click on the arrow to the right of the Bitmap attribute, and do one of the following:

   To place an existing bitmap in the button, choose Select Bitmap, and select a bitmap file from the Open dialog. This action displays the bitmap on the button.

Note When Bitmap type is selected, the label property changes to Bitmap.

Or

To edit an existing bitmap and place it in the button, choose Edit Bitmap, then select the bitmap file from the Open dialog. This opens the bitmap in the Bitmap Editor, and assigns this as the bitmap to display on the button. It is displayed on the button when you save the file.

Or
To create a new bitmap and place it in a button, choose **New Bitmap**. This action opens the Bitmap Editor, which you can use to create the new bitmap. When you save the *.bmp file, it is placed on the button.

When you complete one of these processes, the filename of the selected bitmap appears in the Bitmap field of the Properties dialog, and the bitmap is displayed on the button.

**Using the Bitmap Editor Tools**

The Bitmap Editor tools allow you to select from the color palette, and then use the **Pencil** (pixel fill), the **Flood fill** (fill clear area), or the **Eraser** (clear or color areas). The Bitmap Editor tools are shown in Figure 7-10.

You can select a color by clicking on it in the color selection tool, or you can select your primary colors, the left-button and right-button colors, and then click on a tool and draw on the bitmap canvas. You can change the primary color selections at any time.

To select the left mouse button color:

- Left-click on the color in the color selection area.

To select a right mouse button color:

- Right-click on the color in the color selection area.
To use the left color:

With a tool selected, click or press and drag the right mouse button on the bitmap canvas.

To use the right color:

With a tool selected, click or press and drag the left mouse button on the bitmap canvas.

To change the size of the bitmap:

Drag the bitmap canvas to the desired size.

**Widget Operations**

The IDL GUIBuilder allows you to operate on widgets in many ways. You can select, deselect, move, cut, copy, paste, and delete widgets, and you can undo and redo operations.

**Selecting Widgets**

You can select a widget, then move it or resize it.

To select a widget:

Click on the widget.

To select more than one widget:

Press Shift and click on each widget.

Or

Press Control and click on the widgets. When you press Control, you can change the selection state by clicking again on the widget; pressing Control during selection allows you to toggle the selection state of a widget without affecting the selection state of any other widget.

Or

Press the left mouse button and drag out an area in the top-level base that includes the widgets you want to select. When you release the mouse button, widgets in the selection box are selected.

When you select multiple widgets, there is always one primary selection. The primary widget selection is indicated with the dark, filled-in selection handles. If
you open the Properties dialog with multiple widgets selected, the properties displayed are those for the primary selection.

**Note** When selecting multiple widgets, you can select only widgets that share the same base widget as their parent.

**Moving and Resizing Widgets**

You can move widgets around in their parent base by dragging the widget to a new location or by using the arrow keys.

To move a widget to a new base; to give a widget a new parent base within the same top-level base:

Press Alt and drag and drop the widget on the new parent base.

Or

Right-click on the widget, choose Cut from the menu, right-click on the new base widget, and choose Paste from the menu.

**Note** When you drag a widget to a new location, either in the same base or in another base, and the Layout attribute for the parent base is set to Column or Row, a blue line displays to indicate where the widget will be placed relative to other widgets in the base.

To resize a widget:

Click on a sizing handle, and drag to the desired size.

**Cutting, Copying, and Pasting Widgets**

You can cut, copy, and paste widgets within the same base or to another base in another IDL GUIBuilder window, using the Edit menu items, or the toolbar buttons, or a context menu (opened with a right-click on the widget).

To cut or copy a selected widget, or to paste a widget from the clipboard:

Choose the appropriate operation from the Edit menu, or from the IDLDE toolbar.

Or

Right-click on the widget and select the appropriate operation from the menu. If you are pasting, right-click on the base widget you want to paste into.

Or
Select the widget and use standard windows keyboard shortcuts to cut, copy, or paste the widget.

**Note** All cut or copied items are placed on a local clipboard (not on the system clipboard).

**Deleting Widgets**
To delete a widget:
- Select the widget and choose Delete from the Edit menu.
  - Or
  - Select the widget and press the Delete key.
  - Or
  - Right click on a widget and choose Delete from the menu.

**Undoing and Redoing Operations**
In the IDL GUIBuilder, you can undo or redo unlimited operations between save procedures. If you save the resource file, the operations are cleared from memory.
To undo an operation:
- Choose Undo from the Edit menu.
  - Or
  - Click the Undo button on the IDLDE toolbar.
  - Or
  - Press Control+z.
To redo an operation:
- Choose Redo from the Edit menu.
  - Or
  - Click the Redo button on the IDLDE toolbar.
  - Or
  - Press Control+y.
Generating Files

The IDL GUIBuilder generates the following two types of files:

- *.prc files that contain the resource definitions for the interface definition as displayed in the IDL GUIBuilder.
- *.pro files that contain the generated IDL source code. The generated *.pro files are portable across all IDL-supported platforms.

Generating Resource Files

The *.prc files contain the resource definitions for the graphical interface. You can open *.prc files in the IDL GUIBuilder and modify the interface at anytime. Do not attempt to modify this file directly.

To save a *.prc file for the first time:

Choose Save or Save As from the IDLDE File menu. This opens the Save As dialog, which allows you to select a location and indicate a file name for the *.prc file.

Generating IDL Code

The IDL GUIBuilder can generate these two kinds of *.pro IDL source code files:

- Widget definition code (*.pro files).
- Event-handling code (*_eventcb.pro files).

In the IDLDE, you can choose to generate either or both of these files for an interface.

To save the widget code to a *.pro file:

From the IDLDE File menu, choose Generate .pro, then choose Widget. This action opens the Save As dialog, which you can use to select a location and indicate a file name for the widget code. For example, enter “gui1.pro”.

Note: Never modify the generated *.pro interface file. If you decide to modify the application interface, use the IDL GUIBuilder, then regenerate the file. When you regenerate the widget code, the file is overwritten.

To save the event-handler *_eventcb.pro file, which contains the event routine place holders:

From the IDLDE File menu, choose Generate .pro, then choose Event. This action opens the Save As dialog, which you can use to select a location and indicate a name for the event code. By default, the event code has an
_eventcb.pro extension. Add a prefix that make sense with your other code. For example, enter gui1_eventcb.pro.

To save both the widget code and the event handler *.pro files:

From the IDLDE File menu, choose Generate .pro, then choose Both. This action opens the Save As dialog, which you can use to select a location and indicate a name for the widget code. The event code file name is based on the name specified for the widget code. For example, if you enter “app1.pro” in the File name field, the event code file will be named “app1_eventcb.pro”.

Note If you save both files at the same time, IDL puts the RESOLVE_ROUTINE procedure in the generated widget code. The procedure contains the name of the related *_eventcb.pro event-handler file so that it will be compiled and loaded with when you run the widget code. If you generate the files separately, or generate only the widget file, you must compile the files yourself or add the routine to the *.pro widget code (which is the only modification you should ever need to make to the widget code).

Notes on Generating Code a Second Time

When you modify a interface and save the *.prc file, it is overwritten, which should not be a problem. If you decide to change your interface, you will however need to regenerate the widget code and thus overwrite the *.pro widget code file.

Note that if you regenerate either of the *.pro files, they are overwritten. When writing code, you should modify only the generated event-handling file (*_eventcb.pro); and, you should never modify the generated widget code (the *.pro file). This allows you to change the interface and regenerate the definition code without losing modifications in that file. This should simplify the procedures you need to take to update or change an interface.

Because it is modular, the event-handler code is simple to modify after you change the interface definitions. When you regenerate the IDL source code files, do one of the following to preserve the modifications you have made to the *_eventcb.pro file:

• Move the existing *_eventcb.pro file to another location, then generate a new file. You can easily copy your existing callback routines and paste them into the new file. Then, write the new event callback routines to the newly generated procedures.
• Do not to generate the event code, then add any new event procedures to the event-handling code. If you do this, you must keep track of all the event procedures you add while modifying the interface.

**IDL GUIBuilder Examples**

After you define your interface and generate IDL code using the IDL GUIBuilder, you will write the code that controls the application’s behavior. You can modify the code, compile it, and run it using the IDLDE.

Generally, you will be writing the event-handler callbacks for the procedures located in the generated *_eventcb.pro file. While doing this, you might like to handle initialization states, have multiple GUIs work together, add compound widgets, or control widget display. For examples of how to handle these different types of events, see the following sections:

- “Understanding IDL GUIBuilder Event Handling Code” on page 181
- “Writing Event Callback Routines” on page 182
- “Handling Initialization Arguments” on page 184
- “Integrating Multiple Interfaces” on page 185
- “Adding Compound Widgets” on page 188
- “Controlling Widget Display” on page 191

**Understanding IDL GUIBuilder Event Handling Code**

When using the IDL GUIBuilder, you assign event procedures to specific events using the Properties dialog Events tab. The calling sequence for the events that you set are added to the generated *_eventcb.pro event callback code.

The argument that is passed into the specified event routine depends on the type of event being processed. Creation, realization, and destruction event routines are usually passed the ID of the involved widget, and all other callback routines are passed the appropriate IDL widget event structure.

It is a normal operation in applications to change the attributes of the interface when a certain events occur. One method used in handling events for IDL GUIBuilder generated applications is the UNAME keyword, or the Name property, given to all created widgets. (In a programmatically created IDL application, this action is handled using information stored in a widget component’s user value.)
When you create a widget in the IDL GUIBuilder, IDL gives it a name unique to the widget hierarchy to which it belongs. You can rename the widget using the Name property.

In the generated code, this name is specified by the UNAME keyword. Because these names are unique, you can use the WIDGET_INFO function with the FIND_BY_UNAME keyword in your event callback routines to get the IDs of widgets in the interface application.

**Note** For information on properties, see “Using the Properties Dialog” on page 166, and see “Widget Properties” on page 196.

### Writing Event Callback Routines

This short example shows how basic event processing works in IDL GUIBuilder-generated code. The example demonstrates how to use the FIND_BY_UNAME keyword to obtain the IDs of other widgets in the interface.

To create this simple example application, follow these steps:

1. From the IDLDE File menu, choose New, then choose GUI. This action opens a new IDL GUIBuilder window.

2. In the IDL GUIBuilder window, right-click on the contained top-level base, and choose Properties from the menu. This action opens a Properties dialog, like the one shown in Figure 7-7 on page 168.

3. In the open Properties dialog, click the push pin button to keep the dialog open and on top.

4. On the Properties dialog Attributes tab, set the following for the top-level base:
   - Set the Component Sizing property to Default.
   - Set the Layout property to Column.

5. On the IDL GUIBuilder toolbar, click the Label widget button (the letter A).

6. Click on the top-level base area, which adds to the interface a label widget of the default.

7. With the label widget selected, set the following attributes in the Properties dialog:
   - In the Name field, enter “clock”.
   - Set the Alignment attribute to Center.
   - Set the Component Sizing attribute to Default.
   - In the Text field, enter “No Time Currently Available”.
8. On the IDL GUIBuilder toolbar, click the Button widget (the rectangle icon).
9. Click on the top-level base area, which adds a button widget to the interface.
10. With the button selected, set the following attributes in the Properties dialog:
    • In the Label field, enter “Time”.
11. In the Properties dialog, click the Events tab and do the following:
    • In the OnButtonPress field, enter “OnPress”.

Your interface definition should look like the one shown in Figure 7-11.

12. From the IDLDE File menu, choose Save, which opens the Save As dialog.
13. In the Save As dialog, select a location, enter “time.prc” in the File name field, and click Save. This action saves the interface definition to a resource file.
14. From the IDLDE File menu, choose Generate.pro, then choose Both, which opens the Save As dialog.

Figure 7-11: Handling Events Example Application
15. In the Save As dialog, select the location, enter “time.pro” in the File name field, and click Save. This action saves the time.pro widget code file and the time_eventcb.pro event callback code to the specified directory.

16. From the IDLDE File menu, choose Open, which launches the Open dialog.

17. In the Open dialog, locate and select the time_eventcb.pro file, then click Open. This action opens the file in the IDLDE.

18. In the file, locate the OnPress event procedure placeholder, and add the following IDL code to handle a button press, like this:

```idl
PRO OnPress, Event

; Get the widget ID of the label widget.
Label = widget_info(Event.top, find_by_uname='clock')

; Set the value of the label widget to current time.
widget_control, Label, set_value=Systime(0)

END
```

The first command gets the ID of the label widget by searching the widget hierarchy for a widget named “clock”. This is the name that you gave the label widget in the IDL GUIBuilder Properties dialog. Once the ID is found, the second command sets the value of the label widget to the current system time.

19. From the Run menu, choose Compile time_eventcb.pro, which saves and compiles the file.

20. To execute the program, enter the following at the IDL command prompt:

```
time
```

This compiles and runs the time.pro file. In the running application, you can press the Time button to cause the current time to be displayed in the label.

**Handling Initialization Arguments**

You can provide runtime initialization information to the generated *.pro widget code by modifying the *_eventcb.pro file. Keywords provided to the generated widget interface procedure are passed to the post creation routines using the _EXTRA keyword.

If a routine is defined with the _EXTRA keyword parameter, you can add unrecognized keyword and value pairs, and the pairs are placed in an anonymous structure. The name of each unrecognized keyword becomes a tag name, and each value becomes the tag value.
You will use this feature most often when your application launches floating or modal dialogs, but the functionality is always available.

For example, if you want to display a dialog at the creation of an application, you would follow these basic steps:

1. Create an interface using the IDL GUIBuilder.

2. After creating the interface, open the Properties dialog for the top-level base and set the PostCreation event for the top-level base widget to a routine name, such as "OnCreate".

3. Save the interface definition and generate the IDL source code.

4. In the generated *_eventcb.pro event code file, locate the "OnCreate" routine place holder, which looks like this:

   ```idl
   PRO OnCreate, wWidget, _EXTRA=_VWBExtra_
   END
   ```

5. To process a specific keyword in this post creation routine, declare the keyword in the procedure statement and add the processing code to the procedure.

   For example, to process the DO_DIALOG keyword in the defined OnCreate procedure, add the DO_DIALOG keyword to the procedure, and add the logic to handle it to the event callback routine. The completed procedure should look like this:

   ```idl
   PRO OnCreate, wWidget, DO_DIALOG=DO_DIALOG, _EXTRA=_VWBExtra_
       ; If DO_DIALOG is set, display a simple message box.
       if( Keyword_Set(DO_DIALOG) ) then $
           status = Dialog_Message("On Dialog Set")
   END
   ```

6. Save the file, then compile and generate the application. To show the dialog at creation time, enter the following at the IDL command prompt:

   ```idl
   <ProgramName>, /DO_DIALOG
   ```

**Integrating Multiple Interfaces**

You can create multiple interfaces with the IDL GUIBuilder then integrate them to form the complete application hierarchy. This example shows you how to construct two interfaces and integrate them.
The first interface you will create is the main window, and it will consist of a simple push button that will launch a modal dialog. The second interface you will create is the modal dialog, and it will display a close button.

**Creating the Main Window**

To create the main window, follow these steps:

1. From the IDLDE File menu, choose New, then choose GUI. This action opens a new IDL GUIBuilder window with a top-level base.

2. On the IDL GUIBuilder toolbar, click on the button widget button, then click on the top-level base. This action adds a button of the default size to the base. You can place the button anywhere in the base.

3. Right-click on the newly created button, and choose Properties from the context menu. This action opens a Properties dialog, like the one shown in Figure 7-7 on page 168.

4. In the Properties dialog, click the push pin button to keep the dialog open and on top.

5. With the button selected, set the following in the Properties dialog:
   - In the Label value field, enter “Modal Dialog”.

6. Click on the Properties dialog Events tab, and do the following:
   - In the OnButtonPress value field, enter “OnPress”.

7. From the File menu, choose Save, which opens the Save As dialog.

8. In the Save As dialog, select a location, enter “maingui.prc” in the File name field, and click Save. This action saves the interface definition to an IDL resource file.

9. From the File menu, choose Generate .pro, then choose Both, which opens the Save As dialog.

10. In the Save As dialog, select a location, enter “maingui.pro” in the File name field, and click Save. This action saves the maingui.pro widget code and the maingui_eventcb.pro event-handler code.

11. From the File menu, choose Open, which launches the Open dialog.

12. In the Open dialog, select the maingui_eventcb.pro file, and click Enter.
13. In the open file, locate the `OnPress` event procedure place holder, then enter the code that launches the modal dialog, like this:

```idl
PRO OnPress, Event
    modalgui, group_leader=Event.top
END
```
You will create the “modalgui” dialog in the next set of steps. Note that you set the `GROUP_LEADER` keyword here because the modal dialog requires it.

14. From the Run menu, choose `Compile maingui_eventcb.pro`. This action saves and compiles the file.

Creating the Modal Dialog
To create the modal dialog, follow these steps:

1. Open a new IDL GUIBuilder window.
2. In the IDL GUIBuilder window, select the top-level base, and set the following in the Properties dialog:
   - Set the Modal attribute to True.
   - In the Title field, enter “Modal Dialog”.
3. On the IDL GUIBuilder toolbar, click the button widget, then click on the top-level base. This action adds a button to the top-level base, and you can place it anywhere in the base.
4. With the new button selected, set the following in the Properties dialog:
   - In the Label field, enter “OK”.
5. Click on the Properties dialog Events tab, and do the following:
   - In the OnButtonPress value field, enter “OnModalPress”.
6. From the File menu, choose Save, which opens the Save As dialog.
7. In the Save As dialog, select a location, enter “modalgui.prc” in the Filename field, and click Save. This action saves the interface definition to an IDL resource file.
8. From the File menu, choose Generate.pro, then choose Both, which opens the Save As dialog.
9. In the Save As dialog, select a location, enter “modalgui.pro” in the Filename field, and click Save. This action saves the modalgui.pro widget code file and the modalgui_eventcb.pro event callback file.
10. Open the modalgui_eventcb.pro file and locate the OnModalPress procedure place holder. Then, add the following code so that the dialog closes when the button is pushed:

```idl
PRO OnModalPress, Event
    widget_control, Event.top, /destroy
END
```

11. Save and compile this file.

**Running the Example Application**

Enter the following at the IDL command prompt:

```
maingui
```

This command runs the main window. You can press the Modal Dialog button, and the modal dialog is displayed. When you press the OK button on the modal dialog, the dialog exits.

**Adding Compound Widgets**

The IDL GUIBuilder tools do not allow you to add a compound widget directly to your interface. You can, however, modify your event code to add a compound widget.

To add a compound widget to a IDL GUIBuilder generated interface, you will follow these basic steps:

1. Add the compound widget to the widget tree in a PostCreation event callback procedure.
2. Handle the events generated by the compound widget in the Handle Event callback function. Set this event function value for the base widget that will contain the compound widget.

**Adding a Compound Widget to an Interface**

This example demonstrates how to add a compound widget to an application constructed with the IDL GUIBuilder. The application contains a label and a CW_FSLIDER compound widget. In the running application, the values generated by CW_FSLIDER will be displayed in the label widget.

To create this application, follow these steps:

1. From the IDLDE File menu, choose New, then choose GUI, which opens a new IDL GUIBuilder window with a top-level base.
2. Right-click on the base and choose Properties, which opens the Properties dialog for the top-level base.

3. In the Properties dialog, click the push pin button to keep the dialog on top.

4. In the Properties dialog of the top-level base, set the following properties:
   - Set the Component Sizing attribute to Default.
   - Set the Layout attribute to Column.

5. To add the label, click the label Widget button (the single letter) on the toolbar, then click on the top-level base. This action creates a label widget of the default size.

6. With the label selected, set the following in the Properties dialog:
   - In the Name value field, enter “label”.
   - Set the Alignment attribute to Center.
   - Set the Component Sizing attribute to Default.
   - In the Text value field, enter “000.000”.

7. Click the Base widget button on the toolbar (window icon), and click on the top-level base. This action adds a base to the top-level base.

8. With the new base widget selected, set the following in the Properties dialog:
   - Set the Component Sizing attribute to Default.

9. In the Properties dialog, click on the Events tab and set the following base widget event values:
   - In the Handle Event Value field, enter “HandleEvent”. This is the name of the function that will handle the compound widget events.
   - In the PostCreation Value field, enter “AddCW”. This is the name of the event routine that will create the compound widget.

10. To save the portable resource file, choose Save from the File menu, which opens the Save As dialog.

11. In the Save As dialog, select a location, enter “compound.prc” in the File name field, and click Save. This saves the interface definition to an IDL resource file.

12. From the File menu, choose Generate .pro, then choose Both, which opens the Save As dialog.
13. In the Save As dialog, enter “compound.pro”, and click Save. This action generates the compound.pro widget code file and the compound_eventcb.pro event-handler file.

14. From the IDLDE File menu, choose Open, which launches the Open dialog.

15. In the Open dialog, select the compound_eventcb.pro event file, then click Open, which opens the file in the IDLDE.

16. In the file, locate the AddCW event routine placeholder, and modify the code to add the CW_FSLIDER compound widget to the base widget. The routine should look like this:

```idl
PRO AddCW, wWidget
    idslide = cw_fslider(wWidget, /suppress_value)
END
```

17. Add the event callback routines to the generated HandleEvent function. The function should look like this:

```idl
FUNCTION HandleEvent, Event
    ; Fslider event structure is an anonymous structure, so
    ; the following will return "" if it is from fslider.
    if(Tag_Names(Event, /structure_name) eq "") then begin
        ; Get the id of the label widget using its name.
        id = widget_info(Event.top, find_by_uname='label')

        ; Set the value of the label, to the value in the slider.
        widget_control, id, set_value= $ String(Event.value, format='(f5.2)')
        return, 0
        ; Halt event processing here.
    end
    return, Event
    ; By Default, return the event.
END
```

Note that the callback routine finds the label widget using the FIND_BY_UNAME keyword with the name value you gave the widget in the Properties dialog.

18. From the Run menu, choose Compile compound_eventcb.pro, which saves and compiles the file.
Running the Example
To run the application, enter the following at the IDL command prompt:

    compound

This action compiles and runs the application. In the running application, move the CW_FSLIDER and the value is placed in the label.

Controlling Widget Display
This example demonstrates how to use the IDL GUIBuilder to create an interface that contains overlapping sub-bases containing different types of widgets. The example shows how you can display and hide overlapping controls in an interface created in the IDL GUIBuilder, and it incorporates using the Widget Browser. Note that this example is a slightly more complicated than the others.

This example constructs an interface with the following widgets:

- A droplist.
- A sub-base that contains two sub-bases:
  - One sub-base containing a text widget.
  - One sub-base containing a button.

The two contained sub-bases overlap and the visibility of each is controlled by the value selected in the droplist. When users select an item in the droplist, one sub-base is hidden and the other one is displayed.

Creating the Interface
To create this application interface, follow these steps:

1. From the IDLDE File menu, choose New, then choose GUI. This action opens a new IDL GUIBuilder window with a top-level base.
2. Right-click on the top-level base, and choose Properties from the menu. This action opens a Properties dialog.
3. In the Properties dialog, click the push pin button to keep the dialog open and on top.
4. In the Properties dialog, do the following:
   - Set the Component Sizing attribute to Default.
   - Set the Layout attribute to Column.
5. On the IDL GUIBuilder toolbar, click on the droplist widget button, then click on the top-level base. This action creates a droplist in the base area.
6. With the droplist select, set the following in the Properties dialog:
   - In the Title value field, enter “Active Base”.
   - In the Initial Value field, click on the arrow. This displays a popup edit control in which you can enter “Base One”, press Control+Enter, enter “Base Two”, and press Enter.

   **Note** In the pop-up edit control for the Initial Value attribute, press Control+Enter to move to the next line. To set the values and close the popup edit control, press Enter.

7. Click on the Properties dialog Events tab, and do the following:
   - In the OnSelectValue field, enter “OnSelect”.

8. On the IDL GUIBuilder toolbar, click on the base widget button, then click on the top-level base. This action adds a base widget of the default size to the interface.

9. With the new base selected, set the following attributes in the Properties dialog:
   - In the Name value field, enter “base0”.
   - Set the Frame attribute to True.

10. On the IDL GUIBuilder toolbar, click on the base widget button, then click on the base you just added. This action adds a base widget to the “base0” widget.

11. With the newly-added base selected, set the following attributes in the Properties dialog:
   - In the Name value field, enter “base1”.
   - Set the Component Sizing attribute to Explicit.
   - In the X Offset value field, enter “0”.
   - In the X Size value field, enter “200”.
   - In the Y Offset value field, enter “0”.
   - In the Y Size value field, enter “200”.

12. Right-click on a base, and choose Browse from the context menu. This action opens the Widget Browser.

13. In the Widget Browser, right-click on base1, and choose Copy, which copies the widget to the local clipboard.
14. In the Widget Browser, right-click on “base0”, and choose Paste, which pastes the copied base into the “base0” widget. The new base is called “base1_0”.

15. In the Widget Browser, select “base1_0”. This action selects the base in the IDL GUIBuilder window and updates the Properties dialog with the appropriate properties and values.

16. With “base1_0” selected, set the following attributes in the Properties dialog:
   - In the Name value field, enter “base2”.
   - Set the Component Sizing attribute to Explicit.
   - In the X Offset value field, enter “0”.
   - In the X Size value field, enter “200”.
   - In the Y Offset value field, enter “0”.
   - In the Y Size value field, enter “200”.

17. From the File menu, choose Save, which opens the Save As dialog.

18. In the Save As dialog, select a location, enter “visible.prc” in the File name field, and click Save. This action saves the interface definition. (The completed application is shown in Figure 7-12 on page 194.)

19. In the Widget Browser, select “base1”.

20. With “base1” selected, set the following attribute in the Properties dialog:
   - Set the Visible attribute to False. This will hide “base1” and make “base2” visible.

21. On the IDL GUIBuilder toolbar, click the button widget button, then click on “base2” in the IDL GUIBuilder. This action adds a button to the base widget. Place the button anywhere in this base.

22. With the button selected, set the following attribute in the Properties dialog:
   - In the Label value field, enter “Button 2”.

23. In the Widget Browser, select “base2”, and set the following attribute in the Properties dialog:
   - Set the Visible attribute to False, which hides the base.

24. In the Widget Browser, select “base1”, and set the following attribute in the Properties dialog:
• Set the Visible attribute to True, which shows the base.

25. On the IDL GUIBuilder toolbar, click the label widget button, then click on “base1”. This action adds a label to “base1”. You can place the label anywhere in this base.

26. With the label widget selected, set the following attribute in the Properties dialog:
   • In the Text value field, enter “Label 1”.

27. From the File menu, choose Save, which saves the changes to the visible.prc resource file.

The interface is now complete. It should look similar to the one shown in Figure 7-12.
Generating and Modifying the Code

To generate and modify the code, follow these steps:

1. From the File menu, choose Generate.pro, then choose Both, which opens a Save As dialog.

2. In the Save As dialog, select a location, enter “visible.pro” in the File name field, and click Save. This action saves the visible.pro widget code file and the visible_eventcb.pro event-handler file.

3. From the File menu, choose Open, which launches the Open dialog.

4. In the Open dialog, select the visible_eventcb.pro file, and click Open. This action opens the file in the IDLDE.

5. In the visible_eventcb.pro file, locate the OnSelect event procedure placeholder, then add the following code:

```
PRO OnSelect, Event
    ; Toggle the mapping of the two IDL sub-bases and
    ; get the Widget IDs of the two sub-bases.
    wBase1 = Widget_Info(Event.top, find_by_uname="base1")
    wBase2 = Widget_Info(Event.top, find_by_uname="base2")

    ; Now update the mapping.
    widget_control, wBase1, map=(Event.index eq 0)
    widget_control, wBase2, map=(Event.index eq 1)

END
```

The added IDL code gets the Widget IDs of the sub-bases that you created and sets the mapping (hide or show) of these bases depending on the selected value of the droplist.

6. From the Run menu, choose Compile visible_eventcb.pro, which saves and compiles the file.

Running the Application

To run this application, enter the following at the IDL command prompt:

```
visible
```

This command executes the visible application. In the running application, you can change the selection in the droplist, and the action will change the displayed widget.
Widget Properties

For each widget type, there is a set of attribute values and a set of event values you can set using the IDL GUIBuilder Properties dialog. When you select a widget in the IDL GUIBuilder window or in the Widget Browser, the Properties dialog is updated to contain the properties for the selected widget. These properties include those common to all widgets and those specific to the selected widget.

On the Attributes tab of the Properties dialog, the attributes are set to default values and are arranged in the following order:

- The Name property.
- An alphabetical list of common and widget-specific properties, combined.

On the Events tab, the possible events for a widget are listed in alphabetical order, with the common and the widget-specific events combined. By default, no event values are set initially. When you enter a routine name for an event property, you are responsible for making sure that event procedure exists. IDL does not validate the existence of the specified routine.

For information on how to open and use the Properties dialog, see “Using the Properties Dialog” on page 166.

The rest of this chapter describes the properties you can set for each widget.

Common Widget Properties

There are several attribute and event property values you can set for all widgets. The attribute properties include the name of the widget and the sizing properties. The event properties include creation, realization, destruction, and tracking events.

Common Attributes

These are the common attributes, which you can set for all widgets:

Name

The Name attribute specifies the name of the component. This value can be any string that is unique to the widget hierarchy of the interface, but the string cannot contain spaces. For each widget you create in the IDL GUIBuilder, a default name is supplied, and this name is in the WID_<TYPE>_NUMBER format.

If you copy and paste a widget in the IDL GUIBuilder, the new widget is given a unique name based on the name of the one you copied. A number is added to the first widget’s name, or an existing number is incremented.
You can use the Name value for the widget in your event callback routines. For example, you can use the specified name to find the widget, using the FIND_BY_UNAME keyword to the WIDGET_INFO function. Set the name for each widget to a name that makes sense to you; set the name value to something that is easy to remember and easy to use in your code.

In the generated *.pro file, this value is specified with the UNAME keyword to the widget creation routines.

**Component Sizing**

The Component Sizing keyword determines how the component is sized, which is by one of the following methods:

- **Default**: The widget is sized to a natural or implicit size. This is the default setting for the attribute. For example, a label widget's natural size is determined by the size of the text it is displaying with extra space for margins. The default size for each widgets is controlled by several things, including displayed font size and the characteristics of the operating system displaying the interface.

- **Explicit**: The widget size is determined by several attributes, which include Layout for the base and its own X Size, and Y Size attributes.

In the generated *.pro widget file, this value is specified with the XSIZE, and YSIZE keywords to the widget creation routines.

**Note** The default size of text widgets on Motif is based on the width of text, but the default size for text widgets on Windows and Macintosh is approximately 20 characters.

**Frame**

The Frame attribute determines if the widget will have a frame or border around it. These are the possible values:

- **False**: The widget will have no frame drawn around it. This is the default value.
- **True**: The widget will have a frame or border around it.

In the generated *.pro widget file, this value is specified by the FRAME keyword to the widget creation routines.

**Note** The Frame attribute is not available for top-level base widgets.
Sensitive

The Sensitive attribute determines if the selected widget is active or not active on startup. You can set this value to determine if the user can access and manipulate the widget immediately after creation. These are the possible values:

- True: The widget is initially displayed as enabled and accepts keyboard or mouse input and generates events. This is the default value.
- False: The widget is initially displayed as disabled and does not accept keyboard or mouse input. The appearance of most widgets change when the False value is set, but the appearance does not always change to indicate this state.

In the generated *.pro file, this value is specified with the SENSITIVE keyword to the widget creation routines.

Note To change the sensitivity of a widget after the widget is created, use the WIDGET_CONTROL function with the SENSITIVE keyword.

X Offset

The X Offset attribute specifies the X offset of the component from its parent. The possible values for X Offset are 0 to n, in pixels; any number is valid. The Y Offset attribute specifies the Y offset.

In the generated *.pro file, this value is specified with the XOFFSET keyword to the widget creation routines.

Note The X Offset property value is not used with base widgets that have the Layout property set to Row or Column.

X Size

The X Size attribute specifies the width of the visible component in pixels. This property is disabled when Component Sizing is set to Default (and the default size is used). To enable this value, set Component Sizing to Explicit. The possible values for X Size are 0 to n, in pixels.

In the generated *.pro file, this value is specified with the SCR_XSIZE keyword to the widget creation routines.

Note If you add scroll bars to a widget, use the widget-specific X Scroll property to set the width of the virtual area.

Y Offset

The Y Offset attribute specifies the Y offset of the component from its parent in pixels. The possible values for Y Offset are 0 to n, in pixels; any number is valid. The X Offset attribute specifies the X offset.
In the generated *.pro file, this value is specified by the XOFFSET keyword to the widget creation routines.

**Note** The Y Offset property value is not used with base widgets that have the Layout property set to Row or Column.

Y Size

The Y Size attribute specifies the height of the visible component in pixels. This property is disabled when Component Sizing is set to Default (and the default size is used). To enable this value, set Component Sizing to Explicit. The possible values for Y Size are 0 to \( n \), in pixels.

In the generated *.pro file, this value is specified with the SCR_YSIZE keyword to the widget creation routines.

**Note** If you add scroll bars to a widget, use the widget-specific Y Scroll property to set the height of the virtual area.

Common Events

These are the common events, which you can set for all widgets (by default, no event values are initially set):

Handle Event

The Handle Event value is the function name that is called when an event arrives from a widget that is rooted in a IDL GUIBuilder-created widget in the hierarchy. All events are sent to this event function, except for creation and destruction events.

For example, if you add a compound widget to an interface, using the PostCreation event procedure for a base widget, you should set the Handle Event value for that parent base widget (for the compound widget's parent widget). Then, you can handle all the events returned by the compound widget using this event function value.

In the generated *_eventcb.pro file, the event function place holder looks like this:

```
Function <Name>, Event
   return, Event
End
```

*Name* is the name of the event function you specify. *Event* is the returned event structure, which is specific to the widget event.

For an example of how to handle the generated Handle Event function, see “Adding Compound Widgets” on page 188.
**OnDestroy**

The OnDestroy value is the routine name that is called when the widget is destroyed. In the generated *_eventcb.pro* file, the event calling sequence looks like this:

```
PRO <RoutineName>, wWidget
```

*RoutineName* is the name of the event procedure you specify, *wWidget* is the IDL widget identifier.

**OnRealize**

The OnRealize value is the routine name that is called automatically when the widget is realized. In the generated *_eventcb.pro* file, the event calling sequence looks like this:

```
PRO <RoutineName>, wWidget
```

*RoutineName* is the name of the event procedure you specify, *wWidget* is the IDL widget identifier.

**OnTimer**

The OnTimer value is the routine name that is called when a timer event is detected for a widget. In the generated *_eventcb.pro* file, the event calling sequence looks like this:

```
PRO <RoutineName>, Event
```

*RoutineName* is the name of the event procedure you specify, *Event* is the returned event structure, which has the 3 standard event tags and looks like this:

```
{ WIDGET_TIMER, ID:0L, TOP:0L, HANDLER:0L }
```

You must set timer events for a widget, using the WIDGET_CONTROL function. The code generated by the IDL GUIBuilder only routes the events.

**OnTracking**

The OnTracking value is the routine name that is called when the widget receives a tracking event, which occurs when the mouse pointer enters or leaves the region of the widget. In the generated *_eventcb.pro* file, the event calling sequence looks like this:

```
PRO <RoutineName>, Event
```

*RoutineName* is the name of the event procedure you specify, *Event* is the returned structure, which is of the following type:

```
{ WIDGET_TRACKING, ID:0L, TOP:0L, HANDLER:0L, ENTER:0 }
```

ENTER is 1 if the tracking event is an entry event, and 0 if it is an exit event.
Chapter 7: Using the IDL GUIBuilder

PostCreation

The PostCreation value is the routine name that is called after the widget is created, but before it is realized. In the generated *_eventcb.pro file, the calling sequence looks like this:

```
PRO <RoutineName>, wWidget

RoutineName is the name of the event procedure you specify. wWidget is the IDL widget identifier.
```
Chapter 7: Using the IDL GUIBuilder

Base Widget Properties

A base widget holds other widgets, including other base widgets. You can create groupings of widgets by using a base widget, thus forming a widget hierarchy.

When you open the IDL GUIBuilder, a top-level base is created, and you build your interface in this base. Top-level bases are a special class of the base widgets that are created without parent widgets; they act as the top-level parent in the widget hierarchy.

In the IDL GUIBuilder, you can add a menubar to the top-level base by using the Menu Editor.

In addition, you can make top-level bases float above their group leaders, with the Floating property, or you can make them modal dialogs, with the Modal property. Modal dialogs interrupt program execution until the user closes them. When you make a top-level base floating or modal, you must provide a group leader when calling the generated code, by using the GROUP_LEADER keyword.

When programming in IDL, you create base widgets using the WIDGET_BASE function, which is described in “WIDGET_BASE” on page 1205 of the IDL Reference Guide.

For more information on the Menu Editor, see “Using the Menu Editor” on page 171.

Note A base widget's layout is controlled by where you place it and the properties of its parent base.

Base Widget Attributes

For base widgets, you can set common attributes and base-specific attributes. For a list attributes common to all widgets, see “Common Attributes” on page 196.

Some of the base widget attributes apply to top-level bases only, and this limitation is noted in the following list of base widget attributes:

# of Rows/Columns

The # of Rows/Columns attribute specifies the number of Columns or Rows to use when laying out the base. This property is valid only when the Layout property is set to Column or Row. The possible values for this setting are 1 to n, and the default value is 1.

In the generated *.pro file, this value is specified with the COLUMN or the ROW keyword to the widget creation routine.

For information on other properties that control the layout of contained widgets, see Alignment, Layout, Space, X Pad, and Y Pad.
Alignment

The Alignment attribute defines how components are aligned in the base. The way in which the value of this property affects the display of widgets depends on the value of the Layout property. The following is a list possible values for the Alignment property, and each value description includes information on how it works with the Layout settings:

- Center: Aligns the contained widgets with the center this parent base. This is the default value. For this setting to take effect, the Layout setting must be Row or Column. With Row set, the contained widgets are vertically centered. With Column set, the contained widgets are horizontally centered.
- Top: Aligns contained widgets with the top of this parent base. For this setting to take effect, the Layout setting must be Row.
- Bottom: Aligns the contained widgets with the bottom of this parent base. For this setting to take effect, the Layout setting must be Row.
- Left: Aligns the contained widgets with the left side of this parent base. For this setting to take effect, the Layout setting must be Column.
- Right: Aligns the contained widgets with the right side of this parent base. For this setting to take effect, the Layout setting must be Column.
- Default: Uses the default layout.

In the generated *.pro file, these settings are specified with the BASE_ALIGN_CENTER, BASE_ALIGN_TOP, BASE_ALIGN_BOTTOM, BASE_ALIGN_LEFT, and BASE_ALIGN_RIGHT keywords to the widget creation routine.

For information on other properties that control the layout of contained widgets, see # of Rows/Columns, Layout, Space, X Pad, and Y Pad.

Allow Closing

The Allow Closing attribute determines if the top-level base can be closed by the user. By default, this value is set to True and the base can be closed. To make it so the top-level base cannot be close, set this value to False.

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

For information on other properties that control aspects of top-level bases, see the Allow Moving, Minimize/Maximize, System Menu, and Title Bar properties.
Base Widget Properties

Chapter 7: Using the IDL GUIBuilder

**Note** This property setting is used with top-level bases only. Note that this setting is only a hint to the window system and might be ignored by some window managers.

**Allow Moving**

The **Allow Moving** attribute determines if the base can be moved. By default, this value is set to True, and the base can be moved. To suppress this behavior, set this value to False.

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

For information on other property settings that control aspects of top-level bases, see the Allow Closing, Minimize/Maximize, System Menu, and Title Bar properties.

**Note** This property setting is used with top-level bases only. Note that this setting is only a hint to the window system and might be ignored by some window managers.

**Floating**

The **Floating** attribute determines if the top-level base is a floating base (always on top). By default, this setting is False, indicating that the base is not a floating base. To create a floating base, set this property to True.

If you make a top-level base floating, you must set the **GROUP_LEADER** keyword to a valid widget ID when calling the generated procedure.

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

**Note** This property setting is used with top-level bases only.

**Grid Layout**

The **Grid Layout** attribute determines if the base will have a grid layout, in which all columns have the same width, or in which all rows have the same height. These are the possible values:

- False: Columns or rows will not be the same size. This is the default value.
- True: Column widths or row heights are taken from the largest child widget. If you set this property to True, you must also set the **Layout** property to Column or Row and the # of Rows/Columns property to more than 1.

In the generated *.pro file, this value is specified with the **GRID_LAYOUT** keyword to the widget creation routine.
Layout
The Layout attribute specifies how components are laid out in the base. These are the possible values:

- Bulletin: Indicates that you can position the widgets anywhere on the base. This is the default setting.
- Column: Indicates that widgets should be in columns. If you set this value, you should also set the #of Rows/Columns property and the Alignment property.
- Row: Indicates that widgets should be in rows. If you set this value, you should also set the #of Rows/Columns property and the Alignment property.

Note When using code generated by the IDL GUIBuilder on other non-Windows platforms, more consistent results are obtained by using a row or column layout for your bases instead of a bulletin board layout. By using a row or column layout, differences in the default spacing and decorations (e.g., beveling) of widgets on each platform can be avoided.

The number of child widgets placed in each column or row is calculated by dividing the number of created child widgets by the number of columns or rows specified (#of Rows/Columns). When one column or row is filled, a new one is started.

The width of each column or the height of the row is determined by the largest widget in that column or row. If you set the Grid Layout property to True, all columns or rows are the same size; they are the size of the largest widget.

If you set the Alignment property for the base, the contained widgets are their “natural” size. If you do not set the Alignment property for the base or the child widgets, all contained widgets will be sized to the width of the column or the height of the row.

For information on other properties that control the layout of contained widgets, see #of Rows/Columns, Alignment, Space, X Pad, and Y Pad.

In the generated *.pro file, this value is specified with the COLUMN or the ROW keyword to the widget creation routine.

Note When you create a radio button or checkbox, it is created in a base, and you can add more radio buttons or checkboxes to that base (the added widgets must all be of the same type). The base in which radio buttons and checkboxes are created has a column layout setting, and buttons you add will be lined up in a column format.
**Minimize/Maximize**

The Minimize/Maximize attribute determines if the top-level base can be resized, minimized, and maximized. By default, this value is set to True. To disable this behavior, set this property to False.

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

For information on other property settings that control aspects of top-level bases, see the Allow Closing, Allow Moving, System Menu, and Title Bar properties.

**Note** This property setting is used with top-level bases only. Note that this setting is only a hint to the window system and might be ignored by some window managers.

**Modal**

The Modal attribute determines if this top-level base is a modal dialog. By default, this value is set to False. To make the base a modal dialog, set this property to True.

If you set the Modal property to True, you cannot set the Scroll property, and you cannot define a menu for the top-level base. In addition, the Sensitive common property and the Visible base widget property are also disabled.

If you make a top-level base a modal dialog, you must set the GROUP_LEADER keyword to a valid widget ID in the generated procedure.

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

**Note** This property setting is used with top-level bases only.

**Scroll**

The Scroll attribute determines if the base widget will be support scrolling. By default, this property is set to False, and the base will not support scrolling. To give the widget scroll bars and allow for viewing portions of the widget contents that are not currently in the viewport area, set the Scroll property to True. In the IDL GUIBuilder, scroll bars on bases are live so that you can work on the entire virtual area of your application.

If you set the Modal property to True, you cannot set the Scroll property.

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

To set the size of the scrollable region, use the X Scroll and Y Scroll properties.
Note  For the Macintosh, if you set X Size or Y Size to a value less than 48, the base created with the Scroll property will be a minimum of 48x48. If you have not specified values for the X Size or Y Size property, the base will be set to a minimum of 66x66. If the base is resized, it will jump to the minimum size of 128x64.

Space  
The Space attribute specifies the number of pixels between the contained widgets (the children) in a column or row Layout. By default, this value is set to 3 pixels and that is the space between the contained widgets. Valid values for this property are 0 to n pixels. 

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

To set the space from the edge of the base, use the X Pad and Y Pad properties. For information on other properties that control the layout of contained widgets, see # of Rows/Columns, Alignment, and Layout.

Note  You cannot set this property on a base containing radio buttons or checkboxes.

System Menu  
The System Menu attribute determines if the system menu is displayed or suppressed on a top-level base. By default, this value is set to True, indicating that the system menu will be used. To suppress the menu, set this property to False.

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

For information on other property settings that control aspects of top-level bases, see the Allow Closing, Allow Moving, Minimize/Maximize, and Title Bar properties.

Note  This property setting is used with top-level bases only.

Title  
The Title attribute specifies the title of a top-level base. By default, this value is set to IDL, but you can change it to any string.

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

Note  This property setting is used with top-level bases only.
**Title Bar**

The Title Bar attribute determines if the title bar will be displayed. By default, this value is set to True, and the title bar is displayed. To suppress the display of the title bar, set this value to False.

For interfaces running on the Macintosh, you cannot suppress the title bar because only modal dialogs use a window without a title bar. Suppressing the title bar would be contrary to Macintosh Human Interface Guidelines and would create an immovable window.

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

For information on other property settings that control aspects of top-level bases, see the Allow Closing, Allow Moving, Minimize/Maximize, and System Menu properties.

**Note** This property setting is used with top-level bases only, and it is only a hint to the window system and might be ignored by some window managers.

**Visible**

The Visible attribute specifies whether to show or hide the base component and its descendants. Show, the default value, specifies to display the hierarchy when realized. The Hide value specifies that the hierarchy should not be displayed initially. This mapping operation applies only to base widgets.

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

**Note** If you set the Modal property to True, you cannot set this value.

**X Pad**

The X Pad attribute specifies the horizontal space (in pixels) between child widgets and the edges of rows or columns. By default, this value is set to 3 pixels, indicating that there are 3 pixels between the edge of the base and the contained widgets. Valid values for this property are 0 to \( n \) pixels.

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

To set the space between widgets, use the Space property. For information on other properties that control the layout of contained widgets, see # of Rows/Columns, Alignment, Layout, and Y Pad.

**Note** You cannot set this property for a base that contains radio buttons or checkboxes. In the IDL GUIBuilder, a base is created when you add a radio button or checkbox to an interface, and you can add more radio buttons or
checkboxes to that base. When you add the buttons, they are lined up in a column format.

**X Scroll**

The X Scroll attribute specifies the width in pixels of the base area, which includes the exposed as well as the virtual area. There is no default value set, but you can set this value to any number of pixels from 0 to n. To add scroll bars to the base, use the Scroll property, and to set the height of the scrollable base area, use the Y Scroll property.

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

**Note** To set the width of the displayed widget, use the X Size common property.

**Y Pad**

The Y Pad attribute specifies the vertical space (in pixels) between child components and the edge of the base in a row or column Layout. By default, this value is set to 3 pixels, indicating that there are 3 pixels between the edge of the base and the contained widgets. Valid values for this property are 0 to n pixels.

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

To set the space between widgets, use the Space property. For information on other properties that control the layout of contained widgets, see # of Rows/Columns, Alignment, Layout, and X Pad.

**Note** You cannot set this property on a base containing radio buttons or checkboxes. In the IDL GUIBuilder, a base is created when you add a radio button or checkbox to an interface, and you can add more radio buttons or checkboxes to that base.

**Y Scroll**

The Y Scroll attribute specifies the height in pixels of the base area, which includes the exposed as well as the virtual area. There is no default value set, but you can set this value to any number of pixels from 0 to n.

To add scroll bars to the base, use the Scroll property, and to set the width of the base area, use the X Scroll property.

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

**Note** To set the height of the displayed widget, use the Y Size common property.
Base Widget Properties

For base widgets, you can set common event properties and base-specific event properties. By default, event values are not set. For a list of events common to all widgets, see “Common Events” on page 199.

The following is a list of event properties specific to base widgets:

**OnFocus**
The OnFocus value is the routine name that is called when the keyboard focus of the base changes. In the generated *_eventcb.pro file, the event calling sequence looks like this:

```
PRO <RoutineName>, Event
```

*RoutineName* is the name of the event procedure you specify. *Event* is the returned event structure, which is returned when the keyboard focus changes and is of the following type:

```
{ WIDGET_KBRD_FOCUS, ID:0L, TOP:0L, HANDLER:0L, ENTER:0 }
```

ENTER returns 1 if the base is gaining the keyboard focus, and returns 0 if the base is losing the keyboard focus.

**OnKillRequest**
The OnKillRequest value is the routine that is called when the user attempts to kill the top-level base widget. In the generated *_eventcb.pro file, the event calling sequence looks like this:

```
PRO <RoutineName>, Event
```

*RoutineName* is the name of the event procedure you specify. *Event* is the returned event structure, which is returned when a user tries to destroy the widget using the window manager and is of the following type:

```
{ WIDGET_KILL_REQUEST, ID:0L, TOP:0L, HANDLER:0L }
```

Note that this event structure contains the standard three fields that all widgets contain.

**Note** This event procedure is valid for top-level bases only.

**OnSizeChange**
The OnSizeChange value is the name the routine that is called when the top-level base has been resize. In the generated *_eventcb.pro file, the event calling sequence looks like this:

```
PRO <RoutineName>, Event
```

RoutineName is the name of the event procedure you specify. Event is the returned event structure, which is returned when the top-level base is resized by the user and is of the following type:

{ WIDGET_BASE, ID:0L, TOP:0L, HANDLER:0L, X:0, Y:0 }

The X and Y fields return the new width of the base, not including any frame provided by the window manager.

Note This event procedure is valid for top-level bases only.
Button Widget Properties

In IDL, a button widget can be a button (push button), radio button, or checkbox.

A push button is activated by a single-click. Push buttons can be of any size. You can set the Menu property to yes for a button widget, and then it can contain a pull-down menu. When you do so, the Label is enclosed in a box to indicate that the button is a menu button.

Radio buttons have two states, set and unset, and they belong to a group that allows only one radio button selection for that group. The group is defined as all buttons contained in the same exclusive base widget. When a radio button in a base (in a group) is selected, any other button selection in that base is cleared. When you create a radio button in the IDL GUIBuilder, it is created in an exclusive base widget, and you can add only radio buttons to that base.

Checkboxes have two states, set and unset, and they are grouped in a non-exclusive base widget. This base allow for any number of checkboxes to be set at one time, and you can use single checkboxes in your interface. When you create a checkbox in the IDL GUIBuilder, it is created in an non-exclusive widget base, and you can add only checkboxes to this base.

When programming in IDL, you create push buttons, radio buttons, and checkboxes using the WIDGET_BUTTON function, which is described in "WIDGET_BUTTON" on page 1222 of the IDL Reference Guide.

Note The bases in which radio buttons and checkboxes are created have the Layout attribute set to column so when you add more widgets they are lined up appropriately.

Creating Multiple Radio Buttons or Checkboxes

To create several radio buttons or checkboxes in a base widget:

1. Click on the radio button or checkbox tool, and click on the location to add the button. This action creates a base with radio one button or checkbox in it.

2. Click on the radio button or checkbox tool, and click in the radio button or checkbox base area you just created. This action adds a radio button or checkbox to the base.

When you drop a button in a exclusive or non-exclusive base, the added buttons line up in columns; by default, these exclusive and non-exclusive bases have their Layout property set to Column.

3. Repeat step 2 until you have the desired number of buttons.
4. If you want to change the layout of the checkboxes or radio buttons, you can open the Properties dialog and set the Layout common property for the base widget to Row or Bulletin.

5. To set the properties for each button in the base, open the Properties dialog, click the push pin button to keep it on top, then click on each radio button or checkbox to set their individual properties.

**Button, Radio Button, and Checkbox Widget Attributes**

For button widgets, you can set common attributes and button-specific attributes. For a list of common attributes, see “Common Attributes” on page 196. The following is a list of button widget attributes, which apply to push buttons, radio buttons, and/or checkboxes:

**Alignment**

The Alignment attribute specifies how the text label is aligned in the button widget. These are the possible alignment values:

- Center: The label text is centered. This is the default value.
- Left: The label text is left-justified.
- Right: The label text is right-justified.

In the generated *.pro file, this value is specified by the ALIGN_CENTER, the ALIGN_LEFT, or the ALIGN_RIGHT keyword to the widget creation routine.

**Bitmap**

The Bitmap attribute allows you to select a bitmap to be displayed in the push button, and it allows you to access the Bitmap Editor to create or modify a bitmap file (*.bmp file). This value applies only to buttons (not to radio buttons or checkboxes).

To set this value:

Set the Type value to Bitmap, then the Bitmap attribute displays in the Properties dialog. When the button type is “Bitmap”, you can set the Bitmap attribute to the path and name of the bmp file.

When you click on the arrow in the Bitmap attribute Value field, you can choose from the following options:

- Select Bitmap: Launches an Open dialog that you can use to locate and select the existing *.bmp file to be placed in the button.
• **Edit Bitmap**: Launches an Open dialog that you can use to locate and select the existing *.bmp file to be opened in the Bitmap Editor. You can modify the bitmap and save it. The bitmap is then displayed in the button.

• **New Bitmap**: Opens the Bitmap Editor which you can use to create and save a bitmap. When you save the new bitmap, it is displayed in the button.

In the generated *.pro file, this value is specified with the VALUE and Bitmap keyword to the widget creation routine.

For information on using the Bitmap Editor, see “Using the Bitmap Editor” on page 174.

**Label**

The Label attribute specifies the text label for a button. If you set the Type attribute to Bitmap (for push buttons only), this value is not displayed. For radio buttons and checkboxes, the label value is the text string displayed next to the button. By default, this value is set to Button, and you can change it to any string.

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

**No Release**

The No Release attribute enables and disables the dispatching of button release events for radio buttons and checkboxes. Normal buttons do not generate events when released, but radio buttons and checkboxes can return separate events for the select and release actions. These are the possible values:

• **True**: The release event is not returned; only the select event is returned. This is the default setting.

• **False**: Both the release and select events are returned.

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

**Note** The No Release property is for radio buttons and checkboxes only.

**Type**

The Type attribute specifies if a push button is a plain push button, a menu button, or a bitmap button. This attribute applies only to push buttons (not to radio buttons or checkboxes). These are the possible values:

• **Push**: The button widget is a plain push button. This is the default value.
• Menu: The button contains a menu. After you select this value, you can right-click on the button widget, choose Edit Menu, and define a menu to display, using the Menu Editor.

• Bitmap: The button displays a bitmap, which you would use to create a toolbar for example. If you change the Type value to Bitmap, the Bitmap property is displayed and you can select, modify, or create a bitmap to display on the button.

In the generated *.pro file, this value is specified with the MENU or VALUE keywords to the widget creation routine.

Button, Radio Button, and Checkbox Widget Events

For button widgets, you can set common event properties and button-specific event properties. By default, event values are not set. For a list of events common to all widgets, see “Common Events” on page 199.

The following is the event property specific to button widgets; it applies to push buttons, radio buttons, and checkboxes:

OnButtonPress

The OnButtonPress value is the routine that is called when the button is pressed, or when a button is released for a radio button or checkbox button. In the generated *_.eventcb.pro file, the event calling sequence looks like this:

PRO <RoutineName>, Event

RoutineName is the name of the event procedure you specify. Event is the returned event structure, which is of the following type:

{ WIDGET_BUTTON, ID:0L, TOP:0L, HANDLER:0L, SELECT:0 }

SELECT is set to 1 if the button was set, and 0 if released. Push buttons do not generate events when released, so SELECT will always be 1 for a push button. However, radio buttons and checkboxes are toggle buttons, and thus return separate events for the set and the release actions. To control whether or not release events are returned, set the No Release property, which is described above.
Text Widget Properties

Use text widgets to display text, and optionally, use them to accept textual input from users. The text widgets can have one or more lines, and if necessary, the widget can contain scroll bars to allow for viewing longer text.

When programming in IDL, you create text widgets using the WIDGET_TEXT function, which is described in “WIDGET_TEXT” on page 1309 of the IDL Reference Guide.

Note Use text widgets for displaying large amounts of text, or when you want the user to be able to edit the text. Use label widgets to display single-line labels that the user cannot edit.

Text Widget Attributes

For text widgets, you can set common attributes and text-specific attributes. For a list of common attributes, see “Common Attributes” on page 196. The following are the attributes specific to text widgets:

Editable

The Editable attribute determines if the text widget is editable or not. By default, this value is set to False, which means the text widget is not editable. To make the text widget editable, set this value to True.

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

Height

The Height attribute specifies the height of the text widget in text lines. Valid values for this attribute are 1 to \( n \). The default value is 1, or one text line.

Note that the physical height of the text widget depends on the value of the Height attribute and on the size of the font used. The default font size is used, unless you modify your generated code to use a different font, and the default font size is platform specific.

In the generated *.pro file, this value is specified by the YSIZE keyword to the widget creation routine.

Initial Value

The Initial Value attribute specifies the initial array of values that are placed in the text widget. You can enter either a string or an array of strings.

To enter more than one string in the Value field:
Type in a string, then press Control+Enter (at the end of each line). This action moves you to the next line. When you have entered the strings you want, press Enter to set the values.

In the generated *.pro file, this value is specified by the VALUE keyword to the widget creation routine.

**Note** Variables returned by the GET_VALUE keyword to WIDGET_CONTROL are always string arrays, even if a scalar string is specified in the call to WIDGET_TEXT.

**Scroll**

The Scroll attribute determines if the text widget displays scroll bars. By default, this value is set to False, which indicates that no scroll bars will be displayed. To have the text widget display scroll bars, set this value to True.

In the generated *.pro file, this value is specified by the SCROLL keyword to the widget creation routine.

**Width**

The Width attribute specifies the width of the text widget in characters. Valid values for this attribute are 0 to $n$. By default, Width is set to 0, which indicates that default IDL sizing should be used when, as long as default Component Sizing is also set.

Note that the physical width of the text widget depends on the value of the Width attribute and on the size of the font used. The default font size varies according to your windowing system. On Windows and Macintosh, the default size is roughly 20 characters. On Motif, the default size depends on the system default.

In the generated *.pro code, this value is specified with the XSIZE keyword.

**Word Wrapping**

The Word Wrapping attribute determines whether a scrolling or multi-line text widgets should automatically break lines between words to keep the text from extending past the right edge of the text display area. By default this value is set to False, and carriage returns are not automatically entered; the value of the text widget will remain a single-element array unless. To have the text widget enter carriage returns at the end of lines, change this value to True.

In the generated *.pro code, this value is specified with the WRAP keyword.
Text Widget Events

For text widgets, you can set common event properties and text-specific event properties. By default, event values are not set. For a list of events common to all widgets, see “Common Events” on page 199.

You can set the following event values for text widgets:

OnDelete
The OnDelete value is the routine that is called when text is deleted from the text widget. To set this event value, you must set the Editable attribute to True.

In the generated *_eventcb.pro file, the calling sequence looks like this:

```idl
PRO <RoutineName>, Event
```

`RoutineName` is the name of the event procedure you specify. `Event` is the returned event structure, which is returned when any amount of text is deleted from a text widget. The event structure is of the following type:

```idl
{ WIDGET_TEXT_DEL, ID:0L, TOP:0L, HANDLER:0L, TYPE:2, OFFSET:0L, LENGTH:0L }
```

OFFSET is the (zero-based) character position of the first character to be deleted, and it is also the insertion position that will result when the characters have been deleted. LENGTH gives the number of characters deleted, where 0 (zero) indicates that no characters were deleted.

OnFocus
The OnFocus value is the routine that is called when the keyboard focus changes. In the generated *_eventcb.pro event code, the calling sequence looks like this:

```idl
PRO <RoutineName>, Event
```

`RoutineName` is the name of the event procedure you specify. `Event` is the returned structure, which is of the following type:

```idl
{ WIDGET_KBD_FOCUS, ID:0L, TOP:0L, HANDLER:0L, ENTER:0 }
```

ENTER returns 1 if the text widget is gaining the keyboard focus, or 0 if the text widget is losing the keyboard focus.

OnInsertCh
The OnInsertCh value is the routine that is called when a single character is inserted in the widget. To set this event value, you must set the Editable attribute to True.

In the generated *_eventcb.pro file, the calling sequence looks like this:

```idl
PRO <RoutineName>, Event
```
**RoutineName** is the name of the event procedure you specify. **Event** is the returned event structure, which is returned when a single character is typed or pasted into a text widget by a user. The event structure is of the following type:

```
{ WIDGET_TEXT_CH, ID:0L, TOP:0L, HANDLER:0L, TYPE:0, OFFSET:0L,
  CH:0B }
```

OFFSET is the (zero-based) insertion position that will result after the character is inserted. CH is the ASCII value of the character.

**OnInsertString**

The **OnInsertString** value is the routine that is called when a text string is inserted in the text widget. To set this event value, you must set the Editable attribute to True.

In the generated \_\_eventcb.pro file, the calling sequence looks like this:

```
PRO <RoutineName>, Event
```

**RoutineName** is the name of the event procedure you specify. **Event** is the returned event structure, which is of the following type, which is returned when multiple characters are inserted in to text widget:

```
{ WIDGET_TEXT_STR, ID:0L, TOP:0L, HANDLER:0L, TYPE:1, OFFSET:0L,
  STR:’’ }
```

OFFSET is the (zero-based) insertion position that will result after the text is inserted. STR is the string to be inserted.

**OnTextSelect**

The **OnTextSelect** value is the routine that is called when text is selected in the text widget. To set this event value, you must also set the Editable attribute to True.

In the generated \_\_eventcb.pro file, the calling sequence looks like this:

```
PRO <RoutineName>, Event
```

**RoutineName** is the name of the event procedure you specify. **Event** is the returned event structure, which is returned when an area of text is selected. The event structure is of the following type:

```
{ WIDGET_TEXT_SEL, ID:0L, TOP:0L, HANDLER:0L, TYPE:3, OFFSET:0L,
  LENGTH:0L }
```

This event announces a change in the insertion point. OFFSET is the (zero-based) character position of the first character selected, which can also be the insertion position. LENGTH gives the number of characters involved, where zero indicates that no characters are selected.

**Note** Text insertion, text deletion, or any change in the current insertion point causes any current selection to be lost. In such cases, the loss of selection is
implied by the text event reporting the insert, delete, or movement event, and a separate zero length selection event is not sent.
Label Widget Properties

Label widgets display static text. They are similar to single-line text widgets, but they are optimized for small labeling purposes.

There are not label widget-specific event properties.

When programming in IDL, you create label using the WIDGET_LABEL function, which is described in “WIDGET_LABEL” on page 1280 of the IDL Reference Guide.

Note Use label widgets to display single-line labels that you do not want the user to be able to edit. Use text widgets for displaying larger amounts of text, or text that you want the user to be able to edit.

Label Widget Attributes

For label widgets, you can set common attributes and label-specific attributes. For a list of common attributes, see “Common Attributes” on page 196. These are the label widget attributes:

Alignment

The Alignment attribute specifies how label Text is aligned. These are the possible values:

- Left: The text is left-justified. This is the default value.
- Center: The text is centered.
- Right: The text is right-justified.

In the generated *.pro file, this value is specified with the ALIGN_CENTER, the ALIGN_RIGHT, or the ALIGN_LEFT keyword to the widget creation routine.

Text

The Text attribute specifies the text string that is displayed in the label widget. By default, this value is set to Label, and you can set it to any string.

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

Label Widget Events

There are no events specific to Label widgets. For a list of the common widget events, see “Common Events” on page 199.
Slider Widget Properties

Horizontal or vertical slider widgets allow for the selection of a value within a range of possible integer values. A slider widget is a rectangular region representing a range of values, with a sliding pointer inside that indicates or selects the current value. This sliding pointer can be manipulated by the user dragging it with the mouse, or within IDL code.

When programming in IDL, you create horizontal or vertical slider widgets using the WIDGET_SLIDER function, which is described in “WIDGET_SLIDER” on page 1291 of the IDL Reference Guide.

Horizontal and Vertical Slider Widget Attributes

For slider widgets, you can set common attributes and slider-specific attributes. For a list of common attributes, see “Common Attributes” on page 196. The following is a list of slider attributes:

Maximum Value

The Maximum Value attribute specifies the maximum range value for the slider. The default value is 100, but you can set this property to any integer. This value works with the Minimum Value property.

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

Minimum Value

The Minimum Value attribute specifies the minimum range value of the slider. The default value is 0, but you can set this property to any integer. This property works with the Maximum Value property.

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

Position

The Position attribute specifies the initial value position of the slider. By default this is set to 0, so the initial position will be at 0. You can set this value to any integer within the range of the Maximum Value and Minimum Value attribute settings.

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

Suppress Value

The Suppress Value attribute controls the display of the current slider value. Sliders work only with integer units. You can use this property to suppress the display.
actual value of a slider so that a program can present the user with a slider that seems to work in other units (such as floating-point) or with a non-linear scale. By default, this value is set to False, indicating that the current slider values, in integer units, should be displayed. To suppress the display of the current values, set this property value to True.

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

**Title**

The Title attribute specifies the label or title that is associate with the slider widget. By default, this is not set; it is an empty string. You can set the title to any string.

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

**Horizontal and Vertical Slider Widget Events**

For slider widgets, you can set common event properties and slider-specific event properties. By default, event values are not set. For a list of events common to all widgets, see “Common Events” on page 199.

This is the event property specific to slider widgets:

**OnChangeValue**

The OnChangeValue specifies the routine that is called when the value of the slider is changed. When you set this event value, the calling sequence looks like this in the generated *_eventcb.pro file:

```idl
PRO <RoutineName>, Event
```

*RoutineName* is the name of the event procedure you specify. *Event* is the returned event structure, which is returned when a slider is moved. The event structure is of the following type:

```idl
{ WIDGET_SLIDER, ID:0L, TOP:0L, HANDLER:0L, VALUE:0L, DRAG:0 }`
```

*VALUE* returns the new value of the slider. *DRAG* returns integer 1 if the slider event was generated as part of a drag operation, or zero if the event was generated when the user had finished positioning the slider. Note that the slider widget only generates events during the drag operation if the DRAG keyword is set, and if the application is running on Motif. That is, in most cases, DRAG will return zero.
Droplist Widget Properties

Droplist widgets display a single entry from a list of possible choices. To choose from the list, click the droplist, then click on the item in the list. On Motif operating systems, the droplist widget looks like a button, which when clicked displays the drop-down list.

When programming in IDL, you create droplist widgets using the WIDGET_DROPLIST function, which is described in “WIDGET_DROPLIST” on page 1262 of the IDL Reference Guide.

Droplist Widget Attributes

For droplist widgets, you can set common attributes and droplist-specific attributes. For a list of common attributes, see “Common Attributes” on page 196. These are the droplist attributes:

Initial Value

The Initial Value attribute specifies the initial list of values that are placed in the droplist widget. The initial value of a droplist can be a scalar string, or it can be a list of strings. By default, this value is not set, and the droplist is empty.

To enter more than one string in the Value field:

Type in a string, then press Control+Enter (at the end of each line). This action moves you to the next line. When you have entered as many strings as you want, press Enter to set the values.

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

Title

The Title attribute specifies the title string, or label, for the droplist. This value can be any string. By default, this value is set to NULL.

In the generated *.pro file, this value is specified by the TITLE keyword to the widget creation routine.

Droplist Widget Events

For droplist widgets, you can set common event properties and droplist-specific event properties. By default, event values are not set. For a list of events common to all widgets, see “Common Events” on page 199.

This is the event property specific to droplist widgets:
**OnSelectValue**

The OnSelectValue specifies the routine that is called when a droplist item is selected. When a user selects an item from a droplist, the widget deselects the previously selected item, changes the visible item on the droplist, and generates an event.

When you set this event value, the calling sequence looks like this in the generated \_*_eventcb.pro file:

```idl
PRO <RoutineName>, Event
```

*RoutineName* is the name of the event procedure you specify. *Event* is the returned event structure, which is returned when a user selects an item from a droplist and is of the following type:

```
{ WIDGET_DROPLIST, ID:0L, TOP:0L, HANDLER:0L, INDEX:0L }
```

*INDEX* returns the index of the selected item. This value can be used to index the array of names originally used to set the widget’s value.

**Note** On some platforms, when a droplist widget contains only one item and the user selects the again, the action does not generate an event. Events are always generated on selection actions if the list contains multiple items.
Listbox Widget Properties

The listbox displays a list of text items from which a user can select, by clicking on them. The listboxes have vertical scroll bars to allow viewing of a long list of items.

When programming in IDL, you create listbox widgets using the WIDGET_LIST function, which is described in “WIDGET_LIST” on page 1285 of the IDL Reference Guide.

Listbox Widget Attributes

For listbox widgets, you can set common attributes and listbox-specific attributes. For a list of common attributes, see “Common Attributes” on page 196. These are the listbox widget attributes:

Height

The Height attribute specifies the height of the listbox based on the number of lines that are visible. The possible values for the attribute are 1 to n. By default, Height is set to 1, which indicates the default size of one line will be used.

Note that the final size of the widget may be adjusted to include space for scroll bars, which are not always visible, so the listbox might be slightly larger than specified.

In the generated *.pro file, this value specified with the YSIZE keyword to the widget creation routine.

Initial Value

The Initial Value attribute specifies the initial list of values that are placed in the list widget. By default, the list is empty, but you can set this value to a scalar string or a list of strings. List widgets are sized based on the length (in characters) of the longest item specified in the array of values.

To enter more than one string in the Value field:

Type in a string, then press Control+Enter (at the end of each line). This action moves you to the next line. When you have entered as many strings as you want, press Enter to set the values.

In the generated *.pro file, this value is specified by the VALUE keyword to the widget creation routine.

Multiple

The Multiple attribute determines if the user can select multiple list items. By default, the setting is False, which allows for only one selection. To enable
multiple list item selection, set this value to True. Multiple selections are handled using the method appropriate to the platform the application is running on.

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

**Width**

The Width attribute specifies the width of the listbox in characters. The possible values for the attribute are 0 to n. By default, Width is set to 0, which indicates that default sizing will be used, as long as the Component Sizing attribute is set to default.

By default, IDL sizes widgets to fit the situation. However, if the desired effect is not produced, use explicit Component Sizing with the Width property to set your own sizing. The final size of the widget may be adjusted to include space for the scroll bar, which is not always visible, so your widget may be slightly larger than specified.

In the generated *.pro file, this value specified with the XSIZE keyword to the widget creation routine.

**Listbox Widget Events**

For listbox widgets, you can set common event properties and listbox-specific event properties. By default, event values are not set. For a list of events common to all widgets, see “Common Events” on page 199.

The following is the event property specific to listbox widgets:

**OnSelectValue**

The OnSelectValue specifies a valid IDL routine name that is called when a list item is selected. When a user clicks on an item in the listbox to select the item, an event is generated.

When you set this event value, the calling sequence looks like this in the generated *_eventcb.pro file:

```idl
PRO <RoutineName>, Event
```

*RoutineName* is the name of the event procedure you specify. *Event* is the returned event structure, which is of the following type:

```idl
{ WIDGET_LIST, ID:0L, TOP:0L, HANDLER:0L, INDEX:0L, CLICKS:0L }
```

The first three fields are the standard fields found in every widget event. INDEX returns the index of the selected item. This index can be used to subscript the array of names originally used to set the widget’s value. CLICKS returns either 1 or 2, depending on how the list item was selected. If the list item is double-clicked, CLICKS is set to 2.
Note If you are writing a widget application that requires the user to double-click on a list widget, you will need to handle two events. The CLICKS field will return a 1 on the first click and a 2 on the second click.
Draw Widget Properties

Draw widgets are rectangular regions that IDL treats as standard graphics windows. Use draw widgets to display either IDL Direct graphics or IDL Object graphics, depending on the value of the Graphics Type property. You can direct any graphical output that can be produced by IDL to one of these widgets, either by using the WSET function or by using the object reference of a draw widget's IDLgrWindow object.

Draw widgets can contain scroll bars that allow for viewing of a graphical region larger than the area containing the widget.

When programming in IDL, you create draw area widgets using the WIDGET_DRAW function, which is described in “WIDGET_DRAW” on page 1252 of the IDL Reference Guide.

Draw Area Widget Attributes

For a draw area widget, you can set common attributes and draw area-specific attributes. For a list of common attributes, see “Common Attributes” on page 196. These are the draw area-specific attributes:

Color Model

The Color Model attribute specifies the color model that should be used for displaying information on the draw widget. This property value is used only when the Graphics Type property is set to Object, for IDL Object Graphics. These are the possible values for the Color Model attribute:

- Index: The draw widget’s associated IDLgrWindow object uses indexed color. This is the default value.
- RGB: The RGB color model is used.

In the generated *.pro file, this value is specified by the COLOR_MODEL keyword to the widget creation routine.

For information on using indexed color in Object Graphics window objects, see “Working with Color” in Chapter 5 of Object Graphics.

Colors

The Colors attribute specifies the number of colors that the drawable should attempt to use from the system color table. This property is only valid with the Graphics Type property set to Direct, for IDL Direct Graphics. By default, the Color attribute is set to 0, which indicates that IDL will attempt to get all available colors. That is, all or most of the available color indices are allocated, based on the window system in use. You can set the Colors attribute to any integer, but most values will be in the range of -256 < n < 256.
This property has effect only if it is supplied when the first IDL graphics window is created. To use monochrome windows on a color display, set the Colors property to 2 for the first window. One color table is maintained for all running IDL windows.

In the generated *.pro file, this value is specified by the COLORS keyword to the widget creation routine.

**Graphics Type**

The Graphics Type attribute specifies the type of graphics that the draw widget will support. These are the possible values:

- Direct: The draw widget will display Direct Graphics. This is the default value. The Colors property is used only when Graphics Type is set to Direct.

- Object: The draw widget will display IDL Object Graphics. The Color Model and Renderer properties are used only when the Graphics Type is set to Object.

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

**Renderer**

The Renderer attribute specifies which graphics renderer to use with IDL Object Graphics. That is, for this property to be used, the Graphics Type property should be set to Object. These are the possible values for the Renderer attribute:

- OpenGL: The platform's native OpenGL renderer is used when drawing objects within the window. If your platform does not have a native OpenGL implementation, IDL's software implementation is used as the renderer. This value is set by default.

- Software: IDL's software implementation is used when drawing objects within the window.

In the generated *.pro file, this value is specified by the RENDERER keyword to the widget creation routine.

For more information, see “Hardware vs. Software Rendering” in Chapter 13 of Object Graphics.

**Note** The renderer selection can also affect the maximum size of a draw widget.
Retain
The Retain attribute specifies how backing store is performed in the draw area. These are the possible values:

- None: There is no backing store. When the Retain property is set to None, you should track OnExpose events so that you can handle the redrawing of the screen. This is the default value.
- System: The server or window system should provide backing store.
- IDL Pixmap: IDL should provide backing store.

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

For information on the use of the Retain property with Direct Graphics, see “Backing Store” on page 144. For more information on this property with IDL Object Graphics, see “IDLgrWindow::Init” in Chapter 17 of Object Graphics.

Scroll
The Scroll attribute specifies if the draw area widget will support scrolling, and will have scroll bars. By default, this value is set to False, which indicates there are no scroll bars. To display scroll bars, and enable scrolling, set this value to True. If you do so, set the size of the scrollable area with the X Scroll and Y Scroll properties.

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

X Scroll
The X Scroll attribute specifies the width in pixels of the drawing area. This width includes the exposed and virtual area. By default, this value is not set. You can set X Scroll to any width from 0 to n. If you set this value, also set the Scroll and Y Scroll property values.

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

Note To set the width of the displayed widget, use the X Size common property.

Y Scroll
The Y Scroll attribute specifies the height in pixels of the drawing area. This height includes the exposed and virtual area. By default, this value is not set. You can set Y Scroll to any height in pixels from 0 to n. If you set this value, also set the Scroll and X Scroll properties.
In the generated *.pro file, this value is specified with the YSIZE keyword to the widget creation routine.

**Note** To set the height of the displayed widget, use the Y Size common property.

**Draw Area Widget Events**

For draw area widgets, you can set common event properties and draw area-specific event properties. By default, event values are not set. For a list of events common to all widgets, see “Common Events” on page 199.

These are the draw area event properties:

**OnButton**

The OnButton value is the routine that is called when a mouse button event is detected. In the generated *._eventcb.pro file, the calling sequence looks like this:

```
PRO <RoutineName>, Event

RoutineName is the name of the event procedure you specify. Event is the returned event structure, which is of the following type:

{ WIDGET_DRAW, ID:0L, TOP:0L, HANDLER:0L, TYPE: 0, X:0, Y:0, PRESS:0B, RELEASE:0B, CLICKS:0 }
```

Note that this is the same event structure returned for all draw area events; OnButton, OnExpose, OnMotion, and OnViewportMoved events all return the same structure. Therefore the following paragraphs describe all these events.

TYPE returns a value that describes the type of draw widget interaction that generated an event. If there is a button press, it returns 0, and if there is a button release, it returns 1. If there is motion, it returns 2 (for an OnMotion event). If the viewport moved with the scroll bars, it returns 3 (for an OnViewportMoved event). If the visibility changes, it returns 4 (for an OnExpose event).

The X and Y fields give the device coordinates at which the event occurred, measured from the lower left corner of the drawing area.

PRESS and RELEASE are bitmasks in which the least significant bit represents the left-most mouse button. The corresponding bit of PRESS is set when a mouse button is pressed, and in RELEASE when the button is released. If the event is a motion event, both PRESS and RELEASE returns zero.

CLICKS returns either 1 or 2. If the time interval between button-press events is greater than the time interval for a double-click event for the system, the CLICKS field returns 1. If the time interval between two button-press events is less than the time interval for a double-click event for the platform, the CLICKS field returns 2.
**OnExpose**

The OnExpose value is the routine that is called when the visibility of any portion of the draw window (or viewport) changes or is exposed. In the generated *_eventcb.pro* file, the calling sequence looks like this:

```idl
PRO <RoutineName>, Event
```

*RoutineName* is the name of the event procedure you specify. *Event* is the returned event structure, which is of the following type:

```idl
{ WIDGET_DRAW, ID:0L, TOP:0L, HANDLER:0L, TYPE: 0, X:0, Y:0, 
  PRESS:0B, RELEASE:0B, CLICKS:0 }
```

Note that this is the same event structure returned for all draw area events; OnButton, OnExpose, OnMotion, and OnViewportMoved events all return the same structure. For information on this structure, see OnButton (above).

**OnMotion**

The OnMotion value is the routine that is called when a mouse motion event is detected. In the generated *_eventcb.pro* file, the calling sequence looks like this:

```idl
PRO <RoutineName>, Event
```

*RoutineName* is the name of the event procedure you specify. *Event* is the returned event structure, which is of the following type:

```idl
{ WIDGET_DRAW, ID:0L, TOP:0L, HANDLER:0L, TYPE: 0, X:0, Y:0, 
  PRESS:0B, RELEASE:0B, CLICKS:0 }
```

Note that this is the same event structure returned for all draw area events; OnButton, OnExpose, OnMotion, and OnViewportMoved events all return the same structure. For information on this structure, see OnButton (above).

**OnViewportMoved**

The OnViewportMoved value is the routine that is called when the viewport of a scrolling draw widget is moved, using the scroll bars. In the generated *_eventcb.pro* file, the calling sequence looks like this:

```idl
PRO <RoutineName>, Event
```

*RoutineName* is the name of the event procedure you specify. *Event* is the returned event structure, which is of the following type:

```idl
{ WIDGET_DRAW, ID:0L, TOP:0L, HANDLER:0L, TYPE: 0, X:0, Y:0, 
  PRESS:0B, RELEASE:0B, CLICKS:0 }
```

Note that this is the same event structure returned for all draw area events; OnButton, OnExpose, OnMotion, and OnViewportMoved events all return the same structure. For information on this structure, see OnButton (above).
Table Widget Properties

Table widgets display data and allow for data editing by the user. Tables can have one or more rows and one or more columns.

When programming in IDL, you create table widgets using the WIDGET_TABLE function, which is described in “WIDGET_TABLE” on page 1298 of the IDL Reference Guide.

Table Widget Attributes

For table widgets, you can set common attributes and table-specific attributes. For a list of common attributes, see “Common Attributes” on page 196. These are the table widget-specific attributes:

Alignment

The Alignment attribute specifies how the text is aligned in the cells. These are the possible values:

- Left: The text is left-justified. This is the default value.
- Right: The text is right-justified.
- Center: The text is centered.

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

Column Labels

The Column Labels attribute specifies the labels for the table columns. By default, this value is set to empty strings, but you can set it to any set of strings. To set the labels for table rows, use the Row Labels property.

To enter more than one string in the Value field:

Type in a string, then press Control+Enter (at the end of each line). This action moves you to the next line, or the next label for a column. When you have entered as many labels as you want, press Enter to set the values.

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

Display Headers

The Display Headers attribute determines if the table headings, the row and column labels, are displayed. By default, this value is set to True, indicating that table heading should be displayed. To disable the display of table headings, set this value to False.
In the generated *.pro file, the False value is specified with the NO_HEADERS keyword to the widget creation routine.

**Editable**

The Editable attribute determines if the table widget is editable or not. By default, this value is set to False, which means the text widget is not editable, and the text is read-only. To make the text widget editable, set this value to True.

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

**Number of Columns**

The Number of Columns attribute specifies the number of columns in the table widget. This value sets the full, virtual width of the table. By default, it is set to 6.

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

**Note** To have a scrollable table, set the Scroll property to True. Then, to specify the visible size of the table, set the Viewport Columns property.

**Number of Rows**

The Number of Rows attribute specifies the number of rows in the table widget. This value sets the full, virtual height of the table. By default, it is set to 6.

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

**Note** To have a scrollable table, set the Scroll property to True. Then, to specify the visible size of the table, set the Viewport Columns property.

**Resize Columns**

The Resize Columns attribute determines if this user can resize table columns. By default, this value is set to True, indicating that the user can resize the columns. To specify that the columns of the table are not resizeable by the user, set this value to False.

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

**Note** If you set the Display Headers property to False, the ability to resize the columns is automatically disabled.

**Row/Column Major**

The Row/Column Major attribute specifies how data is transferred to the table widget, either by Row or by Column. By default, this value is set to Row,
indicating that the data should be read into the table as if each element of the vector is a structure containing one row’s data. To specify that the data should be read into the table as if each element of the vector is a structure containing one column’s data, set this value to Column. Note that for either setting to work properly the structures must all be of the same type, and must have one field for each column or row in the table.

In the generated *.pro file, this value is specified with the ROW_MAJOR or the COLUMN_MAJOR keyword to the widget creation routine.

**Row Labels**

The Row Labels attribute specifies the labels for the table rows. By default, this value is set to empty strings, but you can set it to any set of strings. To set the labels for table columns, use the Column Labels property.

To enter more than one string in the Value field:

Type in a string, then press Control+Enter (at the end of each line). This action moves you to the next line, or the next label for a row. When you have entered as many labels as you want, press Enter to set the values.

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

*IDL; implementation always shows scrollbars

**Scroll**

The Scroll attribute determines if the table widget has scroll bars. By default, this value is set to False, indicating that the table will have no scroll bars. To enable scroll bars, set this value to True. If you set this value to True, you can set the size of the scrollable region with the Viewport Rows and Viewport Columns properties.

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

**Viewport Columns**

The Viewport Columns attribute specifies the number of columns that should be visible in the scroll area of the table widget. By default, this value is set to 6.

If you first set the Scroll property to True, you can then set this value to any size from 0 to n columns within the limits of your full table size. The full table size, or virtual width in columns, is set with the Number of Columns property.

This property is used only when the Component Sizing property is set to Default. If you set the Component Sizing property to Explicit, either through the
Properties dialog or by dragging the component to specific size, the Viewport Columns property is ignored, and the X Size and the Y Size properties are used.

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

Viewport Rows
The Viewport Rows attribute specifies the number of rows that should be visible in the scroll area of the table widget. By default, this value is set to 6.

If you first set the Scroll property to True, you can then set this value to any size from 0 to n rows, within the limits of your full table size. The full table size, or virtual height in rows, is set with the Number of Rows property.

This property is used only when the Component Sizing property is set to Default. If you set the Component Sizing property to Explicit, either through the Properties dialog or by dragging the component to specific size, the Viewport Rows property is ignored, and the X Size and the Y Size properties are used.

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

Table Widget Events
For table widgets, you can set common event properties and table-specific event properties. By default, event values are not set. For a list of events common to all widgets, see “Common Events” on page 199.

These are the table widget-specific event properties:

OnCellSelect
The OnCellSelect value is the routine that is called when cells are selected in the table. When you set this value, the calling sequence looks like this in the generated *_eventcb.pro file:

```
PRO <RoutineName>, Event
```

`RoutineName` is the name of the event procedure you specify. `Event` is the returned event structure, which is returned when range of cells is selected or deselected and is of the following type:

```
{ WIDGET_TABLE_CELL_SEL, ID:0L, TOP:0L, HANDLER:0L, TYPE:4,
  SEL_LEFT:0L, SEL_TOP:0L, SEL_RIGHT:0L, SEL_BOTTOM:0L }
```

The range of cells selected is given by the zero-based indices into the table specified by the SEL_LEFT, SEL_TOP, SEL_RIGHT, and SEL_BOTTOM fields. When cells are deselected, either by changing the selection or by clicking in the upper left corner of the table, an event is generated in which the SEL_LEFT, SEL_TOP, SEL_RIGHT, and SEL_BOTTOM fields contain the value -1.
Note Two WIDGET_TABLE_CELL_SEL events are generated when an existing selection is changed to a new selection. If your code uses this event, be sure to differentiate between select and deselect events.

OnColWidth
The OnColWidth value is the routine that is called when the column width is changed. When you set this value, the calling sequence looks like this in the generated _eventcb.pro file:

```idl
PRO <RoutineName>, Event
```

*RoutineName* is the name of the event procedure you specify. *Event* is the returned event structure, which is returned when a column width is changed by the user and is of the following type:

```
{ WIDGET_TABLE_COLUMN_WIDTH, ID:0L, TOP:0L, HANDLER:0L, TYPE:7, COLUMN:0L, WIDTH:0L }
```

*COLUMN* contains the zero-based column number, and *WIDTH* contains the new width.

OnDelete
The OnDelete value is the routine that is called when text is deleted from the table. When you set this value, the calling sequence looks like this in the generated _eventcb.pro file:

```idl
PRO <RoutineName>, Event
```

*RoutineName* is the name of the event procedure you specify. *Event* is the returned event structure, which is returned when any amount of text is deleted from a cell of a table widget and is of the following type:

```
{ WIDGET_TABLE_DEL, ID:0L, TOP:0L, HANDLER:0L, TYPE:2, OFFSET:0L, LENGTH:0L, X:0L, Y:0L }
```

*OFFSET* is the (zero-based) character position of the first character deleted, and it is the insertion position that will result when the next character is inserted. *LENGTH* gives the number of characters involved. The *X* and *Y* fields give the zero-based address of the cell within the table.

OnFocus
The OnFocus value is the routine that is called when the keyboard focus of the base changes. When you set it, the calling sequence looks like this in the generated _eventcb.pro file:

```idl
PRO <RoutineName>, Event
```

*RoutineName* is the name of the event procedure you specify. *Event* is the returned event structure, which is of the following type:
Table Widget Properties

ENTER returns 1 (one) if the table widget is gaining the keyboard focus, or 0 (zero) if the table widget is losing the keyboard focus.

**OnInsertChar**

The OnInsertChar value is the routine that is called when text is inserted in the table. When you set this value, the calling sequence looks like this in the generated *_eventcb.pro file:

```
PRO <RoutineName>, Event
```

*RoutineName* is the name of the event procedure you specify. *Event* is the returned event structure, which is returned when a single character is typed into a cell of a table widget and is of the following type:

```
{ WIDGET_TABLE_CH, ID:0L, TOP:0L, HANDLER:0L, TYPE:0, OFFSET:0L,
  CH:0B, X:0L, Y:0L }
```

OFFSET is the (zero-based) insertion position that will result after the character is inserted. CH is the ASCII value of the character. The X and Y fields indicate the zero-based address of the cell within the table.

**OnInsertString**

The OnInsertString value is the routine that is called when text is inserted in the table. When you set this value, the calling sequence looks like this in the generated *_eventcb.pro file:

```
PRO <RoutineName>, Event
```

*RoutineName* is the name of the event procedure you specify. *Event* is the returned event structure, which is returned when multiple characters are pasted into a cell and is of the following type:

```
{ WIDGET_TABLE_STR, ID:0L, TOP:0L, HANDLER:0L, TYPE:1, OFFSET:0L,
  STR:'', X:0L, Y:0L }
```

OFFSET is the (zero-based) insertion position that will result after the text is inserted. STR is the string to be inserted. The X and Y fields indicate the zero-based address of the cell within the table.

**OnInvalidData**

The OnInvalidData value is the routine that is called when invalid data is set in a cell. When you set this value, the calling sequence looks like this in the generated *_eventcb.pro file:

```
PRO <RoutineName>, Event
```

*RoutineName* is the name of the event procedure you specify. *Event* is the returned event structure, which is returned when the text entered by the user
does not pass validation, and the user has finished editing the field (by pressing Tab or Enter). The event structure is of the following type:

```
{ WIDGET_TABLE_INVALID_ENTRY, ID:0L, TOP:0L, HANDLER:0L, TYPE:8,
  STR:'', X:0L, Y:0L }
```

STR contains invalid contents entered by the user as a text string. The X and Y fields contain the cell location.

**OnTextSelect**

The OnTextSelect value is the routine that is called when text is selected in the table. When you set this value, the calling sequence looks like this in the generated \*_eventcb.pro file:

```
PRO <RoutineName>, Event
```

**RoutineName** is the name of the event procedure you specify. Event is the returned event structure, which is returned when an area of text is selected. The event structure is of the following type:

```
{ WIDGET_TABLE_TEXT_SEL, ID:0L, TOP:0L, HANDLER:0L, TYPE:3,
  OFFSET:0L, LENGTH:0L, X:0L, Y:0L }
```

This event announces a change in the insertion point. OFFSET is the (zero-based) character position of the first character to be selected. LENGTH gives the number of characters involved. A LENGTH of zero indicates that the widget has no selection, and that the insertion position is given by OFFSET. The X and Y fields indicate the zero-based address of the cell within the table.
Chapter 8

Getting Help with IDL

The following topics are covered in this chapter:

- IDL's Online Help System .................... 242
- The Help Menus ................................. 242
- The IDL Online Help Button Bar .......... 247
- The Navigation Dialog ....................... 248
- Extending IDL's Online Help System.... 249
- The HELP Procedure............................ 252
IDL provides two kinds of help information: complete manual text on-line and status information concerning the current environment.

**IDL's Online Help System**

The entire text of the IDL documentation set:

- Using IDL
- Building IDL Applications
- IDL Reference Guide
- Object Graphics
- IDL External Development Guide
- IDL Scientific Data Formats
- Using IDL Insight
- IDL DataMiner Guide

...is available on-line by entering a question mark (?) at the IDL prompt or by selecting “Contents” from the Help menu of the IDL Development Environment. Because features are occasionally added to IDL after the printed manuals have gone to press, you will always find the most current documentation on any aspect of IDL in the on-line documentation.

After entering the ? command, a new window titled “IDL Online Help” appears. Depending upon the speed and type of your computer system, it may take a minute or two for the online help window to appear.

The IDL online help system is a searchable hypertext database. Hypertext links appear in green and can be followed by clicking on the green text with the mouse cursor.

More information about the help viewer can be found in the following sections. Note there are slight differences between the help systems on different IDL platforms.

**The Help Menus**

Several pulldown menus appear at the top of the IDL Online Help window. These menus are described below.
Chapter 8: Getting Help with IDL

File

This menu controls help system file and printing options.

Open
Select this option to open a different help file. Since IDL’s basic help system is contained in a single file named idl.hlp (located in the help subdirectory of the IDL distribution on Macintosh and Windows systems, located in the hyperhelp subdirectory of the help subdirectory on Unix and VMS), this option will not usually be needed.

Close (Macintosh only)
Select this option to close the current help file.

Print
Select this option to print the current topic or a group of topics.

Printer Setup/Page Setup
Select this option to configure IDL Online Help to work with your printer. This menu item is called “Page Setup” on the Macintosh.

Print All Topics (Macintosh only)
Select this option to print all topics in the help file. Note that since the IDL help system contains the full text of both the IDL User’s Guide and the IDL Reference Guide, choosing this option will result in several thousand pages being printed.

Exit/Quit
Select this option to dismiss the online help system. This menu item is called “Quit” on the Macintosh.

Edit

This menu allows you to copy text from the current topic to the clipboard or to add your own notes to a topic. On the Macintosh, the top six items (“Undo,” “Cut,” “Copy,” “Paste,” “Clear,” and “Select All”) function as they do in other Macintosh applications.

Delete Note (Macintosh only)
Select this option to delete the currently-selected “sticky-note” annotation.

Copy
Select this option to copy the selected text to the clipboard.
On the Macintosh you also have the option of saving a topic as a PICT file. Select "Copy Topic as Picture" to save a copy of the topic on the clipboard in PICT format.

**Annotate**

Select this option to add your own notes to the current topic. The Annotate window appears. Type your annotation in the window and click “OK” or “Save” to create a new annotation. A green “paper clip” icon appears at the top of the current topic. Clicking on this icon causes the saved annotation to appear. Annotations can be deleted by clicking the “Delete” button. Press “Cancel” to dismiss the Annotate window.

On Unix systems, annotations are stored in the subdirectory .hh of each user’s home directory (as defined by the $HOME environment variable) in the file idl.hlpab. This location can be changed by setting the value of the $HHLOCAL environment variable. Annotations can be shared if users set the same $HHLOCAL directory.

On Microsoft Windows systems, annotations are stored in the WINDOWS directory in the file idl.ann.

Annotations are handled slightly differently on the Macintosh. In the upper right-hand corner of the help window is a small yellow icon designed to look like a paper sticky-note. To add your own notes, drag a copy of this icon into the topic text. You can add as many lines of annotation, in as many separate notes, as you wish.

To delete an annotation on the Macintosh, place the cursor in the note you wish to delete and either drag it off the help window or select “Delete Note” from the Edit menu.

**Bookmark**

This menu allows you create “bookmarks” that allow easy access to frequently-used help topics.

**Define/Set Bookmark**

Select this option to create a new bookmark or delete existing bookmarks. The Bookmark window appears. The name of the bookmark to be created is shown in the “Selection” or “Bookmark Name” text field. The default name the bookmark is the name of the topic as shown in the non-scrolling region of the IDL Online Help window. On the Macintosh, you have the option of defining a command-key shortcut for the bookmark as well. Change the name or define a shortcut key if you desire, then click “OK”. The newly-defined bookmark becomes a menu item under the Bookmark menu. When this menu item is selected, IDL Online Help jumps to the bookmarked topic.
Under Unix and Windows, the Bookmark window also shows a list of any other bookmarks you have defined. To delete a previously-defined bookmark, click on its name in the list, then click the “Delete” button.

Click the “Cancel” button to dismiss the Bookmark window without changing the list of bookmarks.

**Edit Bookmarks... (Macintosh only)**

On the Macintosh, a second menu option displays a scrolling list of your bookmarks. Jump to a topic by selecting a bookmark from the list and clicking “View Topic.” Delete a bookmark by clicking “Delete Bookmark.” You can also change the name or command-key shortcut in the “Edit Bookmarks” window. Click “OK” to accept any changes you have made or “Cancel” to dismiss the window without saving changes.

**How Bookmarks are Stored**

On Unix systems, bookmarks are stored in each user’s home directory (as defined by the $HOME environment variable) in the file `idl.hlpab`. This location can be changed by setting the value of the $HHLOCAL environment variable. Bookmarks can be shared if users set the same $HHLOCAL directory.

On Microsoft Windows systems, bookmarks are stored in the WINDOWS directory in the file `idl.bkm`.

**Options**

This menu allows you to control various aspects of the online help viewer.

**Keep Help on Top (Windows only)**

Select an option from the sub-menu of this menu item to control whether help viewer windows float “on top” of other application windows or not. By default, help windows in the IDL online help system do not stay on top. Select “On Top” to force all the help windows in the IDL online help system to stay on top. In the IDL online help system, selecting “Not On Top” is the same as leaving the setting on “Default”.

**Display History Window**

Select this option to display the help viewer’s History window. The history window displays the names of recently-visited topics in the help file. Double-clicking on a topic from the list returns you to that topic.

**Font**

Select “Small,” “Normal,” or “Large” to change the size of the text in the help viewer window. Note that if you change the size of the font, some text may appear blocky in the viewer.
Use System Colors (Windows and Macintosh only)
Select this option to replace the colors used in the help file with the system defaults. You will be prompted to close the help file before the change takes effect. The IDL online help file already uses the system default colors, so selecting this option has no effect.

Navigate
This menu allows you to move through the IDL online help file quickly.

Chapter Topics
Select this item to display a pop-up window containing a list of topics in the current chapter. Click on the links in the pop-up window to move quickly from topic to topic.

Contents
Select this item to display the Contents tab. See “Contents Tab” on page 248 for a description of the Contents tab.

Index Search
Select this item to display the Index tab. From the Index tab you can search by keyword for topics in the IDL online help file. See “Index Tab” on page 248 for a description of the Index tab.

Text Search
Select this item to display the Find tab. From the Find tab you can search for any text in the IDL online help file. See “Find Tab” on page 249 for a description of the Find tab.

View (Macintosh only)
The View menu allows you to toggle display of annotations on and off by selecting the “Show Notes” option. The bottom of the View menu contains a list of help files that are currently open. To change to another open help file, select its name from the list.

Help
This menu can be used to access information about the help system. On the Macintosh, the help menu is at the “?” menu at the right of the menu bar.

How to Use Help/Help with QuickHelp
Select this option to open the help file on your platform’s help viewer.
Printer Help (Unix and VMS only)
Select this option to display help on the Bristol Xprinter printer setup process.

About Help
Select this option to see version and copyright information for the help viewer.
On the Macintosh, this menu item is under the Apple menu.

The IDL Online Help Button Bar
Directly below the IDL Online Help menus is a group of buttons that can be used to help navigate through the help system. These buttons are described in more detail below. Note that not all features are available on all platforms.

Contents
Click this button to open the Navigation dialog, displaying the Contents, Index, and Find tabs. See “The Navigation Dialog” on page 248 for a description of the Navigation dialog. Clicking the Contents button opens the Navigation dialog to the tab that was displayed the last time you used it.

Click this button to jump to the “IDL Online Help Navigator Page”. This is the first page that appears if you start the online help system without any arguments. This top-level page allows access to the table of contents for the IDL User’s Guide, alphabetical and topical lists of routines, lists of graphics keywords and system variables, and documentation for graphics devices. There are also links to release notes and more information about Research Systems products.

Back/Go Back
Click this button to jump to the previously-displayed topic. You can continue jumping backward through previously-displayed topics until you reach the first one.

Browse Buttons (<< and >>)
The browse buttons allow you to look at topics related to the currently-displayed topic as if you were browsing through the paper version of the manuals. Click the “>>” button to see the next topic in the sequence. Click the “<<” button to see the previous topic in the sequence. On the Macintosh, the browse buttons portray the pages of a book.

Note that the browse buttons are not active for every topic in the help system. For example, the “IDL Online Help Navigator” page is not part of a browse sequence.

There are many browse sequences defined in the help database. The topics that comprise most chapters in the printed versions of most of the books in the IDL
The Navigation Dialog

When you click the Contents button of the button bar or select an item from the Navigation menu, a tabbed navigation dialog appears. The navigation dialog includes the Contents, Index, and Find tabs.

Contents Tab

The Contents tab displays all the topics available in the IDL online help system, organized into books and sub-books. Double click on a book icon to display the contents of that section of the online help; opening a “book” may reveal one or more sub-books as well as individual topics. Double click on a topic (represented by a page icon) to display that topic.

The first topic displayed on the Contents tab takes you to the “IDL Online Help Navigator Page”. This top-level page allows access to several alternative navigational tools, including topics that list IDL routines alphabetically and by topic. The Navigator Page also includes links to release notes and more information about Research Systems products.

The four main volumes of the IDL documentation set— Using IDL, Building IDL Applications, Object Graphics, and the two volume Reference Guide— are contained in a single help file and are accessible in the main contents tab. Other volumes of the documentation set are contained in separate help files, which are opened by selecting the single topic revealed when you click on the book’s icon. On Unix and VMS platforms, these auxiliary files are opened in the same help viewer application as the main IDL help file. On Windows and Macintosh platforms, a separate viewer application is started for each help file opened from the Contents tab.

Index Tab

The Index tab allows you to search by keyword for topics in the currently open help file.

The Macintosh help viewer provides another method for keyword searching as well. Underneath the button bar is a field with the label “Keyword:”. Begin typing in this field and search keywords that match what you have typed so far will be displayed in gray. Press Return to jump directly to the topic containing the search keyword that is displayed. If more than one topic contains the search keyword...
you specify, the Index window will appear, allowing you to select the topic you wish to view.

**Find Tab**

The Find tab allows you to search for any text in the currently open online help file. Because the help viewer is loading a full-text search index, opening the Find tab for the first time may take a moment.

**Extending IDL's Online Help System**

Users may want to create online help for their own IDL applications, procedures, or functions. The online help system used by IDL emulates (or uses, in the case of IDL for Windows) the Microsoft Windows Help viewer on all supported platforms. Because the online help files are compiled, there is not a simple, no-cost way to include user-created help topics directly in the help system on all platforms. However, there are a number of ways to create your own online help. The techniques described below vary in complexity, cost, and level of integration with IDL's hypertext online help viewer.

**Creating Hypertext Files for Use with IDL's Hypertext Help Viewer**

As mentioned above, the online help system used by IDL emulates (or uses, in the case of IDL for Windows) the Microsoft Windows Help viewer on all supported platforms. It is possible to create your own hypertext help files that can be used with the viewer. The difficulty and expense involved in creating such files depends largely on the platform(s) involved.

**Microsoft Windows**

For Microsoft Windows systems, help files are relatively easy to create. Files must be created in the Rich Text Format (RTF) and compiled with Microsoft's help compiler. The help compiler is part of the Windows Software Developer's Kit, and is now included in several Microsoft programming products, including the Visual C++ development environment. The help compiler may also be available from the Microsoft ftp site (ftp.microsoft.com) or other Microsoft online software libraries at little or no cost.

The Windows help system is often referred to as “WinHelp”. The two components are the viewer (WINHELP.EXE, found in the main WINDOWS directory of all Windows systems), and the help compiler. There are a number of third-party “help authoring systems” that simplify the creation of WinHelp compatible RTF files. Also, a number of third party books describe the WinHelp creation process— Developing Online Help for Windows, by Scott Boggan, David Farkas,
and Joe Welinske, Sams Publishing, 1993, ISBN: 0-672-30230-6 is one that we have found useful.

**Macintosh**

For Macintosh, we use a WinHelp-compatible compiler and viewer licensed from Altura Software, Inc. called QuickHelp. This compiler uses the same RTF files as are used by the Microsoft Help compiler. Altura Software can be contacted at the following address:

Altura Software, Inc.
510 Lighthouse Avenue, Suite 5
Pacific Grove, CA 93950
Phone: 408-655-8005
Fax: 408-655-9663
AppleLink: ALTURA

**Unix and VMS**

For Unix and VMS, we use a compiler and viewer from Bristol Technology, Inc. called HyperHelp. Bristol makes a number of compilers that can compile source files in RTF, FrameMaker’s MIF (Maker Interchange Format), SGML (Standard Generalized Markup Language), HTML (Hypertext Markup Language), and their own simple HyperHelp Text (HHT) format. We use the MIF compiler to create hypertext help files from the same FrameMaker files that produce our hardcopy manuals.

Bristol also makes a product called Bridge that takes compiled HyperHelp files and converts them to RTF files that can be compiled with the Windows and Macintosh help compilers described above. In this way, we can create help files for all supported IDL platforms from a single source.

Bristol Technology can be contacted at the following address:

Bristol Technology, Inc.
241 Ethan Allen Highway
Ridgefield, CT 06877
Phone: 203-438-6969
Fax: 203-438-5013
E-mail: info@bristol.com
http://www.bristol.com

**Accessing Hypertext Help Files**

Once compiled, your own hypertext help files can be opened by selecting “Open” from the help viewer’s File menu. In addition, the ONLINE_HELP procedure can be used in your IDL programs to display help files and control the viewer. See “ONLINE_HELP” on page 781 of the IDL Reference Guide for details.
Chapter 8: Getting Help with IDL

Documentation Professional Services
It may be possible for documentation professionals from Research Systems to write and/or compile hypertext help files for IDL users on a contract basis. Contact the Research Systems sales department for details.

Creating HTML Files from User Routines
If you need a simple, low- or no-cost way to create hypertext online help for user-written routines, consider using HTML (HyperText Markup Language) files in conjunction with a WWW (World-Wide Web) browser such as Mosaic or Netscape. The MK_HTML_HELP procedure can be used to create HTML-format documentation from standard IDL documentation headers. These files can then be viewed with your web browser. See “MK_HTML_HELP” on page 751 of the IDL Reference Guide for details.

Browsers such as Mosaic and Netscape are available for most platforms supported by IDL. For information about the Mosaic viewers, send e-mail to mosaic@ncsa.uiuc.edu. For information about the Netscape viewers, send e-mail to info@netscape.com.

Support for IDL’s Previous Help System
While the procedures and functions required to run the pre-version-4.0 IDL help system are no longer documented, they are still included in IDL. We do not recommend that new users of IDL use the system described in this section.

Users who have existing .help files in the “old” format should move them to the help subdirectory of the main IDL directory. The IDL command WIDGET_O LH can be used to display the old-style help viewer, which will show a category for each .help file found. Note that the MAN_PROC procedure is also still available, but we recommend using WIDGET_O LH to call the old-style help viewer instead.

The MK_LIBRARY_HELP procedure, which creates .help files from standard IDL documentation headers, is still included. It can be found in the file mk_library_help.pro in the lib subdirectory of the main IDL directory. Read its documentation header for instructions. New users who want similar functionality (and previous users who want HTML support) should consider using the MK_HTML_HELP procedure, described above, instead. See “MK_HTML_HELP” on page 751 of the IDL Reference Guide for details.

The DOC_LIBRARY procedure (that reads individual IDL documentation headers) is still included and documented. See “DOC_LIBRARY” on page 440 of the IDL Reference Guide for details.
The HELP Procedure

The HELP procedure gives information about many different aspects of the current IDL session. Entering

HELP

with no parameters prints an overview of the current state, including the definitions of all current variables. Calling HELP with one or more parameters displays the definitions of its parameters.

The HELP procedure also will display other information about the current session if called with a keyword parameter indicating the topic. Only one topic keyword should be specified at a time. See “HELP” on page 537 of the IDL Reference Guide for an explanation of the keywords to HELP.

The Current State

When HELP is called without any plain or keyword arguments, it provides an overview of the current state. The information provided includes:

- A traceback showing the current procedure and function nesting.
- When IDL reads a procedure or function for the first time, it compiles it into the internal form that is executed by the IDL interpreter. Every routine has a code area where the executable code is stored and a data area where information about all locally available variables (including common block variables) resides. The amount of room used in each of these areas is reported, along with the number of local variables and parameters to the current routine.
- A one-line description of every current variable.

Note You can use the NAMES keyword to specify a pattern string that contains wildcard matching characters to select which variables are to be described. An example appears below.

- A description of all currently accessible common blocks.
- The names of all saved procedures and functions.

As an example of a typical IDL session, the command,

HELP

might result in output similar to the following:

% At $MAIN$(0).
Code area used: 0.00% (0/8192), Data area used: 4.88% (100/2048)
# local variables: 4, # parameters: 0
B BYTE = Array(256)
Chapter 8: Getting Help with IDL

Using IDL

The HELP Procedure

G BYTE = Array(256)
I BYTE = Array(512, 512)
R BYTE = Array(256)
X (CBLK) INT = 10
Y (CBLK) INT = 11
Z (CBLK) INT = 12

Common Blocks:
CBLK(3)

Saved Procedures:
COLOR_EDIT COLOR_EDIT_BACK INTERP_COLORS READ_SRF SHOW3

Saved Functions:
AVG BILINEAR CORRELATE CURVEFIT

This display gives the following information:

- The current routine is $MAIN$, which means that we are currently at the main program level and that no called routine is executing.

- The second line gives the information that the code area is empty (0 bytes used out of 8,192 available), and approximately 95 percent of the “data area” (which contains the symbol table) also is free (100 bytes used out of 2,048 available). Note that arrays are not stored in the data area. The code area is empty because we are currently at the main level, and no main program has been entered (using the .RUN executive command).

- The third line gives the number of local variables and the number of parameters to the current routine.

- The next seven lines give one-line descriptions of all locally available variables. The first four variables (B, G, I, and R) are local variables, while the other three (X, Y, and Z) are contained in the common block CBLK. Note that the one-line descriptions of scalar variables give their values, while the descriptions of arrays show their dimensions. Use the PRINT procedure to look at the contents of arrays.

- Following the descriptions of variables is the list of available common blocks. The session depicted above has only a single common block named CBLK, which contains three variables.

- The final information given is the names of all saved procedures and functions.

If you call HELP with arguments (but without any keyword arguments), it simply provides a one-line description of each argument. Hence, for the IDL session described above, the command,

```
HELP, 12.0 * 23, R, I, Z, !D
```

gives the output below:

```
<Expression> FLOAT = 276.000
R BYTE = Array(256)
```
\begin{verbatim}
I BYTE = Array(512, 512)
Z (CBLK) INT = 12
<Expression> STRUCT = -> !DEVICE
\end{verbatim}

It was noted above that the one-line description of scalars gives their values, and that arrays have their dimensions given instead. The last line of this example demonstrates that for structure variables, the name of the structure definition associated with the variable is given. Use the command:

\begin{verbatim}
HELP, /STRUCTURES
\end{verbatim}

to see the form of a structure variable.
Chapter 9
The Structure of the IDL Language

The following topics are covered in this chapter:

- Data Types ........................................... 256
- Numeric Constants.............................. 257
- String Constants................................. 258
- Type Conversion Functions................... 259
- Arrays ............................................... 260
- Structures ......................................... 262
- Variables .......................................... 264
- System Variables ............................... 266
IDL is a complete, structured language that can be used both interactively and to create sophisticated functions, procedures, and applications. The IDL language uses a deliberately simple syntax, and supports a wide range of data types.

This chapter provides a brief overview of IDL's data types and language constructs. Greater detail is available in Building IDL Applications, beginning with Constants and Variables.

Data Types

The IDL language is dynamically typed. This means that an operation on a variable can change that variable's type. In general, when variables of different types are combined in an expression, the result has the data type that yields the highest precision.

For example, if an integer variable is added to a floating-point variable, the result will be a floating-point variable.

Basic Data Types

In IDL there are twelve atomic data types, each with its own form of constant. The basic data types and their syntax are defined in Table 9-1.

<table>
<thead>
<tr>
<th>Data Types</th>
<th>Syntax</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Byte</td>
<td>nB</td>
<td>8-bit unsigned integer ranging in value from 0 to 255</td>
</tr>
<tr>
<td>Integer</td>
<td>n or nS</td>
<td>16-bit signed integer ranging from −32,768 to +32,767</td>
</tr>
<tr>
<td>Unsigned Integer</td>
<td>nU or nUS</td>
<td>16-bit unsigned integer ranging from 0 to 65535.</td>
</tr>
<tr>
<td>Longword</td>
<td>nL</td>
<td>32-bit signed integer ranging in value in the range of ± two billion, 2x10^9</td>
</tr>
<tr>
<td>Unsigned Longword</td>
<td>nUL</td>
<td>32-bit unsigned integer ranging in value in the range 0 to four billion.</td>
</tr>
<tr>
<td>Unsigned 64-bit Integer</td>
<td>nULL</td>
<td>Unsigned 64-bit integer ranging from 0 to 18,446,744,073,709,551,615.</td>
</tr>
<tr>
<td>Floating Point</td>
<td>n.n</td>
<td>32-bit, single-precision, floating-point number in the range of ±10^{38}(IEEE)</td>
</tr>
</tbody>
</table>

Table 9-1: Basic Data Types
Chapter 9: The Structure of the IDL Language

In addition to the twelve basic data types, IDL also supports the following:

- **Structures**: Aggregations of data of various types. Structures are discussed in chapter 4 of Building IDL Applications.
- **Pointers**: A reference to a dynamically-allocated heap variable. Pointers are discussed in chapter 12 of Building IDL Applications.
- **Object References**: A reference to a special heap variable that contains an IDL object structure. Object references are discussed in chapter 13 of Building IDL Applications.

### Numeric Constants

This section briefly discusses the features of IDL's numeric constants. The syntax of numeric constants is described further in “Constants and Variables” on page 11 of Building IDL Applications.

#### Integer Constants

Numeric constants of different types can be represented by a variety of forms. Integer constants can be decimal, hexadecimal or octal. Each of these radices can either be of byte, integer, long, or 64-bit long type. The absolute values of integer constants are given in Table 9-2.

<table>
<thead>
<tr>
<th>Type</th>
<th>Absolute Value Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Byte</td>
<td>0 - 255</td>
</tr>
<tr>
<td>Integer</td>
<td>0 - 32767</td>
</tr>
<tr>
<td>Unsigned Integer</td>
<td>0 - 65535</td>
</tr>
</tbody>
</table>

Table 9-2: Absolute Value Range Of Integer Constants
Integers specified without one of the B, S, L, or LL codes are automatically promoted to an integer type capable of holding them. For example, 4000 is promoted to longword because it is too large to fit into an integer. Any numeric constant can be preceded by a plus (+) or minus (−) sign.

**Floating-Point and Double-Precision Constants**

Floating-point and double-precision constants can be expressed in either conventional or scientific notation. Any numeric constant that includes a decimal point is a floating-point or double-precision constant.

**Complex Constants**

Complex constants contain a real and an imaginary part, both of which are single- or double-precision, floating-point numbers. The imaginary part can be omitted, in which case it is assumed to be zero. The form of a complex constant is as follows:

```
COMPLEX(REAL_PART, IMAGINARY_PART)
```

or

```
COMPLEX(REAL_PART)
```

For example, COMPLEX(1,2) is a complex constant with a real part of one, and an imaginary part of two. COMPLEX(1) is a complex constant with a real part of one and a zero imaginary component. To extract the real part of a complex expression, use the FLOAT function. The ABS function returns the magnitude of a complex expression, and the IMAGINARY function returns the imaginary part.

**String Constants**

An IDL string is a sequence of characters from 0 to 65,535 characters in length, enclosed by apostrophes (‘) or quotes (“). The value of the constant is simply the characters appearing between the leading delimiter (‘ or “) and the next occurrence of the same delimiter. A double apostrophe (‘‘) or quote (””) is
considered to be the null string; a string containing no characters. An apostrophe or quote can be represented within a string by two apostrophes or quotes; e.g., ‘Don’t’ returns Don’t. This syntax often can be avoided by using a different delimiter; e.g., "Don’t" instead of ‘Don’t’.

Strings have dynamic length (they grow or shrink to fit), and there is no need to declare the maximum length of a string prior to using it. As with any data type, string arrays can be created to hold more than a single string. In this case, the length of each individual string in the array depends only on its own length and is not affected by the lengths of the other string elements.

Features of IDL string constants are described further in “Strings” on page 71 of Building IDL Applications.

### Type Conversion Functions

The conversion functions are as follows:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STRING</td>
<td>Convert to string</td>
</tr>
<tr>
<td>BYTE</td>
<td>Convert to byte</td>
</tr>
<tr>
<td>FIX</td>
<td>Convert to (16-bit) integer</td>
</tr>
<tr>
<td>UINT</td>
<td>Convert to (16-bit) unsigned integer</td>
</tr>
<tr>
<td>LONG</td>
<td>Convert to (32-bit) longword</td>
</tr>
<tr>
<td>ULONG</td>
<td>Convert to (32-bit) unsigned longword</td>
</tr>
<tr>
<td>LONG64</td>
<td>Convert to (64-bit) integer</td>
</tr>
<tr>
<td>ULONG64</td>
<td>Convert to (64-bit) unsigned integer</td>
</tr>
<tr>
<td>FLOAT</td>
<td>Convert to floating-point</td>
</tr>
<tr>
<td>DOUBLE</td>
<td>Convert to double-precision floating-point</td>
</tr>
<tr>
<td>COMPLEX</td>
<td>Convert to complex value</td>
</tr>
<tr>
<td>DCOMPLEX</td>
<td>Convert to double-precision complex value</td>
</tr>
</tbody>
</table>

Table 9-3: Type Conversion Functions

These functions are useful in many instances, such as forcing the evaluation of an expression to a certain type, outputting data in a mode compatible with other programs, etc.

The following table illustrates several uses of type conversions:
Arrays

Type conversion between strings and bytes is a special case. The result of the BYTE function applied to a string or a string array is a byte array containing the ASCII codes of the characters of the string. Converting a byte array with the STRING function yields a string array or scalar with one less dimension than the argument.

IDL Type Conversion functions are described further in “Constants and Variables” on page 11 of Building IDL Applications.

### Arrays

Arrays are multidimensional data sets which are manipulated according to mathematical rules. Array elements can be of any IDL data type, but all elements of a given array must be of the same data type. Array subscripts provide a means of selecting one or more elements of an array for retrieval or modification.

This section provides a brief overview of IDL arrays. For more detailed information, see “Array Subscripts” on page 59 of Building IDL Applications.

One-dimensional arrays are often called vectors. The following IDL statement creates a vector with five single-precision floating-point elements:

```idl
array = [1.0, 2.0, 3.0, 4.0, 5.0]
```

Two-dimensional arrays are often used in image processing and in mathematical operations (where they are often termed matrices). The following IDL statement creates a three-column by two-row array:

```idl
array = [[1, 2, 3], [4, 5, 6]]
```
Use the PRINT procedure to display the contents of the array:

```
PRINT, array
```

IDL prints:
```
  1   2   3
  4   5   6
```

Arrays can have up to eight dimensions in IDL. The following IDL statement creates a three-column by four-row by five-layer deep three-dimensional array. In this case, we use the IDL FINDGEN function to create an array whose elements are set equal to the floating-point values of their one-dimensional subscripts:

```
array = FINDGEN(3, 4, 5)
```

IDL is an array-oriented language. This means that array operations execute more efficiently than similar one-dimensional operations. For example, suppose you have a three-dimensional array and wish to divide each element by two. A language that does not support array operations would create a loop to perform the division for each element; IDL accomplishes the division in a single line of code:

```
array = array / 2
```

### Array Subscripts

Individual array elements can be referenced using their subscripts. In IDL, array subscripts are zero-based; this means that the first element in an array is element zero, the second is element one, etc. For example, in the array created by the following IDL statement:

```
array = [ 1, 2, 3 ]
```

The integer 1 is element zero of the array, the integer 2 is element one, and the integer 3 is element two. The following IDL statement creates a new variable that contains element one of `array`:

```
new = array[1]
```

Displaying the value of `new` reveals the following:

```
PRINT, new
```

IDL prints:
```
2
```

The values of the selected array elements are extracted when a subscripted variable reference appears in an expression. New values are stored in selected array elements, without disturbing the remaining elements, when a subscript...
reference appears on the left side of an assignment statement. See “The Assignment Statement” on page 85 of Building IDL Applications for information on the use of the different types of assignment statements when storing into arrays.

The syntax of a subscript reference is:

Variable_Name [ Subscript_List ]

or

(Array_Expression) [ Subscript_List ]

The Subscript_List is simply a list of expressions, constants, or subscript ranges containing the values of one or more subscripts. Subscript expressions are separated by commas if there is more than one subscript. In addition, multiple elements are selected with subscript expressions that contain either a contiguous range of subscripts or an array of subscripts.

Note  In versions of IDL prior to version 5.0, parentheses were used to enclose array subscripts instead of square brackets. While the old syntax (using parentheses to enclose array subscripts) will continue to work, we suggest that you use square brackets in all new code. See “Array Subscript Syntax: [ ] vs. ( )” on page 60 of Building IDL Applications for a discussion of the change.

Structures

IDL supports structures and arrays of structures. A structure is a collection of scalars, arrays, or other structures contained in a variable. Structures are useful for representing data in a natural form, transferring data to and from other programs, and containing a group of related items of various types. There are two types of structures: named and anonymous. Named structures are used when the definitions will not be changed, since the attributes of the structure are stored internally in IDL and cannot be changed in the current IDL session. Anonymous structures are useful when the structure, type, and/or dimensions of its components change during program execution.

This section provides a brief overview of IDL structure variables. For more detailed information, see “Structures” on page 41 of Building IDL Applications.

Creating and Defining Structures

A named structure is created by executing a structure-definition expression, which is an expression of the following form:

{ Structure_Name, Tag_Name1 : Tag_Definition1, ..., Tag_Name_n : Tag_Definition_n }
Anonymous structures are created in the same way, but with the structure's name omitted.

\{Tag_Name1 : Tag_Definition1, ..., Tag_Name_n : Tag_Definition_n\}

Anonymous structures can also be created and combined using the CREATE_STRUCT function.

**Example** Assume that a star catalog is to be processed. Each entry for a star contains the following information: star name, right ascension, declination, and an intensity measured each month over the last 12 months. A structure for this information is defined with the following IDL statement:

\[ A = \{\text{STAR, NAME: ''}, \text{RA: 0.0, DEC: 0.0, INTEN: FLTARR(12)}\} \]

Each tag name, NAME, RA, DEC and INTEN, is followed by its tag definition.

The same structure is created as an anonymous structure by the statement:

\[ A = \{\text{NAME: ''}, \text{RA: 0.0, DEC: 0.0, INTEN: FLTARR(12)}\} \]

or by using the CREATE_STRUCT function:

\[ A = \text{CREATE_STRUCT('NAME', 'RA', 'DEC', 'INTEN', FLTARR(12))} \]

**Structure References**

The basic syntax of a reference to a field within a structure is as follows:

Variable_Name.Tag_Name

**Examples of Structure References**

The name of the star contained in \( A \) is referenced as \( A.\text{NAME} \). The entire intensity array is referred to as \( A.\text{INTEN} \), while the \( n \)th element of \( A.\text{INTEN} \) is \( A.\text{INTEN}[n] \). The following are valid IDL statements using the \text{STAR} structure:

\[ A = \{\text{STAR, NAME: 'SIRIUS', RA: 30., DEC: 40., INTEN: INDGEN(12)}\} \]

Store a structure of type \text{STAR} into variable \( A \). Define the values of all fields.

\[ A.\text{NAME} = 'BETELGEUSE' \]

Set name field only.

\[ \text{PRINT}, A.\text{NAME}, A.\text{RA}, A.\text{DEC} \]

Print name, right ascension, and declination.

\[ Q = A.\text{INTEN}[5] \]

Set \( Q \) to the value of the sixth element of \( A.\text{INTEN} \). \( Q \) will be a floating-point scalar.
A.RA = 23.21
Set RA field to 23.21.

A.INTEN = 0
Zero all 12 elements of intensity field.

B = A.INTEN[3:6]
Store 4th through 7th elements of INTEN field in variable B.

A.NAME = 12
The integer 12 is converted to string and stored in the name field because the field is defined as a string.

B = A
Copy A to B. The entire structure is copied and B contains a STAR structure.

Using HELP with Structures
The statement
HELP, /STRUCTURE, A
shows the type, structure and tag name of each field in a structure.

Parameter Passing with Structures
An entire structure is passed by reference by simply using the name of the variable containing the structure as a parameter. For example, the following statement prints the value of the structure field A.NAME:

PRINT, A.NAME

Variables
Variables are named repositories where information is stored. A variable can have virtually any size and can contain any of the IDL data types. Variables can be used to store images, spectra, single quantities, names, tables, etc.

This section provides a brief overview of IDL variables. For more detailed information, see “Constants and Variables” on page 11 of Building IDL Applications.

Attributes of Variables
Every variable has a number of attributes that can change during the execution of a program or terminal session. Variables have both a structure and a type.
Structure
A variable can contain a single value (a scalar) or a number of values of the same type (an array) or data entities of potentially differing type and size (a structure). Strings are considered as single values, and a string array contains a number of variable-length strings.

In addition, a variable can associate an array structure with a file; these variables are called associated variables. Referencing an associated variable causes data to be read from, or written to, the file. Associated variables are described in “ASSOC” on page 204 of the IDL Reference Guide.

Type
A variable can have one and only one of the following types: undefined, byte, integer, unsigned integer, 32-bit longword, unsigned 32-bit longword, 64-bit integer, unsigned 64-bit integer, floating-point, double-precision floating-point, complex floating-point, double-precision complex floating-point, string, structure, pointer, or object reference.

When a variable appears on the left-hand side of an assignment statement, its attributes are copied from those of the expression on the right-hand side. For example, the statement

ABC = DEF

redefines or initializes the variable ABC with the attributes and value of variable DEF. Attributes previously assigned to the variable are destroyed. Initially, every variable has the single attribute of undefined. Attempts to use the value of an undefined variables result in an error.

Variable Names
IDL variables are named by identifiers. Each identifier must begin with a letter and can contain from 1 to 128 characters. The second and subsequent characters can be letters, digits, the underscore character, or the dollar sign. A variable name cannot contain embedded spaces, because spaces are considered to be delimiters. Characters after the first 128 are ignored. Names are case insensitive. Lowercase letters are converted to uppercase; so the variable name abc is equivalent to the name ABC. Table 9-5 illustrates some acceptable and unacceptable variable names.

Caution A variable cannot have the same name as a function (either built-in or user defined) or a reserved word (see the list below). Giving a variable such a name results in a syntax error or in “hiding” the variable.

Table 9-6 lists all of the reserved words in IDL.
System Variables

System variables are a special class of predefined variables available to all program units. Their names always begin with the exclamation mark character (!). System variables are used to set the options for plotting, to set various internal modes, to return error status, etc.

System variables have a predefined type and structure that cannot be changed. When an expression is stored into a system variable, it is converted to the variable type, if necessary and possible. Certain system variables are read only, and their values cannot be changed. The user can define new system variables with the DEFSYSV procedure.

System variables are discussed in Chapter 3 of the IDL Reference Guide.
Chapter 10
Expressions and Operators

The following topics are covered in this chapter:

Operator Precedence .......................... 268
IDL Operators ..................................... 270
Type and Structure of Expressions....... 278
Variables and constants are combined into expressions using operators and functions, and providing a result. Expressions can be combined with other expressions, variables, and constants to yield more complex expressions. In IDL, unlike FORTRAN or BASIC, expressions can be scalar- or array-valued.

There are many types of operators in IDL. In addition to the usual operators—addition, subtraction, multiplication, division, exponentiation, relations (EQ, NE, GT, etc.), and Boolean arithmetic (AND, OR, NOT, and XOR)—operators exist to find minima, maxima, select scalars and subarrays from arrays (subscripting), and to concatenate scalars and arrays to form new arrays.

Functions, which are operators in themselves, perform operations that are usually of a more complex nature than those denoted by simple operators. Functions exist in IDL for data smoothing, shifting, transforming, evaluation of transcendental functions, and other operations.

Expressions can be arguments to functions or procedures. For example, the expression \( \text{SIN}(A \times \!\pi) \) evaluates the variable \( A \) multiplied by the value of \( \pi \), then applies the trigonometric sine function. This result can be used as an operand to form a more complex expression or as an argument to yet another function (e.g., \( \text{EXP}(\text{SIN}(A \times \!\pi)) \) evaluates \( e^{\sin \pi a} \)).

### Operator Precedence

IDL operators are divided into the levels of algebraic precedence found in common arithmetic. Operators with higher precedence are evaluated before those with lesser precedence, and operators of equal precedence are evaluated from left to right. Operators are grouped into five classes of precedence as shown in Table 10-1.

The effect of operators is based on precedence, not position. This concept is shown by the following examples.

\[ A = 4 + 5 \times 2 \]
\[ A = 14 \text{ because the multiplication operator has a higher precedence than the addition operator.} \]

Parentheses can be used to override the default evaluation.

\[ A = (4 + 5) \times 2 \]
\[ A = 18 \text{ because the expression inside the parentheses is evaluated first.} \]

A useful rule of thumb is, “when in doubt, parenthesize.” Some examples of expressions are provided in Table 10-2.
### Operator Precedence

<table>
<thead>
<tr>
<th>Priority</th>
<th>Operator</th>
</tr>
</thead>
<tbody>
<tr>
<td>First (highest)</td>
<td>* (pointer dereference)</td>
</tr>
<tr>
<td></td>
<td>^ (exponentiation)</td>
</tr>
<tr>
<td>Second</td>
<td>* (multiplication)</td>
</tr>
<tr>
<td></td>
<td># and ## (matrix multiplication)</td>
</tr>
<tr>
<td></td>
<td>/ (division)</td>
</tr>
<tr>
<td></td>
<td>MOD (modulus)</td>
</tr>
<tr>
<td>Third</td>
<td>+ (addition)</td>
</tr>
<tr>
<td></td>
<td>− (subtraction and negation)</td>
</tr>
<tr>
<td></td>
<td>&lt; (minimum)</td>
</tr>
<tr>
<td></td>
<td>&gt; (maximum)</td>
</tr>
<tr>
<td></td>
<td>NOT (Boolean negation)</td>
</tr>
<tr>
<td>Fourth</td>
<td>EQ (equality)</td>
</tr>
<tr>
<td></td>
<td>NE (not equal)</td>
</tr>
<tr>
<td></td>
<td>LE (less than or equal)</td>
</tr>
<tr>
<td></td>
<td>LT (less than)</td>
</tr>
<tr>
<td></td>
<td>GE (greater than or equal)</td>
</tr>
<tr>
<td></td>
<td>GT (greater than)</td>
</tr>
<tr>
<td>Fifth</td>
<td>AND (Boolean AND)</td>
</tr>
<tr>
<td></td>
<td>OR (Boolean OR)</td>
</tr>
<tr>
<td></td>
<td>XOR (Boolean exclusive OR)</td>
</tr>
<tr>
<td>Sixth</td>
<td>? (conditional expression)</td>
</tr>
</tbody>
</table>

Table 10-1: Operator Precedence

### Operator Precedence Examples

<table>
<thead>
<tr>
<th>Expression</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>A + 1</td>
<td>The sum of A and 1.</td>
</tr>
<tr>
<td>A &lt; 2 + 1</td>
<td>The smaller of A or two, plus one.</td>
</tr>
<tr>
<td>A &lt; 2 * 3</td>
<td>The smaller of A and six, since * has higher precedence than &lt;.</td>
</tr>
<tr>
<td>2 * SQRT(A)</td>
<td>Twice the square root of A.</td>
</tr>
<tr>
<td>A + 'Thursday'</td>
<td>The concatenation of the strings A and &quot;Thursday.&quot; An error results if A is not a string</td>
</tr>
</tbody>
</table>

Table 10-2: Operator Precedence Examples
IDL Operators

As described above, operators are used to combine terms and expressions. The set of IDL operators is described below:

Parentheses

Parentheses are used to group expressions and to enclose function parameter lists. Parentheses can be used to override the order of operator evaluation described above. Examples:

\[ \text{SIN(ANG} \* \text{PI/180.)} \]

Parentheses enclose function argument lists.

\[ (A + 5)/B \]

Parentheses specify order of operator evaluation.

The right parenthesis must always close the list begun by the left parenthesis.

Square Brackets

Square brackets are used to create arrays and to enclose array subscripts.

\[ \text{ARRAY} = [1, 2, 3, 4, 5] \]

Use brackets when assigning elements to an array.

\[ \text{ARRAY}[X, Y] \]

Brackets enclose subscripts.

**Note** In versions of IDL prior to version 5.0, parentheses were used to enclose array subscripts. While using parentheses to enclose array subscripts will continue to work as in previous versions of IDL, we strongly suggest that you use brackets in all new code. See “Array Subscript Syntax: [ ] vs. ( )” on page 60 of Building IDL Applications for additional details.

Mathematical Operators

There are seven basic IDL mathematical operators, described below.

Assignment

The equal sign (=) is the assignment operator. The value of the expression on the right hand side of the equal sign is stored in the variable, subscript element, or range on the left side. See “The Assignment Statement” on page 85 for more details. For example:

\[ A = 32 \]

Assigns the value 32 to A.

Addition

The positive sign (+) is the addition operator. When applied to strings, the addition operator concatenates the strings. Example:
B = 3 + 6  
Store the sum of 3 and 6 in B.

B = 'John' + ' ' + 'Doe'  
Store the string value of "John Doe" in B.

**Subtraction and Negation**

The negative sign (-) is the subtraction operator. Also, the minus sign is used as the unary negation operator. Example:

C = 9 - 5  
Store the value of 5 subtracted from 9 in C.

C = -C  
Change the sign of C.

**Multiplication**

The asterisk (*) is the multiplication operator. Example:

C = 2 * 5  
Store the product of 2 and 5 in variable C.

**Division**

The forward slash (/) is the division operator. Example:

D = 10.0 / 3.2  
Store the result of 10.0 divided by 3.2 in variable D.

**Exponentiation**

The caret (^) is the exponentiation operator. $A^B$ is equal to $A$ raised to the $B$ power.

- If $A$ is a real number and $B$ is of integer type, repeated multiplication is applied.
- If $A$ is real and $B$ is real (non-integer), the formula $A^B = e^{B \ln A}$ is evaluated.
- If $A$ is complex and $B$ is real, the formula $A^B = (r e^{i \theta})^B = r^B (\cos B \theta + i \sin B \theta)$ (where $r$ is the real part of $A$ and $i \theta$ is the imaginary part) is evaluated.
- If $B$ is complex, the formula $A^B = e^{B \ln A}$ is evaluated. If $A$ is also complex, the natural logarithm is computed to be $\ln (A) = \ln (r e^{i \theta}) = \ln (r) + i \theta$ (where $r$ is the real part of $A$ and $i \theta$ is the imaginary part).
- $A^0$ is defined as 1.

**Modulo**

The keyword MOD is the modulo operator. $I \mod J$ is equal to the remainder when $I$ is divided by $J$. The magnitude of the result is less than that of $J$, and its sign agrees with that of $I$. 
Example

\[ A = 9 \text{ MOD } 5 \]  Assign the value of 9 modulo 5 (4) to A.

\[ A = (\text{ANGLE} + B) \text{ MOD } (2 \times \pi) \]  Compute angle modulo 2p.

Minimum and Maximum Operators

The IDL minimum and maximum operators return the smaller or larger of their operands, as described below. Note that negated values must be enclosed in parentheses in order for IDL to interpret them correctly.

The Minimum Operator

The “less than” sign (<) is the IDL minimum operator. The value of “A < B” is equal to the smaller of A or B. Examples:

\[ A = 5 < 3 \]  Set A equal to 3.

\[ A = 5 < (-6) \]  Set A equal to -6.

\[ \text{ARR} = \text{ARR} < 100 \]  Set all points in array ARR that are larger than 100 to 100.

\[ X = x0 < x1 < x2 \]  Set X to the smallest of the three operands.

The Maximum Operator

The “greater than” sign (>) is the IDL maximum operator. “A > B” is equal to the larger of A or B. Examples:

\[ C = \text{ALOG}(D > 1E^{-6}) \]  ‘>’ is used to avoid taking the log of zero or negative numbers.

\[ \text{PLOT}, \text{ARR} > 0 \]  Plot positive points only. Negative points are plotted as zero.

Matrix Multiplication

IDL has two operators used to multiply arrays and matrices.

The # Operator

The # operator computes array elements by multiplying the columns of the first array by the rows of the second array. The second array must have the same number of columns as the first array has rows. The resulting array has the same number of columns as the first array and the same number of rows as the second.
The ## Operator

The ## operator does what is commonly referred to as matrix multiplication. It computes array elements by multiplying the rows of the first array by the columns of the second array. The second array must have the same number of rows as the first array has columns. The resulting array has the same number of rows as the first array and the same number of columns as the second array.

For an example illustrating the difference between the two, see “Multiplying Arrays” on page 430.

Array Concatenation

The square brackets are used as array concatenation operators. Operands enclosed in square brackets and separated by commas are concatenated to form larger arrays. The expression \([A, B]\) is an array formed by concatenating A and B, which can be scalars or arrays, along the first dimension.

Similarly, \([A, B, C]\) concatenates A, B, and C. The second and third dimensions can be concatenated by nesting the bracket levels; \([[1, 2], [3, 4]]\) is a 2-element by 2-element array with the first row containing 1 and 2 and the second row containing 3 and 4. Operands must have compatible dimensions; all dimensions must be equal except the dimension that is to be concatenated, e.g., \([2, \text{INTARR}(2, 2)]\) are incompatible. Examples:

\[
\begin{align*}
C &= [-1, 1, -1] & \text{Define C as three-point vector.} \\
C &= [C, 12] & \text{Add 12 to the end of C.} \\
C &= [12, C] & \text{Insert 12 at the beginning of C.} \\
\text{PLOT}, [\text{ARR1}, \text{ARR2}] & \text{Plot ARR2 appended to ARR1.} \\
\text{KER} &= [[1,2,1], [2,4,2], [1,2,1]] & \text{Define a 3x3 matrix.}
\end{align*}
\]

**Note** The maximum number of operands that can appear within brackets varies across IDL implementations but is always at least 25. This number is also influenced by the .SIZE executive command. If you must create an array of more than 25 elements using the concatenation operator, use multiple statements. For example, to create an array with 70-constant elements, use the following statements:

\[
\begin{align*}
A &= [k_0, k_1, \ldots, k_{24}] \\
A &= [A, k_{25}, k_{26}, \ldots, k_{49}] \\
A &= [A, k_{50}, k_{51}, \ldots, k_{69}]
\end{align*}
\]

This method is relatively inefficient and should be performed only once if possible.
Boolean Operators

There are four Boolean operators in IDL. Boolean operators return either “true” or “false” as described previously. Note that the Boolean operators do not work with string and complex arguments.

AND

AND is a Boolean operator that returns “true” whenever both of its operands are true; otherwise, the result is “false.” Any nonzero value is considered true. For integer, longword, and byte operands, a bitwise AND operation is performed. For operations on other types, the result is equal to the second operand if the first operand is not equal to zero or the null string; otherwise, the result is zero or the null string.

NOT

The NOT operator is the Boolean inverse and is a unary operator (it has only one operand). In other words, “NOT true” is equal to “false” and “NOT false” is equal to “true.” NOT complements each bit for integer operands.

Note  Signed integers are expressed using the “2’s complement” representation. This means that to arrive at the decimal representation of a negative binary number (a string of binary digits with a one as the most significant bit), you must take the complement of each bit, add one, convert to decimal, and prepend a negative sign. This means that NOT 0 equals -1, NOT 1 equals -2, etc.

For floating-point operands, the result is 1.0 if the operand is zero; otherwise, the result is zero. The NOT operator is not valid for string or complex operands.

OR

OR is the Boolean inclusive operator. For integer or byte operands, a bitwise inclusive OR is performed. For example, 3 OR 5 equals 7. For floating-point operands, the OR operator returns the first operand if it is non-zero, or the 2nd operand otherwise.

XOR

XOR is the Boolean “exclusive or” function. XOR is only valid for byte, integer, and longword operands. A bit in the result is set to 1 if the corresponding bits in the operands are different; if they are equal, it is set to zero.

Table 10-3 summarizes the action of the boolean operators:

When applied to bytes, integers, and longword operands, the Boolean functions operate on each binary bit. For example:
Results of relational expressions can be combined into more complex expressions using the Boolean operators. Some examples of relational and Boolean expressions are as follows:

(A LE 50) AND (A GE 25)  
True if A is between 25 and 50. If A is an array, then the result is an array of zeros and ones.

(A GT 50) OR (A LT 25)  
True if A is less than 25 or greater than 50. This is the inverse of the first.

ARR AND 'FF'X  
Adds (using the logical AND operator) the hexadecimal constant FF (255 in decimal) to the array ARR. This masks the lower 8-bits and zeros the upper bits.

### Relational Operators

The IDL relational operators can be used to test the relationship between two arguments. The six relational operators are described in the following table:

<table>
<thead>
<tr>
<th>Operator(op)</th>
<th>T op T</th>
<th>T op F</th>
<th>F op F</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>OR</td>
<td>T</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>XOR</td>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>NOT</td>
<td>F</td>
<td>T</td>
<td></td>
</tr>
</tbody>
</table>

Table 10-3: Action of Boolean Operators

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 AND 5 = 1</td>
<td>0011 AND 0101 = 0001</td>
</tr>
<tr>
<td>3 OR 5 = 7</td>
<td>0011 OR 0101 = 0111</td>
</tr>
<tr>
<td>3 XOR 5 = 6</td>
<td>0011 XOR 0101 = 0110</td>
</tr>
<tr>
<td>NOT 5 = -6</td>
<td>NOT 0101 = 1010</td>
</tr>
</tbody>
</table>

Table 10-4: Action of Boolean Operators on Integers
Relational operators apply a relation to two operands and return a logical value of true or false. The resulting logical value can be used as the predicate in IF, WHILE or REPEAT statements can be combined using Boolean operators with other logical values to make more complex expressions. For example: \( 1 \text{ EQ } 1 \) is true, and \( 1 \text{ GT } 3 \) is false.

The rules for evaluating relational expressions with operands of mixed modes are the same as those given above for arithmetic expressions. For example, in the relational expression \( (2 \text{ EQ } 2.0) \), the integer 2 is converted to floating point and compared to the floating point 2.0. The result of this expression is true, as represented by a byte 1.

Individual relational operators are described at the end of this chapter—see “Relational Operator Descriptions” below.

**Definition of True**

In IDL, the value “true” is represented by the following:

- Any odd, nonzero value for byte, integer, and longword data types
- Any nonzero value for single, double-precision, and the real part of a complex number (the imaginary part is ignored)
- Any non-null string

Conversely, false is represented as anything that is not true—zero or even-valued integers, zero-valued, floating-point quantities, and the null string.

The relational operators return a value of 1 for true and 0 for false. The type of the result is always byte.

### Table 10-5: Relational Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQ</td>
<td>Equal to</td>
</tr>
<tr>
<td>NE</td>
<td>Not equal to</td>
</tr>
<tr>
<td>LE</td>
<td>Less than or equal to</td>
</tr>
<tr>
<td>LT</td>
<td>Less than</td>
</tr>
<tr>
<td>GE</td>
<td>Greater than or equal to</td>
</tr>
<tr>
<td>GT</td>
<td>Greater than</td>
</tr>
</tbody>
</table>

Relational operators apply a relation to two operands and return a logical value of true or false. The resulting logical value can be used as the predicate in IF, WHILE or REPEAT statements can be combined using Boolean operators with other logical values to make more complex expressions. For example: \( 1 \text{ EQ } 1 \) is true, and \( 1 \text{ GT } 3 \) is false.

The rules for evaluating relational expressions with operands of mixed modes are the same as those given above for arithmetic expressions. For example, in the relational expression \( (2 \text{ EQ } 2.0) \), the integer 2 is converted to floating point and compared to the floating point 2.0. The result of this expression is true, as represented by a byte 1.

Individual relational operators are described at the end of this chapter—see “Relational Operator Descriptions” below.

**Definition of True**

In IDL, the value “true” is represented by the following:

- Any odd, nonzero value for byte, integer, and longword data types
- Any nonzero value for single, double-precision, and the real part of a complex number (the imaginary part is ignored)
- Any non-null string

Conversely, false is represented as anything that is not true—zero or even-valued integers, zero-valued, floating-point quantities, and the null string.

The relational operators return a value of 1 for true and 0 for false. The type of the result is always byte.
Using Relational Operators with Arrays

Relational operators can be applied to arrays, and the resulting array of ones and zeroes can be used as an operand. For example, the expression, \( \text{ARR} \times (\text{ARR} \leq 100) \) is an array equal to \( \text{ARR} \) except that all points greater than 100 have been reduced to zero. The expression \( (\text{ARR} \leq 100) \) is an array that contains a 1 where the corresponding element of \( \text{ARR} \) is less than or equal to 100, and zero otherwise.

\[
\text{PRINT, TOTAL(ARR GT 0)}
\]

prints the number of positive elements in the array \( \text{ARR} \).

Relational Operator Descriptions

The six relational operators are described below:

**EQ**

EQ is the relational “equal to” operator. This operator returns true if its operands are equal; otherwise, it returns false. This operator always returns a byte value of 1 for true and a byte value of 0 for false.

**GE**

GE is the “greater than or equal to” relational operator. The GE operator returns true if the operand on the left is greater than or equal to the one on the right. One use of relational operators is to mask arrays as shown in the following statement:

\[
A = \text{ARRAY} \times (\text{ARRAY} \geq 100)
\]

This command sets \( A \) equal to \( \text{ARRAY} \) whenever the corresponding element of \( \text{ARRAY} \) is greater than or equal to 100. If the element is less than 100, the corresponding element of \( A \) is set to zero.

Strings are compared using the ASCII collating sequence: "a" is less than "0" is less than "9" is less than "A" is less than "Z" is less than "a" which is less than "z".

**GT**

GT is the “greater than” relational operator. This operator returns true if the operand on the left is greater than the operand on the right. For example, \( 6 \ GT 5 \) returns true.

**LE**

LE is the “less-than or equal-to” relational operator. This operator returns true if the operand on the left is less than or equal to the operand on the right. For example, \( 4 \ LE \ 4 \) returns true.
LT
LT is the “less-than” relational operator. This operator returns true if the operand on the left is less than the operand on the right. For example, 3 LT 4 returns true.

NE
NE is the “not-equal-to” relational operator. This operator returns true whenever the operands are different. For example "sun" NE "fun" returns true.

Conditional Expression

?:
The conditional expression—written with the ternary operator ? :—has the lowest precedence of all the operators and is used wherever any other expression is used. It provides an alternate way to write simple constructions of the IF:THEN:ELSE combination. The following statement shows an example:

IF (a GT b) THEN z = a ELSE z = b  z holds the greater value, a or b. Note that if a=b, z holds b.

Using a conditional expression, this statement can be simplified:

z = (a GT b) ? a : b  Set z to the greater of a and b, with z=b if a=b.

The general form of this expression follows:

expr1 ? expr2 : expr3

The expression expr1 is evaluated first. If expr1 is true, then the expression expr2 is evaluated and set as the value of the conditional expression. If expr1 is false, expr3 is evaluated and set as the value of the conditional expression. Either expr2 or expr3 is evaluated, based on the result of expr1.

Note Since ?: has very low precedence—just above assignment—parentheses are not necessary around the first expression expr1. Parentheses are advisable anyway to distinguish the condition part of the expression.

Type and Structure of Expressions

Every entity in IDL has an associated type and structure. The twelve atomic data types in decreasing order of precedence are as follows:

- Double-precision complex floating-point
- Complex floating-point
• Double-precision floating-point
• Floating-point
• Signed and unsigned 64-bit integer
• Signed and unsigned longword (32-bit) integer
• Signed and unsigned (16-bit) integer
• Byte
• String

The structure of an expression can be either a scalar or an array. The type and structure of an expression depends on the type and structure of its operands. Unlike many other languages, the type and structure of most expressions in IDL cannot be determined until the expression is evaluated. Because of this, care must be taken when writing programs. For example, a variable can be a scalar byte variable at one point in a program while at a later point it can be set to a complex array.

Expression Type

IDL attempts to evaluate expressions containing operands of different types in the most accurate manner possible. The result of an operation becomes the same type as the operand with the greatest precedence or potential precision. For example, when adding a byte variable to a floating-point variable, the byte variable is first converted to floating-point, then added to the floating-point variable, yielding a floating-point result. When adding a double-precision variable to a complex variable, the result is double-precision complex because the double-precision complex type has a higher position in the hierarchy of data types.

Note  Signed and unsigned integers of a given width have the same precedence. In an expression involving a combination of such types, the result is given the result of the leftmost operand.

When writing expressions with mixed types, care must be taken to obtain the desired results. For example, assume the variable A is an integer variable with a value of 5. The following expressions yield the indicated results:

A / 2 = 2  Integer division is performed. The remainder is discarded.

A / 2. = 2.5  The value of A is first converted to floating.
280 Chapter 10: Expressions and Operators

A / 2 + 1. = 3. Integer division is done first because of operator precedence. Result is floating point.

A / 2. + 1 = 3.5 Division is done in floating, then the 1 is converted to floating and added.

A + 5U = 10 Signed and unsigned integer operands have the same precedence, so the leftmost operand determines the type of the result as signed integer.

5U + 1 = 10U As above, the leftmost operand determines the result type between types with the same precedence.

Note When other types are converted to complex type, the real part of the result is obtained from the original value and the imaginary part is set to zero.

When a string type appears as an operand with a numeric data type, the string is converted to the type of the numeric term. For example: ‘123’ + 123.0 is 246.0, while ‘123.333’ + 33 gives the result 156 because 123.333 is first converted to integer type. In the same manner, ‘ABC’ + 123 also causes a conversion error.

Expression Structure

IDL expressions can contain operands with different structures, just as they can contain operands with different types. Structure conversion is independent of type conversion. An expression will yield an array result if any of its operands is an array, as shown in the following table:

<table>
<thead>
<tr>
<th>Operands</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scalar : Scalar</td>
<td>Scalar</td>
</tr>
<tr>
<td>Array : Array</td>
<td>Array</td>
</tr>
<tr>
<td>Scalar : Array</td>
<td>Array</td>
</tr>
<tr>
<td>Array : Scalar</td>
<td>Array</td>
</tr>
</tbody>
</table>

Table 10-6: Structure of Expressions

Functions exist to create arrays of the seven types: BYTARR, INTARR, UINTARR, LONARR, UONARR, LON64ARR, UON64ARR, FLTARR, DBLARR.
Chapter 10: Expressions and Operators

COMPLEXARR, and STRARR. The dimensions of the desired array are the parameters to these functions. The result of \texttt{FLTARR(5)} is a floating-point array with one dimension, a vector, with five elements initialized to zero. \texttt{FLTARR(50, 100)} is a two-dimensional array, a matrix, with 50 columns and 100 rows.

The size of an array-valued expression is equal to the smaller of its array operands. For example, adding a 50-point array to a 100-point array gives a 50-point array; the last 50 points of the larger array are ignored. Array operations are performed point-by-point, without regard to individual dimensions. An operation involving a scalar and an array always yields an array of identical dimensions. When two arrays of equal size (number of elements) but different structure are operands, the result is of the same structure as the first operand. For example:

\texttt{FLTARR(4) + FLTARR(1, 4)} \quad \text{Yields} \texttt{fltarr(4)}.

In the above example, a row vector is added to a column vector and a row vector is obtained because the operands are the same size. This causes the result to take the structure of the first operand. Here are some examples of expressions involving arrays:

\texttt{ARR + 1} \quad \text{An array in which each element is equal to the same element in ARR plus one. The result has the same dimensions as ARR. If ARR is byte or integer, the result is of integer type; otherwise, the result is the same type as ARR.}

\texttt{ARR1 + ARR2} \quad \text{An array obtained by summing two arrays.}

\texttt{(ARR < 100) * 2} \quad \text{An array in which each element is set to twice the smaller of either the corresponding element of ARR or 100.}

\texttt{EXP(ARR/10.)} \quad \text{An array in which each element is equal to the exponential of the same element of ARR divided by 10.}

\texttt{ARR * 3./MAX(ARR)} \quad \text{An inefficient way of writing the following line.}

\texttt{ARR * (3./MAX(ARR))}
In the last example, each point in ARR is multiplied by three, then divided by the largest element of ARR. The MAX function returns the largest element of its array argument. This way of writing the statement requires that each element of ARR be operated on twice. If \((3./\text{MAX}(\text{ARR}))\) is evaluated with one division and the result then multiplied by each point in ARR, the process requires approximately half the time.
Chapter 11

Plotting

The following topics are contained in this chapter:

Plotting Keyword Parameters ............. 284
Graphics Coordinate Systems ............. 285
X Versus Y Plots—PLOT and OPlot .... 287
Annotation – The XYOUTS Procedure . 293
Plotting Symbols ................................ 295
Polygon Filling ............................... 298
Tick Marks ...................................... 302

Logarithmic Scaling ............................ 305
Multiple Plots on a Page ..................... 306
Specifying the Location of the Plot ..... 307
Plotting Missing Data ........................ 309
Using the AXIS Procedure ............... 310
Using the CURSOR Procedure .......... 313
IDL includes several routines that can be used to display data in a variety of plot formats, including general \( x \) versus \( y \), contour, mesh surface, and perspective plots. The routines allow users to display information in a manner that can be easily understood during data analysis.

Optional keyword parameters and system variables enable users to change certain specifications of the routines, such as scaling, style, and colors, for custom or specialized plots.

This chapter provides examples of scientific graphics in which one variable is plotted as a function of another. The routines for the display of functions of two variables, CONTOUR, SHADE_SURF, and SURFACE, are explained in detail in “Plotting Multi-Dimensional Arrays” of Using IDL.

For a complete list of plotting routines, see the IDL HandiGuide or IDL's online help system.

Direct Graphics vs. Object Graphics

The examples in this chapter are all written to take advantage of IDL Direct Graphics. Examples and techniques using IDL Object Graphics are contained in a separate volume: Objects and Object Graphics. See that volume for more information on IDL Object Graphics or see “Graphics” on page 267 of Building IDL Applications for a discussion of the differences between IDL Direct Graphics and IDL Object Graphics.

Running the Example Code

Some of the example code used in this chapter is part of the IDL distribution. All of the files mentioned are located in the \texttt{doc} subdirectory of the \texttt{examples} subdirectory of the main IDL directory. 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 “IPATH” on page 43 of the IDL Reference Guide for information on IDL's path.

Plotting Keyword Parameters

The IDL plotting procedures are designed to produce acceptable results for most applications with a minimum amount of effort. The large number of keyword parameters, described in the IDL Reference Guide, in combination with plotting and graphic system variables, allow users to customize the graphics produced by IDL. Most of these keyword parameters pertain to advanced programming. The major keyword parameters are described and illustrated by example in this chapter.
Correspondence with System Variables

Many of the keyword parameters correspond directly to fields in the system variables !P, !X, !Y, or !Z. When specifying a keyword parameter name and value in a call that value affects only the current call, the corresponding system-variable field is not changed. Changing the value of a system-variable field changes the default for that particular parameter and remains in effect until explicitly changed. The system variables involving graphics and their corresponding keywords are detailed in “System Variables” in the IDL Reference Guide.

Example—The COLOR Keyword Parameter

The keyword parameter COLOR corresponds to the field COLOR of the system-variable structure !P and is referenced as !P.COLOR. To set the color of a plot to color index 12, use the following statement:

```
PLOT, X, Y, COLOR = 12
```

Future plots are not affected and are drawn with color index !P.COLOR, which is normally set to the number of available colors minus one.

The interpretation of the color index varies among the devices supported by IDL. With color video displays, this index selects a color (normally a red, green, blue (RGB) triple stored in a device table). You can control the color selected by each color index with the TVLCT procedure which loads the device color tables.

Other devices have a fixed color associated with each color index. With plotters, for example, the correspondence between colors and color index is established by the order of the pens in the carousel.

To change the default color of future plots, use a statement such as:

```
!P.COLOR = 12
```

which sets the default color to color index 12. You can override this default at any time by including the COLOR keyword in the graphic routine call.

Graphics 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. See “CONVERT_COORD Function” on page 287.
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.

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.

Two-Dimensional Coordinate Conversion
The system variables !D, !P, !X, !Y, and !Z contain the information necessary to
convert from one coordinate system to another. The relevant fields of these
system variables are explained below, and formulae are given for conversions to
and from each coordinate system. Three-dimensional coordinates are discussed
in Chapter .

In the following discussion, \(D\) is a data coordinate, \(N\) is a normalized coordinate,
and \(R\) is a raw device coordinate.

The fields !D.X_VSIZE and !D.Y_VSIZE always contain the size of the visible area
of the currently selected display or drawing surface. Let \(V_x\) and \(V_y\) represent these
two sizes.

The field !X.S is a two-element array that contains the parameters of the linear
equation, converting data coordinates to normalized coordinates. !X.S[0] is the
intercept, and !X.S[1] is the slope. !X.TYPE is 0 for a linear x-axis and 1 for a
logarithmic x-axis. The y and z axes are handled in the same manner, using the
system variables !Y and !Z.

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), and \(X_i = !X.S_i\) (the
scaling parameter).

With the above variables defined, the linear two-dimensional coordinate
conversions for the \(x\) coordinate can be written as follows:
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.

**CONVERT_COORD Function**

The CONVERT_COORD function provides a convenient means of computing the above transformations. It can convert coordinates to and from any of the above systems. The keywords DATA, DEVICE, or NORMAL specify the input system. The output coordinate system is specified by one of the keywords TO_DATA, TO_DEVICE, or TO_NORMAL. For example, to convert the endpoints of a line from data coordinates (0, 1) to (5, 7) to device coordinates, use the following statement:

\[ D = \text{CONVERT\_COORD}([0, 5], [1, 7], \text{DATA}, \text{TO\_DEVICE}) \]

On completion, the variable \( D \) is a (3, 2) vector, containing the x, y, and z coordinates of the two endpoints.

**X Versus Y Plots—PLOT and OPLOT**

This section illustrates the use of the basic x versus y plotting routines, PLOT and OPLOT. PLOT produces linear-linear plots by default, and can produce linear-log, log-linear, or log-log plots with the addition of the XLOG and YLOG keywords.

Data used in these examples are from a fictitious study of Pacific Northwest Salmon fisheries. In the example, we suppose that data were collected in the years 1967, 1970, and from 1975 to 1983. The following IDL statements create and
initialize the variables SOCKEYE, COHO, CHINOOK, and HUMPBACK, which contain fictitious fish population counts, in thousands, for the 11 observations:

SOCKEYE = [463, 459, 437, 433, 431, 433, 431, 428, 430, 431, 430]
COHO = [468, 461, 431, 430, 427, 425, 423, 420, 418, 421, 420]
CHINOOK = [514, 509, 495, 497, 497, 494, 493, 491, 492, 493, 493]
HUMPBACK = [467, 465, 449, 446, 445, 444, 443, 443, 443, 443, 445]
YEAR = [1967, 1970, INDGEN(9) + 1975]

Constructs a vector in which each element contains the year of the sample.

If you prefer not to enter the data by hand, run the batch file plot01 with the following command at the IDL prompt:

@plot01

See “Running the Example Code” on page 284 if IDL does not find the batch file.

The following IDL commands create a plot of the population of Sockeye salmon, by year:

PLOT, YEAR, SOCKEYE, $  
TITLE='Sockeye Population', XTITLE='Year', $  
YTITLE='Fish (thousands)'

The PLOT procedure, which produces an x versus y plot on a new set of axes, requires one or two parameters: a vector of y values or a vector of x values followed by a vector of y values. The first attempt at making a plot produces Figure 11-1. Note that the three titles, defined by the keywords TITLE, XTITLE, and YTITLE, are optional.

**Axis Scaling**

The fluctuations in the data are hard to see because the scores range from 428 to 463, and the plot's y-axis is scaled from 0 to 500. Two factors cause this effect. By default, IDL sets the minimum y-axis value of linear plots to zero if the y data are all positive. The maximum axis value is automatically set by IDL from the maximum y data value. In addition, IDL attempts to produce from three to six tick-mark intervals that are in increments of an integer power of 10 times 2, 2.5, 5, or 10. In this example, this rounding effect causes the maximum axis value to be 500, rather than 463.

The YNOZERO keyword parameter inhibits setting the y-axis minimum to zero when given positive, nonzero data. Figure 11-2 illustrates the data plotted using
Chapter 11: Plotting

Using IDL

X Versus Y Plots—PLOT and OPL

Figure 11-1: Initial Population Plot

this keyword. The y-axis now ranges from 420 to 470, and IDL creates tick-mark intervals of 10.

```
@plot01
PLOT, YEAR, SOCKEYE, /YNOZERO,
    TITLE='Sockeye Population', XTITLE='Year', YTITLE='Fish (thousands)'
```

**Multiline Titles**

The graph-text positioning command !C, starts a new line of text output. Titles containing more than one line of text are easily produced by separating each line with this positioning command.

In the above example, the main title could have been displayed on two centered lines by changing the keyword parameter TITLE to the following statement:

```
TITLE = 'Sockeye!CPopulation'
```

**Note** When using multiple line titles you may find that the default margins are inadequate, causing the titles to run off the page. In this case, set the [XY]MARGIN keywords or increase the values of !X.MARGIN or !Y.MARGIN.
Range Keyword

The range of the x, y, or z axes can be explicitly specified with the [XYZ] RANGE keyword parameter. The argument of the keyword parameter is a two-element vector containing the minimum and maximum axis values.

As explained above, IDL attempts to produce even tick intervals, and the axis range selected by IDL may be slightly larger than that given with the RANGE keyword. To obtain the exact specified interval, set the axis style parameter to one (YSTYLE = 1).

The effect of the YNOZERO keyword is identical to that obtained by including the keyword parameter YRANGE = [MIN(Y), MAX(Y)] in the call to PLOT. You can make /YNOZERO the default in subsequent plots by setting bit 4 of !Y.STYLE to one (!Y.STYLE = 16).

See “STYLE” on page 54 of the IDL Reference Guide for details on the STYLE field of the axis system variables !X, !Y, and !Z. Briefly: Other bits in the STYLE field extend the axes by providing a margin around the data, suppress the axis and
its notation, and suppress the box-style axes by drawing only left and bottom axes.

For example, to constrain the x-axis to the years 1975 to 1983, the keyword parameter \texttt{XRANGE} = [1975, 1983] is included in the call to \texttt{PLOT}. Figure 11-3 illustrates the result. Note that the x-axis actually extends from 1974 to 1984, as IDL elected to make five tick-mark intervals, each spanning two years. If, as explained above, the x-axis style is set to one, the plot will exactly span the given range. The call combining all these options is as follows:

\begin{verbatim}
@plot01
Define variables.
PLOT, YEAR, SOCKEYE, /YNOZERO, $ 
   TITLE='Sockeye Population', XTITLE = 'Year', $ 
   YTITLE = 'Fish (thousands)', XRANGE = [1975, 1983], /XSTYLE
\end{verbatim}

\textbf{Note} The keyword parameter syntax \texttt{/XSTYLE} is synonymous with the expression \texttt{XSTYLE = 1}. Setting a keyword parameter to 1 is often referred to as simply setting the keyword.

Figure 11-3: Plot with X-Axis Range of 1975 to 1983
Overplotting

Additional data can be added to existing plots with the OPLOT procedure. Each call to PLOT establishes the plot window (the rectangular area enclosed by the axes), the plot region (the box enclosing the plot window and its annotation), the axis types (linear or log), and the scaling. This information is saved in the system variables !P, !X, and !Y and used by subsequent calls to OPLOT.

Frequently, the color index, linestyle, or line thickness parameters are changed in each call to OPLOT to distinguish the data sets. The IDL Reference Guide contains a table describing the line style associated with each index.

Figure 11-4 illustrates a plot showing all four data sets. Each data set except the first was plotted with a different line style and was produced by a call to OPLOT. In this example, an (11, 4) array called ALLPTS is defined and contains all the scores for the four categories using the array concatenation operator. Once this array is defined, the IDL array operators and functions can be applied to the entire data set, rather than explicitly referencing the particular sample.

First, we define an n x 4 array containing all four sample vectors. (This array is also defined by the plot01 batch file.)

\[
\text{ALLPTS} = \begin{bmatrix}
\text{[COHO]}, & \text{[SOCKEYE]}, & \text{[HUMPBACK]}, & \text{[CHINOOK]}
\end{bmatrix}
\]
The plot in Figure 11-4 was produced with the following statements:

@plot01 Define variables.

PLOT, YEAR, COHO, YRANGE = [MIN(ALLPTS), MAX(ALLPTS)], $
   TITLE='Salmon Populations', XTITLE = 'Year',$
   YTITLE = 'Fish (thousands)', XRANGE = [1975, 1983], $
/XSTYLE Plot first graph. Set they-axis min and max from the min and max of all data sets. Default linestyle is 0.

FOR I = 1, 3 DO OPLT, YEAR, ALLPTS[*; I], LINE = I Loop for the three remaining scores, varying the linestyle.

**Annotation – The XYOUTS Procedure**

An obvious problem with Figure 11-4 is that each line should be labeled showing what it depicts. The XYOUTS procedure is used to write graphic text at a given location. The call to XYOUTS to write a string starting at location \((x, y)\) is as follows:

\[ \text{XYOUTS, } X, Y, \text{STRING} \]

See “XYOUTS” on page 1379 of the IDL Reference Guide for a complete list of keywords available when adding graphic text to a plot.

Figure 11-5 illustrates one method of annotating each graph with its name. The plot was produced exactly as was Figure 11-4, except that the x-axis range was extended to the year 1990 to allow room for the titles. To accomplish this, the keyword parameter \(\text{XRANGE} = [1967, 1990]\) was added to the call to PLOT. A string vector, NAMES, containing the names of each sample population also is defined. The annotation was written using the following statements:

First, we define an array containing names for each of the lines plotted. (This array is also defined by the plot01 batch file.)

\[ \text{NAMES= ['Coho', 'Sockeye', 'Humpback', 'Chinook']} \]

The plot in Figure 11-5 was produced with the following statements:

@plot01 Define variables.

\[ \text{N1 = N_ELEMENTS(YEAR) - 1} \] Index of last point.

PLOT, YEAR, COHO, YRANGE = [MIN(ALLPTS), MAX(ALLPTS)], $
   TITLE='Salmon Populations', XTITLE = 'Year', $
YTITLE = 'Fish (thousands)', X RANGE = [1965, 1990], $ /XSTYLE

FOR I = 1, 3 DO OPLOT, YEAR, ALLPTS[*, I], LINE = I
Loop for the three remaining scores, varying the linestyle.

FOR I = 0, 3 DO XYOUTS, 1984, ALLPTS[N1, I], NAMES[I] Append the title of each graph on the right.

**Font Selection**

Figure 11-5 also illustrates the use of a PostScript font (Times-Roman, in this case) for the titles and annotations. Note that PostScript fonts can only be used when the current graphics devices is set to PostScript.

This font was selected by first setting the default font, controlled by the system variable !P.FONT, to the hardware-font index of zero, and then calling the DEVICE procedure to set the Times-Roman font. To recreate the plot using this font on your system, inspect the batch file plot02, located in the doc subdirectory of the examples subdirectory of the main IDL directory. Note that running this batch files creates a PostScript file named plot.ps in your current
Chapter 11: Plotting

Plotting Symbols

Each data point can be marked with a symbol and/or connected with lines. The value of the keyword parameter PSYM selects the marker symbol, as described in the IDL Reference Guide. For example, a value of 1 marks each data point with the plus sign, 2 is an asterisk, etc. Setting PSYM to minus the symbol number marks the points with a symbol and connects them with lines. A value of -1 marks points with a plus sign and connects them with lines. Note also that setting PSYM to a value of 10 produces histogram style plots in which a horizontal line is drawn across each x bin.

Frequently, when data points are plotted against the results of a fit or model, symbols are used to mark the data points while the model is plotted using a line. Figure 11-6 illustrates this, fitting the Sockeye population values to a quadratic function of the year. The IDL function POLY_FIT is used to calculate the quadratic.

The statements used to construct the plot on the left are as follows:

@plot01 Define variables.
coeff = LINFIT(YEAR, SOCKEYE)

Use the LINFIT function to fit the data to a line.

YFIT = coeff[0] + coeff[1]*YEAR

YFIT is the fitted line.

PLOT, YEAR, SOCKEYE, /YNOZERO, PSYM = 4, $

TITLE = 'Quadratic Fit', XTITLE = 'Year', $

YTITLE = 'Sockeye Population'

Plot the original data points with PSYM = 4, for diamonds.

O PLOT, YEAR, YFIT

; Overplot the smooth curve using a plain line.

Alternatively, you can run the following batch file to create the plot:
@plot03

See “Running the Example Code” on page 284 if IDL does not find the batch file.

Defining Your Own Plotting Symbols

The USERSYM procedure allows you to define your own symbols by supplying the coordinates of the lines used to draw the symbol. The symbol you define can be drawn using lines or it can be filled using the polygon filling operator.

USERSYM accepts two vector parameters: a vector of x values and a vector of y values. The coordinate system you use to define the symbol’s shape is centered on each data point, and each unit is approximately the size of a character. For example, to define the simplest symbol, use a one character-wide dash centered over the data point:

USERSYM, [-.5, .5], [0, 0]

The color and line thickness used to draw the symbols are also optional keyword parameters of USERSYM. The right half of Figure 11-6 illustrates the use of USERSYM to define a new symbol—a filled circle. It was produced in exactly the same manner as the example above, except with the addition of the following statements that define the marker symbol and use it:

A = FINDGEN(17) * (!PI*2/16.)

Make a vector of 16 points, A[i] = 2πi/16.

USERSYM, COS(A), SIN(A), /FILL

Define the symbol to be a unit circle with 16 points. Set the filled flag.

PLOT, YEAR, SOCKEYE, /YNOZ, PSYM = 8, ... ...

As above, but use symbol index 8 to select user-defined symbols.
Histogram Mode

Using the keyword PSYM=10 with the PLOT routines draws graphs in the “histogram mode,” connecting points with vertical and horizontal lines. Figure 11-7 illustrates the comparison between the distribution of the IDL normally distributed random number function (RANDOMN) to the theoretical normal distribution.

The plot was produced by the following IDL commands:

\[ X = \text{FINDGEN}(200) \div 20. - 5. \]
\[ Y = \frac{1}{\sqrt{2\pi}} \cdot \exp\left(-\frac{X^2}{2}\right) \cdot \left(\frac{10}{200}\right) \]

Two-hundred values ranging from -5 to 4.95.

Theoretical normal distribution, scale so integral is one.

\[ H = \text{HISTOGRAM}(\text{RANDOMN(SEED, 2000)}, \text{BINSIZE} = 0.4, \text{MIN} = -5., \text{MAX} = 5.) \div 2000. \]

Approximate normal distribution with RANDOMN, then form the histogram.

See “USERSYM” on page 1177 of the IDL Reference Guide for additional details.
Polygon Filling

Many scientific graphs use region filling to highlight the difference between two or more curves, to illustrate boundaries, etc. The IDL POLYFILL procedure fills the interior of arbitrary polygons given a list of vertices. The interior of the polygon can be filled with a solid color or with some devices, a user-defined fill pattern contained in a rectangular array.

Figure 11-8 illustrates a simple example of polygon filling by filling the region under the Chinook population graph with a color index of 25 percent the maximum, then filling the region under the Sockeye population graph with 50 percent of the maximum index. Because the Chinook populations are always higher than the Sockeye populations, the graph appears as two distinct regions.

The program that produced this figure is shown below. It first draws a plot axis with no data, using the NODATA keyword. The minimum and maximum y values are directly specified with the YRANGE keyword. Because the y-axis range does not always exactly include the specified interval (see "X Versus Y Plots—
PLOT and O PLOT" on page 287), the variable MIN VAL, is set to the current y-axis minimum, !Y.CRANGE[0]. Next, the upper Chinook population region is shaded with a polygon that contains the vertices of the Chinook samples, preceded and followed by points on the x-axis, (YEAR[0], MINVAL), and (YEAR[n-1], MINVAL). The polygon for the Sockeye samples is drawn using the same method with a different color. Finally, the XYOUTS procedure is used to annotate the two regions.

Enter the following IDL commands to create the plot:

```idl
@plot01
Define variables.

PLOT, YEAR, CHINOOK, YRANGE = [MIN(SOCKEYE), MAX(CHINOOK)], $ /NODATA, TITLE='Sockeye and Chinook Populations', $ XTITLE='Year', YTITLE='Fish (thousands)'

PXVAL = [YEAR[0], YEAR, YEAR[N1]]
Make a vector of x values for the polygon by duplicating the first and last points.

MINVAL = !Y.CRANGE[0]
Get y value along bottom x-axis.

POLYFILL, PXVAL, [MINVAL, CHINOOK, MINVAL], $ COL = 0.75 * !D.N_COLORS
Make a polygon by extending the edges down to the x-axis.

POLYFILL, PXVAL, [MINVAL, SOCKEYE, MINVAL], $ COL = 0.50 * !D.N_COLORS
Same with second polygon.

XYOUTS, 1968, 430, 'SOCKEYE', SIZE=2
Label the polygons.

XYOUTS, 1968, 490, 'CHINOOK', SIZE=2

Alternatively, you can run the following batch file to create the plot:

@plot04
See “Running the Example Code” on page 284 if IDL does not find the batch file.

Bar Charts

Bar (or box) charts are used in business-style graphics and are useful in comparing a small number of measurements within a few discrete data sets. Although not designed as a tool for business graphics, IDL can produce many business-style plots with little effort.

The following example produces a box-style chart showing the four salmon populations as boxes of differing colors or shading. The commands used to draw
Figure 11-9: Bar Chart Drawn with POLYFILL

Figure 11-9 are shown below and annotated. You do not need to type these commands in yourself; they are collected in the files `plot05.pro`, which contains the two procedures, and `plot06`, which contains the found in the `doc` subdirectory of the examples subdirectory of the main IDL directory.

First, we define a procedure called BOX, which draws a box given the coordinates of two diagonal corners:

```idl
PRO BOX, X0, Y0, X1, Y1, color

Define a procedure that draws a box, using POLYFILL, whose corners are (X0, Y0) and (X1, Y1).

POLYFILL, [X0, X0, X1, X1], [Y0, Y1, Y1, Y0], COL = color

Call POLYFILL.

END

End the program.

Next, create a procedure to draw the bar graph:

```idl
PRO BARGRAPH, minval

Define a procedure that produces a bar graph from the population data.

@plot01

Define variables.

del = 1./5.

Width of bars in data units.
ncol=\!D.N\_COLORS/5

The number of colors used in the bar graph is defined by the number of colors available on your system.

colors = ncol*INDGEN(4)+ncol

Create a vector of color indices to be used in this procedure.

FOR iscore = 0, 3 DO BEGIN

Loop for each sample.

yannot = minval + 20 *(iscore+1)

They value of annotation. Vertical separation is 20 data units.

XYOUTS, 1984, yannot, names[iscore]

Label for each bar.

BOX, 1984, yannot - 6, 1988, yannot - 2, colors[iscore]

Bar for annotation.

xoff = iscore * del - 2 * del

The x offset of vertical bar for each sample.

FOR iyr=0, N\_ELEMENTS(year)-1 DO $

Draw vertical box for each year's sample.

BOX, year[iyr] + xoff, minval, $
year[iyr] + xoff + del, $
allpts[iyr, iscore], $
colors[iscore]

ENDFOR

End the FOR loop.

END

End the procedure.

Enter the following at the IDL prompt to compile these two procedures from the IDL distribution:

.run plot5.pro

To create the bar graph on your screen, enter the following commands.

LOADCT, 39

Load a color table.

As in the previous example, the PLOT procedure is used to draw the axes and to establish the scaling using the NODATA keyword.

PLOT, year, CHINOOK, YRANGE = [MIN(allpts), MAX(allpts)], $ 
TITLE = 'Salmon Populations', /NODATA, $ 
XRANGE = [year[0], 1990]

Get they value of the bottom x-axis.

minval = \!Y\_CRANGE[0]
BARGRAPH, minval
Create the bar chart.

Alternatively, you can run the following batch file to create the plot:
@plot06

See “Running the Example Code” on page 284 if IDL does not find the batch file.

**Tick Marks**

You have almost complete control of the number, style, placement, thickness, and annotation of the tick marks. The following plotting keyword parameters and their corresponding system variable fields affect the tick marks:

**[XYZ]GRIDSTYLE**
The index of the linestyle to be used for plot tickmarks and grids (i.e., when TICKLEN is set to 1.0). See “[XYZ]GRIDSTYLE” on page 102 of the IDL Reference Guide for more information.

**[XYZ]MINOR**
The number of minor-tick intervals. If set to zero, the default, IDL automatically determines the number of minor ticks in each major tick-mark interval. Setting this parameter to 1 suppresses the minor ticks, and setting it to a positive, nonzero number, n, produces n minor-tick intervals, and n-1 minor-tick marks. See “[XYZ]MINOR” on page 103 of the IDL Reference Guide for more information.

**[XYZ]THICK**
The thickness of the x, y, or z axes and their tick marks. This parameter is set with the field THICK in the axes system variables, !X, !Y, and !Z (e.g., !X.THICK controls the x-axis thickness). There are no keyword parameters affecting the axis thickness. See “[XYZ]THICK” on page 107 of the IDL Reference Guide for more information.

**[XYZ]TICKFORMAT**
Set this keyword to a format string or a string containing the name of a function that returns a string to be used to format the axis tick mark labels. See “[XYZ]TICKFORMAT” on page 107 of the IDL Reference Guide for more information.

**TICKLEN**
The length of each major-tick mark, expressed as a fraction of the window size in the tick mark’s direction. The default value is 0.02. A length of 1.0 produces a grid. A value of -0.02 makes tick marks that extend away from the plot. Individual
axis ticks can be controlled with the \([XYZ]TICKLEN\) keyword. See “TICKLEN” on page 108 of the IDL Reference Guide for more information.

**\([XYZ]TICKNAME\)**

A string array containing the annotation of each major-tick mark. If omitted or if a given string element contains the null string, IDL labels the tick mark with its value. To suppress the tick labels, supply a string array of one-character long, blank strings, i.e., `REPLICATE(' ', N)`. Null strings force IDL to number the tick mark with its value.

**Note** If there are \(n\) tick-mark intervals, there are \(n + 1\) tick marks and labels.


**\([XYZ]TICKS\)**

The number of major tick-mark intervals. If set to zero or omitted, IDL produces between three and six intervals. See \([XYZ]TICKS\)” on page 109 of the IDL Reference Guide for more information.

**\([XYZ]TICKV\)**

The data values of each tick mark. You can directly specify these values, producing graphs with arbitrary tick marks. If you do this, IDL scales the axis from the first tick value to the last unless you directly specify a range. As above, be sure to provide \(n + 1\) tick values. See \([XYZ]TICKV\)” on page 109 of the IDL Reference Guide for more information.

**Example: Specifying Tick Marks**

Figure 11-10 is a box chart that illustrates the direct specification of the x-axis tick values, number of ticks, and tick names. Building upon the previous program, this program shows each of the four scores for the year 1967, the first year in our data. It uses the BOX procedure from the previous example to draw a rectangle for each sample. Using the data and variables from above, the following commands create the box chart:

Enter

```
.run plot5.pro
```

at the IDL prompt to compile the BOX and BARGRAPH procedures (discussed in the previous example) from the IDL distribution. Enter the following commands to create the box chart:

```
@plot01
Define variables.
XVAL = FINDGEN(4)/5. + .2
Tick x values, 0.2, 0.4, 0.6, 0.8.
```
304 Chapter 11: Plotting

Tick Marks

Figure 11-10: Controlling Tick Marks and Their Annotation

YVAL = [COHO[0], SOCKEYE[0], HUMPBACK[0], CHINOOK[0]]

Make a vector of scores from first year, corresponding to the name vector from above.

PLOT, XVAL, YVAL, /YNOZERO, XRANGE = [0,1], XTICKV = XVAL, $\ 
XTICKS = 3, XTICKNAME = NAMES, /NODATA, $\ 
TITLE = 'Salmon Populations, 1967'

Make the axes with no data. Force x range to [0,1], centering xval, which also contains the tick values. Force three tick intervals making four tick marks. Specify the tick names from the names vector.

FOR I = 0, 3 DO BOX, XVAL[I] - .08, !Y.CRANGE[0], $\ 
XVAL[I] + 0.08, YVAL[I], 128

Draw the boxes, centered over the tick marks. !Y.CRANGE[0] is the y value of the bottom x-axis.

Alternatively, you can run the following batch file to create the plot:

@plot07
Logarithmic Scaling

The XLOG, YLOG, and ZLOG keywords can be used with the PLOT routine to get any combination of linear and logarithmic axes. The OPLOT procedure uses the same scaling and transformation as did the most recent plot. Figure 11-11 illustrates using PLOT to make a linear-log plot. It was produced with the following statements:

\[
\begin{align*}
X & = \text{FLTARR}(256) & \text{Created data array.} \\
X[80:120] & = 1 & \text{Make a step function. Array elements 80 through 120 are set to 1.} \\
\text{FREQ} & = \text{FINDGEN}(256) & \text{Make a filter.}
\end{align*}
\]

Figure 11-11: Example of Logarithmic Scaling

See “Running the Example Code” on page 284 if IDL does not find the batch file.

More Tick Mark Examples

See “Multiple Plots on a Page” on page 306 for more examples of ways you can control where axes are drawn, tick mark length, and placement.
FREQ = FREQ < (256-FREQ)  Make the filter symmetrical about the value \( x = 128 \).

FIL = 1./(1+(FREQ/20)^2)  Second order Butterworth, cutoff frequency = 20.

PLOT, /YLOG, FREQ, ABS(FFT(X,1)), $ 
  XTITLE = 'Relative Frequency', YTITLE = 'Power', $ 
  XSTYLE = 1  Plot with a logarithmic x-axis. Use exact axis range.

OPLOT, FREQ, FIL  Plot graph.

Alternatively, you can run the following batch file to create the plot:

@plot08

See “Running the Example Code” on page 284 if IDL does not find the batch file.

### Multiple Plots on a Page

Plots can be “ganged” on the display or page in the horizontal and/or vertical directions using the system variable field \!P\_MULTI. IDL sets the plot window to produce the given number of plots on each page and moves the window to a new sector at the beginning of each plot. If the page is full, it is first erased. If more than two rows or columns of plots are produced, IDL decreases the character size by a factor of 2.

\!P\_MULTI controls the output of multiple plots. Set \!P\_MULTI equal to an integer vector in which:

- The first element of the vector contains the number of empty sectors remaining on the page. The display is erased if this field is zero when a new plot is begun.
- The second element of the vector contains the number of plots per page in the horizontal direction.
- The third element contains the number of plots per page in the vertical direction.
- The fourth element contains the number of plots stacked in the Z dimension.
- The fifth element controls the order in which plots are drawn. Set the fifth element equal to zero to make plots from left to right (column major), and top to bottom. Set the fifth element equal to one to make plots from top to bottom, left to right (row major).
Omitting any of the five elements from the vector is the same as setting that element equal to zero.

For example, to set up IDL to stack two plots vertically on each page, use the following statement:

```idl
!P.MULTI = [0, 1, 2]
```

Note that the first element, `!P.MULTI (0)`, is set to zero to cause the next plot to begin a new page. To make four plots per page with two columns and two rows, use the following statement:

```idl
!P.MULTI = [0, 2, 2]
```

To reset to the default of one plot per page, set the value of `!P.MULTI` to 0, as shown in the following statement:

```idl
!P.MULTI = 0
```

Figure 11-12 shows four plots in a single window. For details, inspect the batch file `plot09` in the `doc` subdirectory of the `examples` subdirectory of the main IDL directory. Note the following features of the plots in the figure:

1. The plot in the upper left has grid-style tick marks. This is accomplished by setting the `TICKLEN` keyword equal to 1.0
2. The plot in the upper right has outward-facing tick marks. This is accomplished by setting the `TICKLEN` keyword to a negative value.
3. The plot in the lower left corner has different axes on left and right, top and bottom. This is accomplished by drawing the top and right axes separately, using the `AXIS` procedure.
4. The plot in the lower right uses no default axes at all. The centered axes are drawn with calls to the `AXIS` procedure.

**Specifying the Location of the Plot**

The plot-data window is the region of the page or screen enclosed by the axes. The plot region is the box enclosing the plot-data window and the titles and tick annotation. Figure 11-13 illustrates the relationship of the plot-data window, plot region, and the entire device area. These areas are determined by the following system variables and keyword parameters, in order of decreasing precedence:

**POSITION**

The `POSITION` keyword is accepted by the `CONTOUR`, `MAP_SET`, `PLOT`, `SHADE_SURF`, and `SURFACE` routines. Its value is a four-element vector (six...
elements for three-dimensional plots) containing the position of the axis endpoints: \([x_0, y_0, x_1, y_1]\). Coordinates are specified in normalized coordinates or in device coordinates if the DEVICE keyword is present.

**!P.POSITION**

!P.POSITION is the system variable equivalent of the POSITION keyword. Its value is a four-element vector in the same form as above containing the normalized coordinates of the plot-data window. !P.POSITION is ignored if \(x_0\) is equal to \(x_1\), (that is, if \(!P\).POSITION[0] EQ !P.POSITION[2]), which is the default.

**!P.REGION**

The !P.REGION system variable is another four-element vector in the same form as above containing the normalized coordinates of the plot region, the rectangle

---

Figure 11-12: Multiple Plots Per Page, Various Tick Marks, and Multiple Axes
Plotting Missing Data

The MAX_VALUE and MIN_VALUE keywords to PLOT can be used to create “missing data” plots wherein “bad” data values are not plotted. Data values
greater than the value of the \texttt{MAX\_VALUE} keyword or less than the value of the \\
\texttt{MIN\_VALUE} keyword are treated as missing and are not plotted. The following \\
code creates a dataset with “bad” data values and plots it with and without these \\
keywords:

\begin{Verbatim}
\texttt{A = FINDGEN(100)} \hspace{2cm} \text{Make a 100-element array where} \\
\texttt{each element is set equal to its index.}
\end{Verbatim}

\begin{Verbatim}
\texttt{A(RANDOMU(SEED, 20)*100)=400} \hspace{2cm} \text{Set 20 random point in the array} \\
equal to 400. This simulates “bad” data values above the range of the \\
“real” data.
\end{Verbatim}

\begin{Verbatim}
\texttt{A(RANDOMU(SEED, 20)*100)=-10} \hspace{2cm} \text{Set 20 random point in the array} \\
equal to -10. This simulates “bad” data values below the range of the \\
“real” data.
\end{Verbatim}

\begin{Verbatim}
\texttt{PLOT, A} \hspace{2cm} \text{Plot the dataset with the bad val}\\n\text{ues. Looks pretty bad!}
\end{Verbatim}

\begin{Verbatim}
\texttt{PLOT, A, MAX\_VALUE=101} \hspace{2cm} \text{Plot the dataset, but don’t plot any} \\
value over 101. The resulting plot looks better, but still shows spuri-
\text{ous values.}
\end{Verbatim}

\begin{Verbatim}
\texttt{PLOT, A, MAX\_VALUE=101, MIN\_VALUE=0} \hspace{2cm} \text{This time leave out both high and} \\
low spurious values. The resulting plot more accurately reflects the \\
“real” data.
\end{Verbatim}

The following plotting routines allow you to set maximum and minimum values \\
in this manner: \texttt{CONTOUR}, \texttt{PLOT}, \texttt{SHADE\_SURF}, \texttt{SURFACE}.

In addition to the maximum and minimum values specified with the \\
\texttt{MAX\_VALUE} and \texttt{MIN\_VALUE} keywords, these plotting routines treat the IEEE \\
floating-point value NaN (Not A Number) as missing data automatically. (For \\
more information on NaN, see “Special Floating-Point Values” on page 152 of \\
Building IDL Applications.)

\section*{Using the AXIS Procedure}

The \texttt{AXIS} procedure draws and annotates an axis. It optionally saves the scaling \\
established by the axis for use by subsequent graphics procedures. It can be used \\
to add additional axes to plots or to draw axes at a specified position.
The AXIS procedure accepts the set of plotting keyword parameters that govern the scaling and appearance of the axes. Additionally, the keyword parameters XAXIS, YAXIS, and ZAXIS specify the orientation and position (if no position coordinates are present) of the axis. The value of these parameters are 0 for the bottom or left axis and 1 for the top or right. The tick marks and their annotation extend away from the plot window. For example, specify YAXIS = 1 to draw a y-axis on the right of the window.

The optional keyword parameter SAVE saves the data-scaling parameters established for the axis in the appropriate axis system variable, !X, !Y, or !Z. The call to AXIS is as follows:

AXIS[[, X, Y], Z], Keyword_Parameters

where X, Y, and optionally Z specify the coordinates of the axis. Any of the coordinate systems can be used by including the appropriate coordinate keyword in the call. The coordinate corresponding to the axis direction is ignored. When specifying an x-axis, the x coordinate parameter is ignored, but must be present if there is a y coordinate.

Example  The bottom left plot of Figure 11-12 illustrates using AXIS to draw axes with a different scale, opposite the main x and y axes. The plot is produced using PLOT with the bottom and left axes annotated and scaled in units of days and degrees Fahrenheit. The XMARGIN and YMARGIN keyword parameters are specified to allow additional room around the plot window for the new axes. The keyword parameters XSTYLE = 8 and YSTYLE = 8 inhibit drawing the top and right axes.

Next, the AXIS procedure is called to draw the top, XAXIS = 1, axis, labeled in months. Eleven tick intervals with 12 tick marks are drawn. The x value of each monthly tick mark is the day of the year that is approximately the middle of the month. Tick-mark names come from the MONTH string array.

The right y-axis, YAXIS = 1, is drawn in the same manner. The new y-axis range is set by converting the original y-axis minimum and maximum values, saved by PLOT in !Y.CRANGE, from Fahrenheit to Celsius, using the formula C = 5(F-32)/9. The keyword parameter YSTYLE = 1 forces the y-axis range to match the given range exactly. The program is as follows:

PLOT, DAY, TEMP, /YNOZERO, $      Plot the data, omit right and top axes.
SUBTITLE = 'Denver Average Temperature', $
XTITLE = 'Day of Year', $
YTITLE = 'Degrees Fahrenheit', $
XSTYLE=8, YSTYLE=8, XMARGIN=[8, 8], YMARGIN=[4, 4]
AXIS, XAXIS=1, XTICKS=11, XTICKV=DAY, XTICKN=MONTH, $
XTITLE='Month', XCHARS=0.7$

Draw the top x-axis, supplying labels, etc. Make the characters smaller so they will fit.

AXIS, YAXIS=1, YRANGE = (!Y.CRANGE-32)*5./9., YSTYLE = 1, $
YTITLE = 'Degrees Celsius'

Draw the right y-axis. Scale the current y-axis minimum values from Fahrenheit to Celsius and make them the new min and max values. Set YSTYLE=1 to make axis exact.

The code above is included in the batch file plot09 in the doc subdirectory of the examples subdirectory of the main IDL directory.

Using AXIS with Polar Plots

If the POLAR keyword parameter is set, the IDL PLOT procedure converts its coordinates from polar to cartesian coordinates when plotting. The first parameter to plot is the radius, R, and the second is the angle \( \theta \) (expressed in radians). Polar plots are produced using the standard axis and label styles, with box axes enclosing the plot area.

The bottom right plot in Figure 11-12 illustrates using AXIS to draw centered axes, dividing the plot window into the four quadrants centered about the origin. This method uses PLOT to plot the polar data and to establish the coordinate scaling, but suppresses the axes. Next, two calls to AXIS add the x and y axes, drawn through data coordinate (0, 0):

\[
R = \text{FINDGEN}(100) \quad \text{Make a radius vector.}
\]

\[
\text{THETA} = \frac{R}{5} \quad \text{Make a vector.}
\]

\[
\text{PLOT, R, THETA, SUBTITLE='Polar Plot', XSTY=4, YSTY=4, /POLAR}
\quad \text{Plot the data, suppressing the axes by setting their styles to 4.}
\]

\[
\text{AXIS, 0, 0, XAX=0}
\]

\[
\text{AXIS, 0, 0, YAX=0}
\quad \text{Draw the x and y axes through (0, 0).}
\]

The code above is included in the batch file plot09 in the doc subdirectory of the examples subdirectory of the main IDL directory.
Using the CURSOR Procedure

The CURSOR procedure reads the position of the interactive graphics cursor of the current graphics device. It enables the graphic cursor on the device, optionally waits for the user to move it and/or press a locator button to terminate the operation (or type a character if the device has no buttons), and then reports the cursor position.

Note, however, that CURSOR should not be used with draw widgets, created by the WIDGET_DRAW function. If you need to find the position of the mouse or status of mouse buttons in a draw widget, set the BUTTONEVENTS and MOTIONEVENTS keywords to WIDGET_DRAW, then examine the events returned by your draw widget. See “WIDGET_DRAW” on page 1252 of the IDL Reference Guide.

The CURSOR procedure is called as follows:

CURSOR, X, Y [, WAIT]

where x and y are the named variables that receive the cursor position. Normally, the position is reported in data coordinates, but the DATA, DEVICE, and NORMAL keywords can be used to explicitly specify the coordinate system.

See “CURSOR” on page 332 of the IDL Reference Guide for details.

When CURSOR returns, the button field of the system variable !MOUSE is set to the button status. Each mouse button is assigned a bit in the button field. Bit 0 is the leftmost button (value = 1), bit 1 is the middle button (value = 2), and bit 3 is the rightmost button (value = 4) for the typical three-button mouse. See “!MOUSE” on page 40 of the IDL Reference Guide for details.

Simple Interactive Examples

The following two programs demonstrate simple applications of the interactive graphics cursor and the CURSOR procedure. The code for both routines is located in the file plot10.pro, located in the doc subdirectory of the examples subdirectory of the main IDL directory. You can also create either routine at the IDL command line by starting with the .RUN command and entering each line individually.

The first routine is a simple drawing program. Straight lines are connected to positions marked with the left or middle mouse buttons until the right button is pressed.

PRO DRAW
ERASE

Start with a blank screen.
CURSOR, X, Y, /NORMAL, /DOWN

Get the initial point in normalized coordinates.

WHILE (!MOUSE.button NE 4) DO BEGIN

CURSOR, X1, Y1, /NORMAL, /DOWN

Get the second point.

PLOTS, X, X1, Y, Y1, /NORMAL

Draw the line.

X = X1 & Y = Y1

Make the current second point be the new first.

ENDWHILE

END

The second simple procedure can be used to label plots using the cursor to position the text:

PRO LABEL, TEXT

Text is the string to be written on the screen.

PRINT, 'Use the mouse to mark the text position:'

Ask the user to mark the position.

CURSOR, X, Y, /NORMAL, /DOWN

Get the cursor position after pressing any button.

XYOUTS, X, Y, TEXT, /NORMAL, /NOCLIP

Write the text at the specified position. The NOCLIP keyword is used to ensure that the text will appear even if it is outside the plotting region.

END

To place annotation on a device with an interactive pointer, call this procedure with the command:

ANNOTATE, 'Text for label'

Next, move the pointer device (mouse, cursor, or joystick) to the desired spot, and press the locator button. Consider how you might augment the LABEL procedure to allow you to specify the size and font of the annotation text.
Chapter 12
Plotting Multi-Dimensional Arrays

The following topics are covered in this chapter:

Contour Plots ........................................ 318
Overlaying Images and Contour Plots . 324
Additional Contour Options .......... 328
The SURFACE Procedure ................. 331
Three-Dimensional Graphics .......... 334
Three-Dimensional Transformations .... 343
Shaded Surfaces................................. 349
Volume Visualization ....................... 351
References ....................................... 355
This chapter describes the facilities for drawing representations of two-dimensional arrays. The two intrinsic procedures for the display of arrays are CONTOUR and SURFACE. Procedures for displaying two-dimensional arrays in the form of images, using color or grayscale pixels, are discussed in Chapter 14, “Image Display Routines”.

CONTOUR and SURFACE both use line graphics to depict the value of a two-dimensional array. As its name implies, CONTOUR draws contour plots. SURFACE depicts the surface created by interpreting each element value as an elevation. These three-dimensional, wire-mesh surface plots can have almost any rotation about the x and z axes (the data z axis must project parallel to the device’s y axis).

Three-dimensional graphics, coordinate systems, and transformations also are included in this chapter. Almost all of the information concerning coordinate systems, keyword parameters, and system variables discussed in Chapter 11, “Plotting”, apply to CONTOUR and SURFACE as well.

Direct Graphics vs. Object Graphics
The examples in this chapter are all written to take advantage of IDL Direct Graphics. Examples and techniques using IDL Object Graphics are contained in a separate volume: Objects and Object Graphics. See that volume for more information on IDL Object Graphics or see “Graphics” on page 267 of Building IDL Applications for a discussion of the differences between IDL Direct Graphics and IDL Object Graphics.

Running the Example Code
Some of the example code used in this chapter is part of the IDL distribution. All of the files mentioned are located in the doc subdirectory of the examples subdirectory of the main IDL directory. 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” on page 43 of the IDL Reference Guide for information on IDL's path.

Contour Plots
The CONTOUR procedure draws contour plots from data stored in a rectangular array. In its simplest form, CONTOUR makes a contour plot given a two-dimensional array of z values. In more complicated forms, CONTOUR accepts, in addition to z, arrays containing the x and y locations of each column, row, or point, plus many keyword parameters. In more sophisticated applications, the output of CONTOUR can be projected from three dimensions to two
dimensions, superimposed over an image, or combined with the output of SURFACE. The basic call to CONTOUR is as follows:

```
CONTOUR, Z
```

where Z is a two-dimensional array. This call labels the x and y axes with the subscript along each dimension. For example, when contouring a $10 \times 20$ array, the x axis ranges from 0 to 9, and the y axis ranges from 0 to 19.

You can explicitly specify the x and y locations of each cell as follows:

```
CONTOUR, Z, X, Y
```

where the x and y arrays can be either vectors or two-dimensional arrays of the same size as Z. If they are vectors, the element $z_{i,j}$ has a coordinate location of $(x_i, y_j)$. Otherwise, if the x and y arrays are two-dimensional, the element $z_{i,j}$ has the location $(x_{i,j}, y_{i,j})$. Thus, vectors should be used if the x location of $z_{i,j}$ does not depend upon $j$ and the y location of $z_{i,j}$ does not depend upon $i$.

Dimensions must be compatible. In the one-dimensional case, x must have a dimension equal to the number of columns in Z, and y must have a dimension equal to the number of rows in Z. In the two-dimensional case, all three arrays must have the same dimensions.

IDL uses linear interpolation to determine the x and y locations of the contour lines that pass between grid elements. The cells must be regular in that the x and y arrays must be monotonic over rows and columns, respectively. The lines describing the quadrilateral enclosing each cell and whose vertices are $(x_{i,j}, y_{i,j})$, $(x_{i,j+1}, y_{i+1,j})$, $(x_{i+1,j+1}, y_{i+1,j+1})$, and $(x_{i+1,j}, y_{i,j+1})$ must intersect only at the four corners and the quadrilateral must not contain other nodes.

See "CONTOUR" on page 300 of the IDL Reference Guide for a complete list of CONTOUR’s parameters and keywords.

**Contouring Methods**

In order to provide a wide range of options, CONTOUR uses one of two contouring algorithms. The algorithm used depends on the keywords specified, and is one of the following:

**Cell Drawing**

The first algorithm, used by default, examines each array cell and draws all contours emanating from that cell before proceeding to the next cell. This method is efficient in terms of computer resources, but does not allow options such as contour labeling or smoothing.
Contour Following

The second method searches for each contour line, then follows the line until it reaches a boundary or closes. This method gives better looking results with dashed linestyles and allows contour labeling and bi-cubic spline interpolation, but requires more computer time. The contour following method is used if any of these keywords are specified: C_ANNOTATION, C_CHARSIZE, C_LABELS, CLOSED, FOLLOW, PATH_FILENAME, or DOWNHILL.

Note Due to their differing algorithms, these two methods will often draw slightly different, yet correct, contour maps for the same data. This difference is a direct result of the fact that there is often more than one valid way to draw contours and should not be a cause for concern.

Example: Maroon Bells Peaks

Digital elevation data of the Maroon Bells area, near Aspen, Colorado, are used to illustrate the CONTOUR procedure. The data set was obtained from a United States Geological Survey Digital Elevation Model tape. This data provides terrain elevation data over a 7.5-minute square (approximately $11 \times 13.7$ kilometers at the latitude of Maroon Bells), with 30-meter sampling.

The data are contained in a 360 x 460 array A, sampled in 30-meter square intervals, measured in Universal Transverse Mercator (UTM) coordinates. The rectangular array is not completely filled with data because the 7.5-minute square is not perfectly oriented to the UTM grid system. Missing data are represented as zeroes. Elevation measurements range from 2658 to 4241 meters or from 8720 to 13,914 feet.

The Maroon Bells data is used in a number of examples in this chapter, and is included in an IDL SAVE file in the data subdirectory of the examples subdirectory of the main IDL directory. To restore the save file, issue the following commands at the IDL prompt (change the path separator characters as necessary for your platform):

```idl
CD, !DIR+'/examples/data'
RESTORE, 'marbells.dat'
```

The batch file cntour01, located in the doc subdirectory of the examples subdirectory of the main IDL directory, restores this data and defines several variables used in the examples in this chapter.

This command creates an IDL variable named elev that contains the 360 x 460 integer array.

Figure 12-1 is the result of applying the CONTOUR procedure to the data, using the default settings:
A number of problems are apparent.

- IDL selected six contour levels, by default, at \((4241 - 0)/7\)-meter intervals or approximately 605 meters. The levels are 605, 1250, ..., 3635 meters, even though the range of valid data is from 2658 to 4241 meters. This is because the missing data values of 0 were considered when selecting the intervals. It is more appropriate to select levels only within the range of valid data.

- The vertical contours along the left edge are an artifact due to contouring missing data and should not be present.

- For most display systems and for contour intervals of approximately 200 meters, the data has too many samples in the \(xy\) direction. This “oversampling” has two adverse effects: the contours appear jagged and a large number of short vectors are produced.

- The axes are labeled by point number, but should be in UTM coordinates.

- It is difficult to visualize the terrain and to discern maxima from minima because each contour is drawn with the same type of line.

The above problems are readily solved using the following simple techniques:
• Specify the contour levels directly using the LEVELS keyword parameter. Selecting contour intervals of 250 meters, at elevation levels of [2750, 3000, 3250, 3500, 3750, 4000], results in six levels.

• Change the missing data value to a value well above the maximum valid data value, then use the MAX_VALUE keyword parameter to exclude missing points. In this example, we set missing data values to one million with the following statement:

   `elev(WHERE(elev EQ 0)) = 1.0E6`

• Use the REBIN function to decrease the sampling in x and y by a factor of 5:

   `new = REBIN(elev, 360/5, 460/5)`

   This smooths the contours, because the call to REBIN averages \(5^2=25\) bins when resampling. The number of vectors transmitted to the display also are decreased by a factor of approximately 25. The variable `new` is now a 72 x 92 array.

Care is taken in the second step to ensure that the missing data are not confused with valid data after REBIN is applied. As in this example, REBIN averages bins of \(5^2=25\) elements, the missing data value must be set to a value of at least 25 times the maximum valid data value. After applying REBIN, any cell with a missing original data point will have a value of at least \(10^6/25 = 40000\), well over the largest valid data value of approximately 4,500.

The x and y vectors are constructed containing the UTM coordinates for each row and column. From the USGS data tape, the UTM coordinate of the lower-left corner of the array is (326,850: 4,318,500) meters. As the data spacing is 30 meters in both directions, the x and y vectors, in kilometers, are easily formed using the FINDGEN function, as shown in the example below.

Contour levels at each multiple of 500 meters (every other level) are drawn with a solid linestyle, while levels that fall between are drawn with a dotted line. In addition, the 4000-meter contour is drawn with a triple thick line, emphasizing the top contour.

The result of these improvements is shown in Figure 12-2. It was produced with the following IDL statements:

```idl
@cntour01
restore variables.

elev (WHERE (elev EQ 0)) = 1E6
   \text{Set missing data points to a large value.}

new = REBIN(elev, 360/5, 460/5)
   \text{REBIN down to a 72 x 92 matrix.}

X = 326.850 + .030 * FINDGEN(72)
   \text{Make the x and y vectors giving the position of each column and row.}
```
Using IDL

Contour Plots

Chapter 12: Plotting Multi-Dimensional Arrays

Using IDL

Contour Plots

Y = 4318.500 + .030 * FINDGEN(92)
CONTOUR, new, X, Y, LEVELS = 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], $
   TITLE = 'Maroon Bells Region', $
   SUBTITLE = '250 meter contours', $
   XTITLE = 'UTM Coordinates (KM)'

Make the plot, specifying the contour levels, missing data value, line

If you prefer not to enter the code by hand, run the batch file cntour02 with the
following command at the IDL prompt:

@cntour02

See “Running the Example Code” on page 318 if IDL does not find the batch file.
Figure 12-3 illustrates the data displayed as a grayscale image. Higher elevations are white. This image demonstrates that contour plots do not always provide the best qualitative visualization of many two-dimensional data sets.

Overlaying Images and Contour Plots

Superimposing an image and its contour plot combines the best of both worlds: the image allows easy visualization and the contour lines provide a semi-quantitative display. The programs presented in the rest of this section are for advanced computing only.

A combined contour and image display, such as that discussed in this section, can be created with the IMAGE_CONT procedure. The following material is intended to illustrate the ways in which images and graphics can be combined using IDL.

The technique used to overlay plots and images depends on whether or not the device is able to represent pixels of variable size, as does PostScript, or if it has pixels of a fixed size. If the device does not have scalable pixels, the image must be resized to fit within the plotting area if it is not already of a size suitable for viewing. This leads to three separate cases that are illustrated in the following examples.

Figure 12-3: Maroon Bells Data Displayed as an Image
Overlaying with Scalable Pixels

Certain devices, notably PostScript, can display pixels of varying sizes. With these devices, it is easy to set the size and position of an image so that it exactly overlays the plot window. In creating Figure 12-4, we used the actual dimensions of the contour plot window (contained in the !X.WINDOW and !Y.WINDOW system variables) to calculate the new size of the image.

**Note** In order to do this successfully, you must establish the size of the plot window before scaling the image. This means that you must make a call to CONTOUR before displaying the image, to set the window size, and another call to CONTOUR after displaying the image, to draw the contour lines on top of the image data.

Inspect the batch file cntour03 located in the doc subdirectory of the examples subdirectory of the main IDL directory to see how the figure was created. Note that the aspect ratio of the image was changed to fit that of the plot window. To retain the original image aspect ratio, the plot window must be resized to an identical aspect ratio using the POSITION keyword parameter.

Maroon Bells Region

Figure 12-4: Overlay of image and contour plots.
Overlaying with Fixed Pixels

If the pixel size is fixed (as is true on most displays), we can either resize the image to fit the plotting window or size the plotting window to fit the image dimensions.

Method 1: Scale the Image to Fit the Display

We can use the CONGRID function to create an image of the same size as the plotting window. The REBIN function also can be used to resample the original image if the plot window dimensions are an integer multiple or factor of the original image dimensions. REBIN is always faster than CONGRID. The following IDL procedure creates an image of the same size as the window, displays it, then overlays the contour plot.

```idl
@cntour01
Restore variables.
elev (WHERE (elev EQ 0)) = 1E6  Set missing data points to a large value.
new = REBIN(elev, 360/5, 460/5)  REBIN down to a 72 x 92 matrix.
image = BYTSCL(elev, MIN=2658, MAX=4241)  Scale image intensities.
CONTOUR, new, X, Y, LEVELS = 2750 + FINDGEN(6) * 250., MAX_VALUE = 5000, XSTYLE = 1, YSTYLE = 1,  Before displaying the image, use
TITLE = 'Maroon Bells Region', SUBTITLE = '250 meter contours', /NODATA
XTITLE = 'UTM Coordinates (KM)', /NODATA

PX = !X.WINDOW * !D.X_VSIZE  Get size of plot window in device pixels.
PY = !Y.WINDOW * !D.Y_VSIZE
SX = PX[1] - PX[0] + 1  Desired size of image in pixels.
SY = PY[1] - PY[0] + 1
TVSCL, CONGRID(image, SX, SY), PX[0], PY[0]  Display the image with its lower-
```

Overlaying Images and Contour Plots  Using IDL
CONTOUR, new, X, Y, LEVELS = 2750 + FINDGEN(6) * 250., $  
   MAX_VALUE = 5000, XSTYLE = 1, YSTYLE = 1,  
   TITLE = 'Maroon Bells Region', $  
   SUBTITLE = '250 meter contours', $  
   XTITLE = 'UTM Coordinates (KM)', /NOERASE  
   Write the contours over the image, being sure to use the exact axis styles so that the contours fill the plot window. Inhibit erasing.

If you prefer not to enter the code by hand, run the batch file cntour04 with the following command at the IDL prompt:

@cntour04

See “Running the Example Code” on page 318 if IDL does not find the batch file.

**Method 2: Scale the Display to Fit the Image**

If the image is already close to the proper display size, it is simpler and more efficient to change the plot window size to that of the image. The following procedure displays the image at the window origin, then sets the plot window to the image size, leaving its origin unchanged.

@cntour01  

```idl
restore variables.
elev (WHERE (elev EQ 0)) = 1E6  
set missing data points to a large value.
new = REBIN(elev, 360/5, 460/5)  
REBIN down to a 72 x 92 matrix.
image = BYTSCL(elev, MIN=2658, MAX=4241)  
Scale image intensities.
PX = !X.WINDOW * !D.X_VSIZE  
Get size of plot window in device pixels.
PY = !Y.WINDOW * !D.Y_VSIZE  
Get the size of the image.
SZ = SIZE(image)  
Display the image with its lower-left corner at the origin of the plot window.
TVSCL, image, PX[0], PY[0]

CONTOUR, new, X, Y, XSTYLE = 1, YSTYLE = 1, $  
```

left corner at the origin of the plot window and with its size scaled to fit the plot window.

CONTOUR, new, X, Y, LEVELS = 2750 + FINDGEN(6) * 250., $  
   MAX_VALUE = 5000, XSTYLE = 1, YSTYLE = 1,  
   TITLE = 'Maroon Bells Region', $  
   SUBTITLE = '250 meter contours', $  
   XTITLE = 'UTM Coordinates (KM)', /NOERASE  
   Write the contours over the image, being sure to use the exact axis styles so that the contours fill the plot window. Inhibit erasing.

If you prefer not to enter the code by hand, run the batch file cntour04 with the following command at the IDL prompt:

@cntour04

See “Running the Example Code” on page 318 if IDL does not find the batch file.
POSITION = [PX[0], PY[0], PX[0]+SZ[1]-1, PY[0]+SZ[2]-1], $ 
LEVELS = 2750 + FINDGEN(6) * 250., MAX_VALUE = 5000, $ 
TITLE = 'Maroon Bells Region', $ 
SUBTITLE = '250 meter contours', $ 
XTITLE = 'UTM Coordinates (KM)', /NOERASE, /DEVICE 
Write the contours over the image, being sure to use the exact axis styles so that the contours fill the plot window. Inhibit erasing.

If you prefer not to enter the code by hand, run the batch file cntour05 with the following command at the IDL prompt:

@cntour05

See “Running the Example Code” on page 318 if IDL does not find the batch file.

Of course, these procedures can be customized by using other keyword parameters with the CONTOUR procedure.

**Additional Contour Options**

In addition to the abilities of CONTOUR demonstrated above, there are several options that depend upon the use of the contour following algorithm. These options are as follows:

**Labeling Contours**

The C_ANNOTATION, C_CHARSIZE, and C_LABELS keywords are used to control contour labeling. Using them, possibly in conjunction with the LEVELS keyword, it is possible to specify which contours should be labeled, the size of the labels, and the actual labels that should be used.

In the following discussion, a variable named DATA is contoured. This variable contains uniformly distributed random numbers obtained using the following statement:

```idl
SEED = 20 & DATA = RANDOMU(SEED, 6, 6)
```

To label contours using the defaults for label size and contours to label, it is sufficient to select the FOLLOW keyword. In this case, CONTOUR labels every other contour using the default label size (three-fourths of the plot axis label size). Each contour is labeled with its value. Figure 12-5 was produced using the following statement:
CONTOUR, /FOllOW, DATA

The C_CHARSIZE keyword is used to specify the size of the characters used for labeling in the same manner that SIZE is used to control plot axis label size. The C_LABELS keyword can be used to select the contours to be labeled. For example, suppose that we want to contour the variable DATA at 0.2, 0.5, and 0.8, and we want all three levels labeled. In addition, we wish to make each label larger, and use hardware fonts. This can be accomplished with the statement below.

CONTOUR, LEVEL=[0.2, 0.5, 0.8], C_LABELS=[1, 1, 1], $ 
C_CHARSIZE = 1.25, DATA, FONT = 0

The result is the plot on the left in Figure 12-6.

Finally, it is possible to specify the text to be used for the contour labels using the C_ANNOTATION keyword, as shown in the statements below.

CONTOUR, LEVEL=[0.2, 0.5, 0.8], C_LABELS=[1, 1, 1], $ 
C_ANNOTATION = ["Low", "Medium", "High"], DATA, FONT=0

The result is the plot on the right in Figure 12-6.

Smoothing Contours

The MIN_CURVE_SURF function can be used to smoothly interpolate both regularly and irregularly sampled surfaces before contouring. This function replaces the older SPLINE keyword to CONTOUR, which was inaccurate and is
Additional Contour Options

Using IDL

MIN_CURVE_SURF interpolates the entire surface to a relatively fine grid before drawing the contours. See “CONTOUR” on page 300 of the IDL Reference Guide for an example using the MIN_CURVE_SURF function. See also “MIN_CURVE_SURF” on page 748 of the IDL Reference Guide for further details.

The following short example shows the difference between a smoothed and an unsmoothed contour plot:

```idl
data = RANDOMU(seed, 7, 7)       ; Create a simple dataset.
CONTOUR, data                   ; Plot the unsmoothed data.
CONTOUR, MIN_CURVE_SURF(data)    ; Plot the smoothed data.
```

Filling Contours

Set the FILL keyword to produce a filled contour plot. The contours are filled with solid or line-filled polygons. For solid polygons, use the C_COLOR keyword to specify the color index of the polygons for each contour level. For line fills, use C_ORIENTATION, C_SPACING, C_COLOR, C_LINESTYLE, and/or C_THICK to specify attributes for the lines.

If the current device is not a pen plotter, each polygon is erased to the background color before the fill lines are drawn, to avoid superimposing one pattern over another.

The FILL keyword replaces the use of the PATH_FILENAM E keyword and POLYFILL procedure from previous versions of IDL. Setting the FILL keyword also closes any open contours before filling.
The following example illustrates various filled contour plot options.

```
data = RANDOMU(seed, 7, 7)
CONTOUR, data, NLEVELS=6, /FILL
CONTOUR, data, NLEVELS=6, /NOERASE
```

Instead of solid colors, contours can be filled with lines:

```
ANGLES = [0, 45, -45]
C = [70, 120, 200, 255]
CONTOUR, data, NLEVELS=10, C_ORIENT=ANGLES, C_COLORS=C
CONTOUR, data, NLEVELS=10, /NOERASE
```

There are many other controls for filled contour plots. The C_COLORS, C_LINESTYLE, C_SPACING, and C_THICK keywords can also be used to control the type of fill. For a complete description of CONTOUR, see the IDL Reference Guide.

**Indicating Direction of Grade**

Setting the DOWNHILL keyword creates short, perpendicular tick marks along each contour that point in the “downhill” direction. These marks make the direction of the grade readily apparent. For example:

```
CONTOUR, data, /DOWNHILL
```

**The SURFACE Procedure**

The SURFACE procedure draws “wire mesh” representations of functions of x and y, just as CONTOUR draws their contours. Parameters to SURFACE are similar to CONTOUR. SURFACE accepts a two-dimensional array of z (elevation) values, and optionally x and y parameters indicating the location of each z element.

**Note** The grid defined by the x and y parameters must be regular, or nearly regular, or errors in hidden line removal will result. Also, the rotation must project the data z axis so that it is parallel to the drawing surface’s y axis or errors in hidden line removal will result.
SURFACE projects the three-dimensional array of points into two dimensions after rotating about the $z$ and then the $x$ axes. Each point is connected to its neighbors by lines. Hidden lines are suppressed. The rotation about the $x$ and $z$ axes can be specified with keywords or a complete three-dimensional transformation matrix can be stored in the field !P.T for use by SURFACE. Details concerning the mechanics of three-dimensional projection and rotation are covered in the next section.

The following IDL code illustrates the most basic call to SURFACE. It produces a two-dimensional Gaussian function, then calls SURFACE to produce Figure 12-7.

\[
Z = \text{SHIFT(DIST}(40), 20, 20) \\
Z = \exp(-{(Z/10)^2}) \\
\text{SURFACE, Z}
\]

Create a 40 by 40 array in which each element is equal to the Euclidean distance from the center.

Make Gaussian with a 1/e width of 10.

Call SURFACE to display plot.

In the example above, the DIST function creates an ($n, n$) array in which each element is set to its Euclidean distance from the origin.
Chapter 12: Plotting Multi-Dimensional Arrays

Using IDL

The SURFACE Procedure

SURFACE Keyword Parameters

In addition to the standard graphics keyword parameters, SURFACE accepts a number of unique keyword parameters. See “SURFACE” on page 1090 of the IDL Reference Guide for details.

Example

Figure 12-8 illustrates the application of the SURFACE procedure to the Maroon Bells data used in the first section of this chapter. As with CONTOUR, it is often useful to reduce the number of individual data values, so that the surface is not obscured by excessive detail. The left illustration in Figure 12-8 was produced by the following statements:

```idl
@cntour01
surf = REBIN(elev > 2650, 360/5, 460/5)  
SURFACE, surf, X, Y, SKIRT = 2650
```

Restore variables.
Resize the original data into a 72 x 92 array, setting all data values which are less than 2650 (the lowest elevation we wish to show) to 2650.
Display the surface, drawing a “skirt” down to 2650 meters.

Alternatively, run the batch file surf01 with the following command at the IDL prompt:

```
@surf01
```
See “Running the Example Code” on page 318 if IDL does not find the batch file.

The right illustration in Figure 12-8 shows the Maroon Peaks area looking from the back row to the front row (north to the south) of the Maroon Peaks area. This perspective on the data is created by setting the angle of rotation around the z axis to 210 degrees (setting $AZ = 210$), and increasing the azimuth from the default 30 degrees to 45 (setting $AX = 45$). Also, only the horizontal lines are drawn because the /HORIZONTAL keyword is present in the following call:

```
SURFACE, surf, X, Y, SKIRT = 2650, /HORIZ, AZ = 210, AX = 45
```

Because the axes are rotated 210 degrees about the original z axis, the annotation is reversed and the x axis is behind and obscured by the surface. This undesirable effect can be eliminated by reversing the minimum and maximum values of the X and Y ranges used when drawing the surface:

```
SURFACE, surf, X, Y, SKIRT = 2650, /HORIZONTAL, AX = 45, $YRANGE = [\text{MAX}(Y), \text{MIN}(Y)], \text{XRANGE} = [\text{MAX}(X), \text{MIN}(X)]
```

As above, but reverse the data rather than the axes.

**Three-Dimensional 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 (1982). 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.

**Homogeneous Coordinates**

A point in homogeneous coordinates is represented as a four-element column vector of three coordinates and a scale factor $w \neq 0$. 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

Note For most applications, it is not necessary to create, manipulate, or to even understand transformation matrices. The procedure T3D, explained below, implements most of the common transformations.

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 $A_1$, $A_2$, and $A_3$ are transformation matrices to be applied in order, and the matrix $A$ is the product of the three matrices, the following applies.

$((P \cdot A_1) \cdot A_2) \cdot A_3 \equiv P \cdot ((A_1 \cdot A_2) \cdot A_3) = P \cdot A$

IDL stores the concatenated transformation matrix in the system variable field !P.T.

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.

$$
\begin{bmatrix}
1 & 0 & 0 & D_x \\
0 & 1 & 0 & D_y \\
0 & 0 & 1 & D_z \\
0 & 0 & 0 & 1
\end{bmatrix}
$$

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.
Rotation

Rotation about the x, y, and z axes is represented respectively by the following three matrices:

\[
R_x = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & \cos \theta_x & -\sin \theta_x & 0 \\
0 & \sin \theta_x & \cos \theta_x & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

\[
R_y = \begin{bmatrix}
\cos \theta_y & 0 & \sin \theta_y & 0 \\
0 & 1 & 0 & 0 \\
-\sin \theta_y & 0 & \cos \theta_y & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

\[
R_z = \begin{bmatrix}
\cos \theta_z & -\sin \theta_z & 0 & 0 \\
\sin \theta_z & \cos \theta_z & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

T3D Procedure

The IDL procedure T3D creates and accumulates transformation matrices, storing them in the system variable field !P.T. The procedure can be used to create a transformation matrix composed of any combination of translation, scaling,
rotation, perspective projection, oblique projection, and axis exchange. Transformations are applied in the order of the keyword descriptions below:

**RESET**
Set this keyword to reset the transformation matrix to the identity matrix to begin a new accumulation of transformations. If this keyword is not present, the current transformation matrix !P.T is post-multiplied by the new transformation. The final transformation matrix is always stored back in !P.T.

**TRANSLATE**
This keyword argument accepts a 3-element vector. The viewpoint is translated by the three-element vector \([T_x, T_y, T_z]\).

**SCALE**
This keyword argument accepts a 3-element vector. The viewing area is scaled by factor \([S_x, S_y, S_z]\).

**ROTATE**
This keyword accepts a 3-element vector. The viewing area is rotated about each axis by the amount \([\theta_x, \theta_y, \theta_z]\), in degrees.

**PERSPECTIVE**
A scalar \((p)\) indicating the \(z\) distance of the center of the projection in the negative direction. Objects are projected into the \(xy\) plane, at \(z = 0\), and the “eye” is at point \((0, 0, -p)\).

**OBLIQUE**
A two-element vector, \([d, \alpha]\), specifying the parameters for an oblique projection. Points are projected onto the \(xy\)-plane at \(z = 0\) as follows:
\[
\begin{align*}
x_0 &= x + z(d \cos \alpha) \\
y_0 &= y + z(d \sin \alpha)
\end{align*}
\]
An oblique projection is a parallel projection in which the normal to the projection plane is the \(z\) axis, and the unit vector \((0, 0, 1)\) is projected to \((d \cos \alpha, d \sin \alpha)\) where \(\alpha\) is expressed in degrees.

**XYEXCH**
If set, exchanges the \(x\) and \(y\) axes.

**XZEXCH**
If set, exchanges the \(x\) and \(z\) axes.
YZEXCH
   If set, exchanges the y and z axes.

Example: The Transformation Created by SURFACE
   The SURFACE procedure creates a transformation matrix from its keyword
   parameters AX and AZ as follows:
   1. Starting with the identity transformation, SURFACE translates the center of the
      normalized cube to the origin.
   2. SURFACE rotates 90 degrees about the x axis to make the +z-axis of the data the
      +y axis of the display. The +y data axis extends from the front of the display to the
      rear.
   3. SURFACE rotates AZ degrees about the y axis. This rotates the result counter-
      clockwise, as seen from above the page.
   4. SURFACE rotates AX degrees about the x axis, tilting the data towards the viewer.
   5. The procedure then removes the translation applied in the first step and scales the
      data so that the data are still contained within the normal coordinate unit cube
      after transformation.

   These transformations can be created using T3D as shown below. The SCALE3
   procedure, documented in the IDL Reference Guide, mimics the transformation
   matrix created by SURFACE using the following method:

   T3D, /RESET, TRANSLATE = [-.5, -.5, -.5]
      Translate to move center of cube to origin.

   T3D, ROTATE = [-90, AZ, 0]
      Rotate 90 degrees about x axis, so
      +z axis is now +y. Then rotate AZ
      degrees about y axis.

   T3D, ROTATE = [AX, 0, 0]
      Rotate AX about x axis.

   T3D, TRANSLATE = [0.5, 0.5, 0.5]
      Restore origin.

   The SCALE3 procedure, documented in the IDL Reference Guide, scales the unit
   cube by a fixed factor, 1/\sqrt{3} to ensure that the corners of the rotated cube fit
   within the drawing area. If requested, it also will set the data scaling. Animations
   involving rotations or the SURFACE procedure should have their scaling and
   viewing transformation set by SCALE3 rather than the obsolete SURFR
   procedure, so that the scaling does not vary between frames.
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 \( N_x = X_0 + X_1D_x \). 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 \( !P.T \):

\[
P' = P \times !P.T
\]

- The vector \( P' \) is scaled by dividing by \( w \), and the normalized two-dimensional coordinates are extracted:

\[
N'_x = P'_x / P'_w \quad \text{and} \quad N'_y = P'_y / P'_w
\]

- The normalized \( xy \) coordinate is converted to device coordinates as described in “Two-Dimensional Coordinate Conversion” on page 286.

The `CONVERT_COORD` function performs the above process when converting to and from coordinate systems when the `T3D` keyword is specified. For example, if a three-dimensional coordinate system is established, then the device coordinates of the data point \((0, 1, 2)\) can be computed as follows:

\[
D = \text{CONVERT\_COORD}(0, 1, 2, /TO\_DEVICE, /T3D, /DATA)
\]

On completion, the three-element vector \( D \) will contain the desired device coordinates. The process of converting from three-dimensional to two-dimensional coordinates also can be written as an IDL function:

```idl
FUNCTION CVT_TO_2D, X, Y, Z


P = P \times !P.T

RETURN, [N'_x, N'_y]
```

Accept a three-dimensional data coordinate, return a two-element vector containing the coordinate transformed to two-dimensional normalized coordinates using the current transformation matrix.

Make a homogeneous vector of normalized three-dimensional coordinates.

Transform by \( !P.T \).
RETURN, [P[0] / P[3], P[1] / P[3]]  \hspace{1cm} \text{Return the scaled result as a two-element, two-dimensional, xy-vector.}

END

**Establishing a Three-Dimensional Coordinate System**

Usually, scaling parameters for coordinate conversion are set up by the higher-level procedures. To set up your own three-dimensional coordinate system with a given transformation matrix and x, y, z data range, follow these steps:

- Establish the scaling from your data coordinates to normalized coordinates—the (0,1) cube. Assuming your data are contained in the range \((X_{\text{min}}, Y_{\text{min}}, Z_{\text{min}})\) to \((X_{\text{max}}, Y_{\text{max}}, Z_{\text{max}})\), set the data scaling system variables as follows:
  \[
  !X.S = \left[ \begin{array}{c} -X_{\text{min}} \\ 1 \end{array} \right] / (X_{\text{max}} - X_{\text{min}})
  \]
  \[
  !Y.S = \left[ \begin{array}{c} -Y_{\text{min}} \\ 1 \end{array} \right] / (Y_{\text{max}} - Y_{\text{min}})
  \]
  \[
  !Z.S = \left[ \begin{array}{c} -Z_{\text{min}} \\ 1 \end{array} \right] / (Z_{\text{max}} - Z_{\text{min}})
  \]

- Establish the transformation matrix that determines the view of the unit cube. This can be done by either calling T3D, as explained above or by directly manipulating !PT yourself. If you wish to simply mimic the rotations provided by the SURFACE procedure, call the SCALE3 procedure (which can also be used to perform the previous step).

**Example**

This example draws four views of a simple house. The procedure HOUSE defines the coordinates of the front and back faces of the house. The data-to-normal coordinate scaling is set, as shown above, to a volume about 25 percent larger than that enclosing the house. The PLOTS procedure is called to draw lines describing and connecting the front and back faces. XYOUTS is called to label the front and back faces.

The commands shown after the definition of the HOTUS procedure contain four sequences of calls to T3D to establish the coordinate transformation, each followed by a call to HOUSE. If you prefer not to enter the IDL code by hand, run the batch file showhaus with the following command at the IDL prompt:

```
@showhaus
```

See “Running the Example Code” on page 318 if IDL does not find the batch file.

**PRO HOUSE**

Define a procedure to draw a house.
Chapter 12: Plotting Multi-Dimensional Arrays

house_x = [0, 16, 16, 8, 0, 0, 16, 16, 8, 0]  
X coordinates of 10 vertices. First 5 are front face, second 5 are back face. The range is 0 to 16.

house_y = [0, 0, 10, 16, 10, 0, 0, 10, 16, 10]  
The corresponding y values range from 0 to 16.

house_z = [54, 54, 54, 54, 54, 30, 30, 30, 30, 30]  
The z values range from 30 to 54.

min_x = -4 & max_x = 20.  
Define max and min xy values to scale. Slightly larger than data range.

!X.S = [(-(-4), 1.]/(20 - (-4))  
Set x data scale to range from −4 to 20.

!Y.S = !X.S  
Same for y.

!Z.S = [-10, 1.]/(70 - 10)  
The z range is from 10 to 70.

face = [INDGEN(5), 0]  
Indices of front face.

PLOTS, house_x[face], house_y[face], $  
house_z[face], /T3D, /DATA  
Draw front face.

PLOTS, house_x[face + 5], house_y[face + 5], $  
house_z[face + 5], /T3D, /DATA  
Draw back face.

FOR I = 0, 4 DO PLOTS, [house_x[i], house_x[i + 5]], $  
[house_y[i], house_y[i + 5]], $  
[house_z[i], house_z[i + 5]], /T3D, /DATA  
Connecting lines from front to back.

XYOUTS, house_x[3], house_y[3], Z = house_z[3], 'Front', $  
/T3D, /DATA, SIZE = 2  
Annotate front peak.

XYOUTS, house_x[8], house_y[8], Z = house_z[8], 'Back', $  
/T3D, /DATA, SIZE = 2  
Annotate back.

END  
The HOUSE procedure could be called from the IDL command line to produce a number of different plots. For example:
T3D, /RESET & HOUSE

H = [0.5, 0.5, 0.5]

T3D, /RESET, TRANS = -H, ROT = [30, 30, 0] & $

T3D, TR = H & HOUSE

Set up no rotation, scale, and draw house.

Create a handy constant.

Straight projection after rotating 30 degrees about x and y axes.

T3D, /RESET, TRANS = -H, ROT=[0, 0, 0], OBLIQUE=[.5, -45] & $

T3D, TR = H & HOUSE

No rotation, oblique projection, z factor = 0.5, angle = 45.

Figure 12-9 illustrates the different transformations. The four rotations are:

- Upper left: no rotation, plain projection
- Upper right: oblique projection, factor = 0.5, angle = -45
- Bottom left: rotation of 30 degrees about both the x and y axes, plain projection

Figure 12-9: Illustration of Different Three-Dimensional Transformations
Rotating the House

A common procedure for visualizing three-dimensional data is to animate the data by rotating it about one or more axes. To make an animation of the house in the preceding example with the XINTERANIMATE procedure, use the following example.

```
sizx = 300
sizy = 300
nframes = 16
XINTERANIMATE, SET=[sizx, sizy, nframes]
```

```
FOR i = 0, nframes - 1 DO BEGIN
  SCALE3, AX = 75, AZ = i * 360. / nframes &
  Rotate about the z axis.
  ERASE &
  HOUSE &
  Draw the house.
  SCALE3, AX = 75, AZ = i * 360. / nframes &
  XINTERANIMATE, FRAME=i, WINDOW=!D.WINDOW &
  Save the window.
ENDFOR
XINTERANIMATE
```

In the above example, SCALE3 rather than SCALE3D is used to maintain the same scaling in all rotations. If you prefer not to enter the IDL code by hand, run the batch file `animhaus` with the following command at the IDL prompt:

```
@animhaus
```

See “Running the Example Code” on page 318 if IDL does not find the batch file.

Three-Dimensional Transformations

The CONTOUR and PLOT procedures output their results using the three-dimensional coordinate transformation contained in !P.T when the keyword T3D is specified. Note that !P.T must contain a valid transformation matrix prior to using the T3D keyword.
PLOT and its variants output graphs in the xy-plane at the normal coordinate z value given by the keyword ZVALUE. If this keyword is not specified, the plot is drawn at the bottom of the unit cube at z = 0.

CONTOUR draws its axes at z = 0 and its contours at their z data value if ZVALUE is not specified. If ZVALUE is present, CONTOUR draws both the axes and contours in the xy-plane at the given z value.

**Combining CONTOUR and SURFACE**

It is easy to combine the results of SURFACE with the other IDL graphics procedures. The keyword parameter SAVE causes SURFACE to save the graphic transformation it used in !P.T. Then, when either CONTOUR or PLOT is called with the keyword parameter T3D, its output is transformed with the same projection. For example, Figure 12-10 illustrates combining SURFACE with CONTOUR. In essence, this is a combination of Figure 12-2 and Figure 12-8. Using the same variables as in the first two sections of this chapter, the figure was produced with the following statements:

```
@contour01
restore variables.
surf = REBIN(elev > 2650, 360/5, 460/5)
```

Resize the original data into a 72
Chapter 12: Plotting Multi-Dimensional Arrays

x 92 array, setting all data values which are less than 2650 (the lowest elevation we wish to show) to 2650.

SURFACE, surf, X, Y, SKIRT=2650, /SAVE

Make the mesh as in Figure 12-8.

CONTOUR, surf, X, Y, /T3D, /NOERASE, TITLE = 'Contour Plot', $ MAX_VAL = 5000., ZVALUE = 1.0, /NOCLIP, $ LEVELS = 2750. + FINDGEN(6) * 250 Specify T3D to align with SURFACE, at ZVALUE of 1.0. Suppress clipping as the plot is outside the normal plot window.

More Complicated Transformations

Figure 12-11 illustrates the application of three-dimensional transforms to the output of CONTOUR and PLOT. Using the two-dimensional Gaussian array \( z \) defined in “The SURFACE Procedure” on page 331, it draws a three-dimensional contour plot with the contours stacked above the axes in the \( z \) direction. It then plots the sum of the columns, also a Gaussian, in the \( xz \)-plane, and the sum of the rows in the \( yz \) plane. It was constructed as follows:

Figure 12-11: Example of PLOT and CONTOUR with a Three-dimensional Transform
First, the SCALE3 procedure is called to establish the default three- to two-dimensional transformation used by SURFACE, as explained above. The default rotations are 30 degrees about both the x and z axes.

Next, a vector, POS, defining the cube containing the plot window is defined in normalized coordinates. The cube extends from 0.1 to 1.0 in the x and y directions and from 0 to 1 in the z direction. Each call to CONTOUR and PLOT must explicitly specify this window to align the plots. This is necessary because the default margins around the plot window are different in each direction.

CONTOUR is called to draw the stacked contours with the axes at z=0. Clipping is disabled to allow drawing outside the default plot window, which is only two-dimensional.

The procedure T3D is called to exchange the y and z axes. The original xyz coordinate system is now xzy.

PLOT is called to draw the column sums which appear in front of the contour plot. The expression Z#REPLICATE(1., Ny) creates a row vector containing the sum of each row in the two-dimensional array z. The NOERASE and NOCLIP keywords are specified to prevent erasure and clipping. This plot appears in the xz-plane because of the previous axis exchange.

T3D is called again to exchange the x and z axes. This makes the original xyz coordinate system, which was converted to xzy, now correspond to yzx.

PLOT is called to produce the column sums in the yz-plane in the same manner as the first plot. The original x axis is drawn in the y-plane, and the y axis is in the z plane. One unavoidable side effect of this method is that the annotation of this plot is backwards. If the plot is transformed so the letters read correctly, the x axis of the plot would be reversed in relation to the y axis of the contour plot.

The IDL code used to draw Figure 12-11 is as follows:

```
Z = SHIFT(DIST(40), 20, 20)                # Create the Z variable.
Z = EXP(-(Z/10)^2)                        # NX and NY are the X and Y dimensions of the Z array.
NX = (SIZE(Z))(1)
NY = (SIZE(Z))(2)
SCALE3                                    # Set up IP.T with default SURFACE transformation.
POS= [.1, .1, 1, 1, 0, 1]                 # Define the three-dimensional plot window: x = 0.1 to 1, y = 0.1 to 1, and z = 0 to 1.
```
CONTOUR, Z, /T3D, NLEVELS=10, /NOCLIP, POSIT=POS, CHARSIZE=2
Make the stacked contours. Use 10 contour levels.

T3D, /YZEXCH
Swap y and z axes. The original xyz system is now xzy.

PLOT, Z#REPLICATE(1., NY), /NOERASE, /NOCLIP, /T3D, $
TITLE='COLUMN SUMS', POSITION = POS, CHARSIZE = 2
Plot the column sums in front of the contour plot.

T3D, /XZEXCH
Swap x and z—original xyz is now yzx.

PLOT, REPLICATE(1., NX)#Z, /NOERASE, /T3D, /NOCLIP, $
TITLE = 'ROW SUMS', POSITION = POS, CHARSIZE = 2
Plot the row sums along the right side of the contour plot.

If you prefer not to enter the IDL code by hand, run the batch file cntour06 with the following command at the IDL prompt:

@cntour06

See “Running the Example Code” on page 318 if IDL does not find the batch file.

Combining Images with Three-Dimensional Graphics
Images are combined with three-dimensional graphics, as shown in Figure 12-12, using the transformation techniques described above. The rectangular image must be transformed so that it fits underneath the mesh drawn by SURFACE. The general approach is as follows:

- Use SURFACE to establish the general scaling and geometrical transformation. Draw no data, as the graphics made by SURFACE will be over-written by the transformed image.

- For each of the four corners of the image, translate the data coordinate, which is simply the subscript of the corner, into a device coordinate. The data coordinates of the four corners of an (m, n) image are (0, 0), (m-1, 0), (0, n-1), and (m-1, n-1). Call this data coordinate system (x, y). Using a procedure or function similar to CVT_TO_2D (see page 339) convert to device coordinates, which in this discussion are called (U, V).

- The image is transformed from the original xy coordinates to a new image in UV coordinates using the POLY_2D function. POLY_2D accepts an input image and
the coefficients of a polynomial in UV giving the xy coordinates in the original image. The equations for x and y are below.

\[
X = S_{0,0} + S_{1,0}U + S_{1,1}UV \\
Y = T_{0,0} + T_{1,0}U + T_{1,1}UV
\]

We solve for the four unknown S coefficients using the four equations relating the x corner coordinates to their U coordinates. The T coefficients are similarly found using the y and V coordinates. This can be done using matrix operators and inversion or more simply, with the procedure POLY_WARP.

- The new image is a rectangle that encloses the quadrilateral described by the UV coordinates. Its size is specified in the formula below:

\[
\text{MAX}(U) - \text{MIN}(U) + 2, \text{MAX}(V) - \text{MIN}(V) + 1
\]

- POLY_2D is called to form the new image which is displayed at device coordinate \((\text{MIN}(U), \text{MIN}(V))\).

- SURFACE is called once again to display the mesh surface over the image.
Finally, \texttt{CONTOUR} is called with \texttt{ZVALUE} set to 1.0, placing the contour above both the image and the surface.

The \texttt{SHOW3} procedure performs these operations. It should be examined for details of how images and graphics can be combined.

The following IDL commands were used to create Figure 12-12:

\begin{verbatim}
@cntour01
new = REBIN(elev, 360/5, 460/5) # Reduce the size of elev array.
levs = (FINDGEN(10)*100)+3500    # Create an array of levels for CONTOUR.
SHOW3, new, E_SURFACE={min:2000}, E_CONTOUR={levels:levs} # Use SHOW3. Note the use of keywords E_SURFACE and E_CONTOUR to pass values to the SURFACE and CONTOUR routines used within SHOW3.
\end{verbatim}

### Shaded Surfaces

The \texttt{SHADE_SURF} procedure creates a shaded representation of a surface made from regularly gridded elevation data. The shading information can be supplied as a parameter or computed using a light-source model. Displays are easily constructed depicting the surface elevation of a variable shaded as a function of itself or another variable. This procedure is similar to the \texttt{SURFACE} routine, but it renders the visible surface as a shaded image rather than a mesh.

Parameters are identical to those of the \texttt{SURFACE} procedure. See “SHADE_SURF” on page 1004 of the IDL Reference Guide for details.

### 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 (1982, Chapter 19):
\[ I = I_a + d \cdot I_p(L \cdot N) \]

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.
- \( I_p(L \cdot N) \): Term due to diffuse reflection. The reflected light is proportional to the cosine of the angle between the surface normal vector \( N \) and the vector pointing to the light source, \( L \). \( I_p \) 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.

**Shading Parameters**

Parameters affecting the method of shading interpolation, light source direction, and rejection of hidden faces are set with the `SET_SHADING` procedure, described in the IDL Reference Guide. Defaults are Gouraud interpolation, light-source direction \([0, 0, 1] \), and rejection of hidden faces enabled.

See the description of `SET_SHADING` in the IDL Reference Guide for a more complete description of the parameters. Note that the REJECT keyword has no effect on the output of SHADE_SURF—it is used only with solids.

**Examples Using SHADE_SURF**

Figure 12-13 illustrates the application of SHADE_SURF, with light-source shading, to the two dimensional Gaussian, drawn as a mesh in Figure 12-7. This figure was produced by the following statements.

```idl
Z = SHIFT(DIST(40), 20, 20)  # Create a 40 by 40 array in which each element is equal to the Euclidean distance from the center.
Z = EXP(-(Z/10)^2)           # Make Gaussian with a 1/e width of 10.
SHADE_SURF, Z                # Show Gaussian with shades created by scaling elevation into the range of bytes.
```

The right half of Figure 12-13 shows the use of an array of shades, which in this case is simply the surface elevation scaled into the range of bytes. The output of SURFACE is superimposed over the shaded image with the statements below.

```idl
SHADE_SURF, Z, SHADES=BYTSCL(Z, TOP = !D.TABLE_SIZE)  # Show Gaussian with shades created by scaling elevation into the range of bytes.
```
Volume Visualization

A common problem in data visualization is how to display a constant density surface (also known as an isosurface), given a three-dimensional grid of density measurements. In medical imaging, stacking a series of two-dimensional images created by computed tomography or magnetic resonance creates a grid of density measurements that can be contoured to display the surfaces of anatomical structures. Atmospheric scientists create three-dimensional grids of water densities that can be contoured at the proper density level to show the surface of clouds. It is relatively easy to produce these surfaces using the SHADE_VOLUME procedure in conjunction with the POLYSHADE function.
SHADE_VOLUME accepts a three-dimensional grid of densities and a contour level. It outputs the set of polygons that describe the surface of the contour. The polygons are described by a \((3, n)\) array of vertices and a polygon list array that contains the vertices belonging to each polygon. Given a volume array with dimensions of \((D_0, D_1, D_2)\), the resulting vertex coordinates range between 0 and \(D_0 - 1\) in \(x\), 0 and \(D_1 - 1\) in \(y\), and 0 and \(D_2 - 1\) in \(z\). Keyword parameters to SHADE_VOLUME include the following:

**LOW**
A flag that indicates which side of the contour surface is to be viewed: 1 for the high side and 0 for the low (the default). If the contour to be viewed encloses high data values, as in the cloud example on 353, set the LOW keyword parameter to 1.

**SHADES**
An array of shading values for each volume element (voxel). On completion, SHADE_VOLUME replaces this array with the interpolated shading for each vertex of the surface.

These polygons are then fed to the POLYSHADE function to produce the shaded surface representation. It must be noted that the maximum volume size and polygon complexity are limited by the amount of available memory, as these routines store the density measurements, vertex list, and polygon list in memory.

For example, Figure 12-15 produced by the following IDL code, shows the three-dimensional contour surface of the precipitating region of a thunderstorm simulated by a three-dimensional cloud model. The data were provided by the
National Center for Atmospheric Research. The original data are contained in an array called clouds, a (55, 55, 32) element floating-point array. Each array element contains the amount of water contained in the corresponding volume of air.

RESTORE, FILEPATH('clouds3d.dat', SUBDIR=['examples','data'])

Restore the data.

SHADE_VOLUME, clouds, 0.1, v, p, /LOW

Create the contour surface polygons (v and p) at density 0.1, from clouds. Show the low side.

s = SIZE(clouds)

Obtain the dimensions of the volume. Variables S[1], S[2], and S[3] now contain the number of columns, rows, and slices in the volume.
SCALE3, XRANGE=[0,S[1]], YRANGE=[0,S[2]], ZRANGE=[0,S[3]], AX=0, AZ=45

Use SCALE3 to establish the three-dimensional transformation matrix. Rotate 45 degrees about the z-axis.

TV, POLYSHADE(v, p, /T3D)

Render and display the polygons.

If you prefer not to enter the IDL code by hand, run the batch file clouds with the following command at the IDL prompt:

@clouds

See “Running the Example Code” on page 318 if IDL does not find the batch file.

The shaded volume can be viewed from different rotations by changing the three-dimensional transformation matrix, !P.T, and calling POLYSHADE for each view. The following code will display 20 views of the volume, each separated by 18 degrees.

nframes = 20

Define number of views.

FOR i = 0, nframes - 1 DO BEGIN &

T3D, TR=[-.5, -.5, -.5], ROT=[0, 360./NFRAMES, 0] &

Translate the center of the (0, 1) unit cube to (0,0) and rotate about the x axis.

T3D, TR=[.5, .5, .5] &

Translate the center back to (0.5, 0.5, 0.5).

TV, POLYSHADE(v, p, /T3D) &

Show the surface.

ENDFOR

The animation rate of the above loop will not be very fast, especially with a larger number of polygons. Each image could be saved for rapid replay by writing it to a disk file. Given enough memory and/or display resources, the XINTERANIMATE procedure could be used to animate the views.

The SLICER3 Tool

IDL also includes an interactive volume visualization tool called SLICER3. This tool can be used to view isosurfaces and slices of volume data. See “SLICER3” on page 1027 of the IDL Reference Guide.
References

Chapter 13

Map Projections

The following topics are covered in this chapter:

- The MAP_SET Procedure .................... 358
- The MAP_GRID Procedure .................. 360
- The MAP_CONTINENTS Procedure ..... 360
- Graphics Techniques for Mapping ...... 361
- Map Projections Described ............... 362
- Azimuthal Projections ...................... 362
- Cylindrical Projections ................... 368
- Pseudocylindrical Projections .......... 373
- Putting Data on Maps ................. 375
- High-Resolution Continent Outlines ... 377
- References .......................................... 378
The IDL mapping package contains the following procedures:

**MAP_SET**
This procedure establishes the coordinate conversion mechanism for mapping points on a globe's surface to points on a plane, according to one of 16 possible projections. This procedure also sets up the clipping parameters of the region to be mapped, the center of the map, and the polar rotation. MAP_SET must be called to set up a map projection before any other mapping routines are called. See “MAP_SET” on page 723 of the IDL Reference Guide.

**MAP_GRID**
This procedure draws the graticule of parallels and meridians (grid lines) according to the specifications established by MAP_SET. See “MAP_GRID” on page 711 of the IDL Reference Guide.

**MAP_CONTINENTS**
This procedure draws continental or other boundaries over a map projection established by MAP_SET. Continents, coastlines, rivers, and political borders can be drawn in either low or high resolution. Continents may also be filled with solid colors. See “MAP_CONTINENTS” on page 708 of the IDL Reference Guide.

**MAP_IMAGE and MAP_PATCH**
These functions return an image warped to fit the current map projection. See the IDL Reference Guide for more information. See “MAP_IMAGE” on page 715 of the IDL Reference Guide and “MAP_PATCH” on page 718 of the IDL Reference Guide.

**Direct Graphics vs. Object Graphics**
The examples in this chapter are all written to take advantage of IDL Direct Graphics. Examples and techniques using IDL Object Graphics are contained in a separate volume: *Objects and Object Graphics*. See that volume for more information on IDL Object Graphics or see “Graphics” on page 267 of *Building IDL Applications* for a discussion of the differences between IDL Direct Graphics and IDL Object Graphics.

**The MAP_SET Procedure**
The MAP_SET procedure establishes the axis type and coordinate conversion mechanism for mapping points on the Earth's surface, expressed in latitude and longitude, to points on a plane, according to one of 16 possible map projections. Many other keywords are available to control various graphics options. For information on all the available keywords, see “MAP_SET” on page 723 of the IDL Reference Guide.
You can select the map projection, the map center, polar rotation, and geographical limits. The system variable !MAP1 retains the information needed to effect coordinate conversions to the plane and inversely from the projection plane to points on the earth in latitude and longitude. Do not change the values of the fields in !MAP1 directly. You can plot the graticule and continental boundaries with MAP_SET by setting the GRID and CONTINENT keywords.

The procedure has the calling sequence:

```idl
MAP_SET[, P0lat, P0lon, Rot]
```

where

- **P0lat**
  
  P0lat is the latitude of the point on the earth's surface at the center of the projection plane. Latitude is measured in degrees North of the equator, where -90° ≤ P0lat ≤ 90°. If P0lat is not set, the default value is zero.

- **P0lon**

  P0lon is the longitude of the point on the Earth's surface to be mapped to the center of the map projection. Longitude is measured in degrees east of the Greenwich meridian and -180° ≤ P0lon ≤ 180°. If P0lon is not set, the default value is zero.

- **Rot**

  Rot is the angle through which the North direction should be rotated around the line L between the Earth's center and the point (P0lat, P0lon). Rot is measured in degrees with the positive direction being clockwise rotated around L. Rot should satisfy -180° ≤ Rot ≤ 180°.

  If the center is at the North Pole, the North direction is in the direction of P0lon + 180 degrees. If the origin is at the South Pole, then North is in the direction of P0lon. The default value of Rot is zero.

**MAP_SET Keywords**

MAP_SET accepts many keywords that customize the projection attributes of the map. A few of the important ones are described below. See “MAP_SET” on page 723 of the IDL Reference Guide for descriptions of all the keywords.

- **CONTINENTS**

  Set this keyword to plot the continental boundaries.

- **GRID**

  Set this keyword to draw the grid of parallels and meridians.
ISOTROPIC
Set this keyword to produce a map that has the same scale in the X and Y directions.

LIMIT
Set this keyword to a four- or eight-element vector. The four-element vector, [Lat_min, Lon_min, Lat_max, Lon_max], specifies the boundaries of a simple region to be mapped. (Lat_min, Lon_min) and (Lat_max, Lon_max) are the latitudes and longitudes of two points diagonal from each other on the region's boundary. For more complex regions or projections, the eight-element vector, [Lat_0, Lon_0, Lat_1, Lon_1, Lat_2, Lon_2, Lat_3, Lon_3] specifies four points located, respectively, on the left, top, right and bottom edges of the map.

SCALE
Set this keyword to construct an isotropic map with the given scale, set to the ratio of 1:scale. If SCALE is not specified, the map is fit to the window. The typical scale for global maps is in the ratio of between 1:100 million and 1:200 million. For continents, the typical scale is in the ratio of approximately 1:50 million. For example, SCALE=100E6 sets the scale at the center of the map to 1:100 million, which is in the same ratio as 1 inch to 1578 miles (1 cm to 1000 km).

The MAP_GRID Procedure
MAP_GRID draws the graticule of parallels and meridians according to the specifications established by MAP_SET. The MAP_SET procedure should be called before MAP_GRID to establish the projection type, the center of the projection, polar rotation, and geographical limits. Latitude and/or longitude lines can be drawn in different linestyles, colors, and spacings. For information on all the available options, see "MAP_GRID" on page 711 of the IDL Reference Guide.

The MAP_CONTINENTS Procedure
MAP_CONTINENTS draws the projection of the continental boundaries, according to the specifications established by MAP_SET. MAP_SET should be called before MAP_CONTINENTS to establish the projection type, the center of the projection, polar rotation, and geographical limits. For information on all the available options, see "MAP_CONTINENTS" on page 708 of the IDL Reference Guide.
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 MAP_SET procedure automatically sets up the proper mapping technique to best fit the projection selected by the user.

Caution 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 Projections Described

In the following sections, the available projections are discussed in detail. The projections are grouped within three categories: azimuthal, cylindrical, and pseudo-cylindrical.

Note that 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 $P_{0lat}$ and $P_{0lon}$, respectively. $Rot$ 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, stereographic, gnomonic, azimuthal equidistant, Aitoff, Lambert's azimuthal equal area, Hammer-Aitoff, and satellite.

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 statements are used to produce an orthographic projection centered over Eastern Spain at a scale of 70 million to 1:

```
MAP_SET, /ORTHOGRAPHIC, 40, 0, SCALE=70e6, /CONTINENTS, $
/GRID, LONDEL=15, LATDEL=15, 
TITLE = 'Oblique Orthographic'
```

The output of these statements is shown Figure 13-1.

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
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 statement uses the stereographic projection to draw the hemisphere centered on the equator at longitude -105 degrees and produces an equatorial stereographic map:

```
MAP_SET, /STEREO, 0, -105, /ISOTROPIC, $
 /GRID, LATDEL = 20, LONDEL = 20, /HORIZON, /CONTINENT, $
 TITLE = 'Equatorial Stereographic'
```

The output of this statement is shown in the Figure 13-2(upper-left corner).

Since the LATDEL and LONDEL keywords are set to 20, parallels and meridians are spaced 20 degrees apart. The GRID and CONTINENT keywords signal that the grid and continents should be drawn.

**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
Azimuthal Projections

clipping region for the gnomonic projection is a circle with a radius of 60 degrees at the center of projection.

The oblique gnomonic projection shown in the Figure 13-2 (lower-left corner) is produced by the following statement:

```
MAP_SET, /GNOMIC, 40, -105, LIMIT = [20, -130, 70, -70], $
   /ISOTROPIC, /GRID, /CONTINENT, $
   TITLE = 'Oblique Gnomonic'
```

This projection 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.
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. 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.

If the keyword LIMIT is not set, the whole globe is mapped to the UV plane. The polar azimuthal projection of Figure 13-2 (lower-right corner) is created using the following statement:

```
MAP_SET, /AZIMUTHAL, /ISOTROPIC, -90, $ 
   /GRID, LONDEL=20, LATDEL=20, /CONTINENT, $ 
   /HORIZON, TITLE = 'Polar Azimuthal'
```

It is centered at the South Pole and shows the entire globe.

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 can be produced by the command:

```
MAP_SET, 0, 180, /Aitoff, /GRID, /CONTINENTS, /ISOTROPIC, $ 
   TITLE = '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 statement produces the polar Lambert projection in Figure 13-2 (upper-right corner):

```
MAP_SET, /AZIMUTHAL, /ISOTROPIC, -90, $ 
   /GRID, LONDEL=20, LATDEL=20, /CONTINENT, $ 
   /HORIZON, TITLE = 'Polar Lambert'
```
MAP_SET, /LAMBERT, 90, 0, -105, /ISOTROPIC, $
   /GRID, LATDEL=20, LONDEL=20, $
   /CONTINENTS, E_CONTINENTS={FILL:1}, /HORIZON, $
   TITLE = 'Polar Lambert'

Note that this map shows the Northern Hemisphere rotated counterclockwise 105 degrees, filling the continents with a solid color.

**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 can be produced by the command:

```
MAP_SET, 0, 180, /HAMMER, /GRID, /CONTINENTS, /ISOTROPIC, $
   /HORIZON, TITLE= '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 $\Gamma$ degrees clockwise from the north, then tilted $\omega$ degrees downward from horizontal.
For the satellite projection, \( P_{\text{OLat}} \) and \( P_{\text{O Lon}} \) represent the latitude and longitude of the sub-satellite point. Three additional parameters, \( P \), \( \Omega \), and \( \Gamma \) (supplied as a three-element vector argument to the SAT_P keyword), are required where:

- \( P \) is the distance of the point of perspective (camera) from the center of the globe, expressed in units of the radius of the globe.
- \( \Omega \) is the downward tilt of the camera, in degrees from the new horizontal. If both \( \Gamma \) and \( \Omega \) are 0, a Vertical Perspective projection results.
- \( \Gamma \) is the angle, expressed in degrees clockwise from north, of the rotation of the projection plane.

**Note** Since all meridians and parallels are oblique lines or arcs, the LIMIT keyword must be supplied as an eight-element vector representing four points that delineate the limits of the map. The extent of the map limits, when expressed in latitude/longitude, is a complicated polygon, rather than a simple quadrilateral.

The map in Figure 13-3, which shows the eastern seaboard of the United States from an altitude of about 160km, above Newburgh, NY, was produced with the code shown below.

The parameters for this satellite projection are:

- Center of projection = 41.5N latitude, -74W longitude
Cylindrical Projections

- \( P \) (altitude) = 1.025 = \((1.0 + 160 / 6371\text{km})\)
- Gamma (rotation of projection plane) = 150 degrees
- Omega (tilt of projection plane) = 55 degrees
- The eight element LIMIT keyword array specifies the latitude/longitude locations of points at the bottom, left, top, and right of the map respectively.
- The HORIZON keyword draws a horizon line.

Example  Labeling and drawing a vector on a satellite projection.

```idl
MAP_SET, /SATELLITE, SAT_P=[1.0251, 55, 150], 41.5, -74., 
/ISOTROPIC, /HORIZON, 
LIMIT=[39, -74, 33, -80, 40, -77, 41,-74], 
/CONTINENTS, TITLE='Satellite / Tilted Perspective'
MAP_GRID, /LABEL, LATLAB=-75, LONLAB=39, LATDEL=1, LONDEL=1
Set up the satellite projection.

p = convert_coord(-74.5, [40.2, 40.5], /TO_NORM)
Get North vector.

ARROW, p(0,0), p(1,0), p(0,1), p(1,1), /NORMAL
Draw North arrow.

XYOUTS, -74.5, 40.1, 'North', ALIGNMENT=0.5
```

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 \( P_{o\text{lat}}, P_{o\text{lon}}, \) and Rot determine the great circle that passes through the point \(C=(P_{o\text{lat}}, P_{o\text{lon}})\). 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. 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. By default, the keyword LIMIT is set to [-80, -180, 80, 180] because of the great distortions around the poles when the equator is selected.

The following statement produces a simple Mercator projection:

```idl
MAP_SET, /MERCATOR, 0, 0, /ISOTROPIC, $
        /GRID, /CONTINENTS, $
        TITLE = 'Simple Mercator'
```

The result of this statement is shown in Figure 13-4 (upper-left corner). Latitudes range from -80 degrees to 80 degrees.

**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, but its parameters can be altered with the ELLIPSOID keyword.

**Example**

To create a UTM map, centered near London, with a scale of 10 million to one, type the following:

```idl
MAP_SET, /TRANSVERSE, 51, 0, SCALE=10e6, $
```

---

**Using IDL**

**Cylindrical Projections**
When the eccentricity of the Earth is not important, global scale Transverse Mercator projections can be easily created using the Mercator projection with the CENTRAL_AZIMUTH keyword set to 90 degrees, and setting Rot to rotate the map 90 degrees. For example, to create the Transverse Mercator map showing North and South America, with a central meridian of -90 degrees West and centered on the Equator, shown in Figure 13-4 (upper-right corner). It is produced by the following statement:

```
MAP_SET, /MERCATOR, 0, -75, 90, CENTRAL_AZIMUTH=90, 
/ISOTROPIC, LIMIT= [32,-130, 70,-86, -5,-34, -58, -67], $
```

---

Figure 13-4: Cylindrical Projections

```
/GGRID, LATDEL=2.5, LONDEL=2.5, /LABEL, LONLAB=48, $
/CONTINENTS, E_CONT={COUNTRIES:1, COASTS:1}, $
TITLE='UTM Projection'
```

When the eccentricity of the Earth is not important, global scale Transverse Mercator projections can be easily created using the Mercator projection with the CENTRAL_AZIMUTH keyword set to 90 degrees, and setting Rot to rotate the map 90 degrees. For example, to create the Transverse Mercator map showing North and South America, with a central meridian of -90 degrees West and centered on the Equator, shown in Figure 13-4 (upper-right corner). It is produced by the following statement:

```
MAP_SET, /MERCATOR, 0, -75, 90, CENTRAL_AZIMUTH=90, 
/ISOTROPIC, LIMIT= [32,-130, 70,-86, -5,-34, -58, -67], $
```
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 code is used to produce a simple cylindrical equidistant projection and an oblique cylindrical equidistant projection as shown in the lower left and right sections of Figure 13-4:

```idl
MAP_SET, /CYLINDRICAL, 0, 0, /GRID, /CONTINENTS, TITLE = 'Simple Cylindrical Equidistant',
MAP_SET, /CYLINDRICAL, 0, 0, 45, /GRID, /CONTINENT, /HORIZON, TITLE='Oblique Cylindrical Equidistant'
```

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 are projected onto circles and meridians onto equally
spaced straight lines. The STANDARD_PARALLELS keyword specifies the latitudes of one or two standard parallels.

The following statement produces the map shown in Figure 13-5, which features North America with standard parallels at 20 degrees and 60 degrees:

```idl
MAP_SET, /CONIC, 40, -80, STANDARD_PARALLELS=[20,60], 
   /ISOTROPIC, LIMIT=[0, -260, 80, 100], 
   /GRID, LATDEL=15, LONDEL=20, /CONTINENT, 
   TITLE= 'Lambert’s Conic'
```

**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.

Note  The central meridian should always be 0 degrees longitude to retain the correct balance of shapes, sizes, and relative positions.

The following statement produced the example of the Robinson projection in Figure 13-6 (bottom left)

```
MAP_SET, /ROBINSON, 0, 0, /ISOTROPIC, /GRID, $
/HORIZON, E_CONTINENTS={FILL:1}, TITLE='Robinson'
```

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 $\phi$ and longitude $\lambda$ of a point on the globe and its image on the UV plane.

\[
\begin{align*}
    u &= \lambda \cos \phi \\
    v &= \phi
\end{align*}
\]

The parameters \(P_{\text{OLat}}\) and \(\text{Rot}\) of the MAP_SET procedure must be zero. If they are not, an error message results and the procedure MAP_SET will reset both of these parameters to zero and continue. By default, \(P_{\text{OLon}}\) (the central longitude) is zero, but the user can set it to any other value between -180 and 180. If the keyword LIMIT is undefined, the entire globe is the region selected for mapping.
The following statements produce the sinusoidal map of the whole globe centered at longitude 0 degrees and latitude 0 degrees:

```idl
MAP_SET, /SINUSOIDAL, /ISOTROPIC, 
/CONTINENTS, TITLE='Sinusoidal'
MAP_GRID, LONDEL=20, /HORIZON
```

The result of these statements is shown in Figure 13-6 (top left).

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.

If the keyword LIMIT is not set, the whole globe will be mapped to the plane. The following statement produces a Mollweide projection in oblique form, as illustrated in Figure 13-6 (upper right):
MAP_SET, /MOLLWEIDE, 45, 0, /ISOTROPIC, $ 
/GRID, LATDEL=20, LONDEL=20, $ 
/HORIZON, E_CONTINENTS=(FILL:1), $ 
TITLE='Oblique Mollweide'

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 statement produced the example of Goode's Homolosine projection in Figure 13-6 (bottom right):

```
MAP_SET, /GOODESHOMOLOSINE, 0, 0, /ISOTROPIC, /GRID, $ 
LATDEL=15, LONDEL=20, /HORIZON, E_CONTINENTS=(FILL:1), $ 
TITLE='Goode Homolosine'
```

**Putting Data on Maps**

The procedures PLOT, O PLOT, PLOTS, XYOUTS, and CONTOUR can be used to display and annotate geographical data on maps created by the routines MAP_SET, MAP_GRID, and MAP_CONTINENTS. The MAP_IMAGE procedure can be used to warp regularly-gridded images to map projections.

**Example—Using CONTOUR with MAP_SET**

The following simple example creates a CONTOUR plot over a Mollweide map projection and then over a polar stereographic projection. The resulting map is shown in Figure 13-7.

```
lat = REPLICATE(10., 37) # FINDGEN(19) - 90.
lon = FINDGEN(37) # REPLICATE(10, 19) 
```

Make a 10 degree latitude/longitude grid covering the Earth.
X = COS(!DTOR * lon) * COS(!DTOR * lat)
Y = SIN(!DTOR * lon) * COS(!DTOR * lat)
Z = SIN(!DTOR * lat)

Convert lat and lon to Cartesian coordinates

F = (X-1.)^2 + (Y-1.)^2 + (Z-1.)^2

Create the function to be plotted, set it equal to the distance squared from (1,1,1).

MAP_SET, /MOLLWEIDE, 0, 0, /ISOTROPIC, $
/HORIZON, /GRID, /CONTINENTS, $
TITLE='Mollweide Contour'
CONTOUR, F, lon, lat, NLEVELS=7, $
/OVERPLOT, /DOWNHILL, /FOLLOW
MAP_SET, /STEREO, 90, 0, $
/ISOTROPIC, /HORIZON, E_HORIZON={FILL:1}, $

Figure 13-7: Combining CONTOUR with MAP_SET.
TITLE='Stereographic Contour'

CONTOUR, F(*,10:*), lon(*,10:*), lat(*,10:*), 
/OVERPLOT, /FILL, NLEVELS=5

Display points in the northern hemisphere only.

MAP_GRID, /LABEL, COLOR=255
MAP_CONTINENTS, COLOR=255

Limitations

Filling contours or polygons over maps that cover more than a hemisphere will produce incorrect results. This is because of the ambiguity between polygons that enclose an area, and those that enclose the entire surface of the sphere outside the area; and because of the ambiguity of determining the “clockwise-ness” of polygons on a sphere that cover more than a hemisphere.

High-Resolution Continent Outlines

IDL supports two different datasets that contain continent outlines and other geographical and political boundaries. The default dataset 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 dataset that may not have been installed when your copy of IDL was first installed.

To access the high-resolution dataset, 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. For an example of using the high-resolution outlines, see “MAP_CONTINENTS” in the IDL Reference Guide.

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 maps subdirectory of the resource subdirectory of the main IDL directory). 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:

<table>
<thead>
<tr>
<th>Feature</th>
<th>Low-Resolution</th>
<th>High-Resolution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coastlines, islands, and lakes</td>
<td>Data in file supmap.dat.</td>
<td>Entire CIA World Map.</td>
</tr>
<tr>
<td>(including continental outlines)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Continental polygons</td>
<td>Data extracted from supmap.dat.</td>
<td>Every 20th point of CIA World Map.</td>
</tr>
<tr>
<td>Rivers</td>
<td>Every 250th point of the CIA World Map.</td>
<td>Entire CIA World Map.</td>
</tr>
<tr>
<td>National boundaries</td>
<td>Every 100th point of CIA World Map.</td>
<td>Entire CIA World Map.</td>
</tr>
</tbody>
</table>

Table 13-1: Comparison between low-resolution and high-resolution map databases

**References**


Chapter 14

Image Display Routines

The following topics are found in this chapter:

Images ............................................... 380
Imaging Routines................................ 380
Image Display..................................... 381
Reading from the Display Device ........ 385
Color Tables........................................ 386
True-Color Displays............................. 392
Controlling the Device Cursor ............ 395
References ........................................ 396
IDL provides a powerful environment for image processing and display. The routines described in this chapter provide the interface between IDL and the image display system. This chapter describes these image display and control routines and provides examples of their use.

Direct Graphics vs. Object Graphics
The examples in this chapter are all written to take advantage of IDL Direct Graphics. Examples and techniques using IDL Object Graphics are contained in a separate volume: Objects and Object Graphics. See that volume for more information on IDL Object Graphics or see “Graphics” on page 267 of Building IDL Applications for a discussion of the differences between IDL Direct Graphics and IDL Object Graphics.

Images
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.

Imaging Routines
The following IDL routines are used for the display and manipulation of images:

TVCRS
This procedure manipulates the image device cursor. TVCRS allows the cursor to be enabled and disabled, as well as allowing it to be positioned.

TV
This procedure displays images on the image display.

TVSCL
This procedure scales the intensity values of the image into the range of the display device, then displays the result on the image display.

TVLCT
This procedure loads a new color table into the display device.

TVRD
This function reads image pixels back from the display device.
In addition, most routines used for plotting and graphics can be used with the display of images as well. These routines are described in Chapter 11, “Plotting” and Chapter 12, “Plotting Multi-Dimensional Arrays”. For example, to overlay an image and its contour plot, the output of the CONTOUR procedure is combined with that of TV. The CURSOR routine, described in “Using the CURSOR Procedure” on page 313, reads the position of the interactive pointing device and may also be used to determine the location of image pixels.

### Image Display

The TV and TVSCL procedures display images on the screen. These procedures use the same arguments and keywords and differ only in that TVSCL scales the image into the intensity range of the display device, while TV displays the image directly. They have the form:

- `TV, IMAGE[, POSITION]`
- `TV, IMAGE[, X, Y[, CHANNEL]]`
- `TVSCL, IMAGE[, POSITION]`
- `TVSCL, IMAGE[, X, Y[, CHANNEL]]`

where

**IMAGE**

A vector or two-dimensional matrix to be displayed as an image. If not already of byte type, it is converted prior to use.

**X, Y**

If present, these arguments specify the lower-left coordinate of the displayed image.

**POSITION**

Position number of the image. Image positions are discussed in detail below.

**CHANNEL**

Some image display devices are capable of storing more than a single image or can combine three single color images to form a true color image. CHANNEL specifies the memory channel to be written. It is assumed to be zero if not specified. This parameter is ignored on display systems that have only one memory channel.

If no optional parameters are present, IMAGE is output to the display with its lower-left corner at coordinate (0, 0). The optional parameters can be used to specify the screen position of the image in a variety of ways.
**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 \((x_{\text{size}}-1, y_{\text{size}}-1)\), where \(x_{\text{size}}\) and \(y_{\text{size}}\) are the dimensions of the visible area of the display. The descriptions of the image display routines that follow assume a display size of \(512 \times 512\), 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.

**Image Position**

Image positions run from the left of the screen to the right and from the top of the screen to the bottom. If a position number is used instead of \(x\) and \(y\), the position of the image is calculated from the dimensions of the image (using integer arithmetic) as follows:

\[
\begin{align*}
\text{x}_{\text{size}} & \times \text{y}_{\text{size}} = \text{size of display or window} \\
\text{x}_{\dim} & \times \text{y}_{\dim} = \text{dimensions of array} \\
\text{N}_x & = \frac{\text{x}_{\text{size}}}{\text{x}_{\dim}} = \text{images across screen} \\
\text{x} & = \text{x}_{\dim}\text{Position}_{\text{moduloN}_x} = \text{startingx} \\
\text{y} & = \text{y}_{\text{size}} \times \text{y}_{\dim} \times \left(1 + \frac{\text{Position}}{\text{N}_x}\right) = \text{startingy}
\end{align*}
\]

For example, when displaying \(128 \times 128\) images on a \(512 \times 512\) display, the position numbers run from 0 to 15 as follows:

```
0 1 2 3
4 5 6 7
8 9 10 11
12 13 14 15
```
Image Size

Most image devices have a fixed number of display pixels. Common sizes are 512 × 512, 1280 × 1024, and 900 × 1152 (for Sun workstations). Such pixels have a fixed size which cannot be changed. For such devices, the area written on the screen is the same size as the dimensions of the image array. One-dimensional vectors are considered row vectors. The x and y parameters specify the coordinates of the lower-left corner of the area written on the display.

There are some devices, however, that have the ability to place an image with any number of pixels into an area of arbitrary size. PostScript devices are a notable example. Such devices are said to have scalable pixels, because there is no direct connection between the number of pixels in the image and the physical space it occupies in the displayed image. When the current image device has scalable pixels, IDL sets the first bit of !D.FLAGS. The following IDL statement can be used to determine if the current device has scalable pixels:

```
SP = !D.FLAGS AND 1
```

SP will be nonzero if the device has scalable pixels. When displaying an image on a device with scalable pixels, the default uses the entire display surface for the image. The XSIZE and YSIZE keywords can be used to override this default and specify the width and height that should be used.

The XSIZE and YSIZE keywords also should be used when positioning images with the POSITION argument to TV or TVSCL. POSITION normally uses the size of the image in pixels to determine the placement of the image, but this is not possible for devices with scalable pixels. Instead, the default for such devices is to assume a single position that fills the entire available display surface. However, if XSIZE and YSIZE are specified, POSITION will use them to determine image placement.

Examples

```
TV, REPLICATE(100B, 512, 512)  ; Set all display memory to 100.
ABC = BYTARR(50, 100)          ; Define a 50 column by 100 row array.
TV, ABC, 300, 400               ; Display array ABC starting at location x = 300, y=400. Display pixels in columns 300 to 349, and rows 400 to 499 are zeroed.
TV, ABC/2, 12                   ; Display image divided by 2 at position number 12.
TV, A, 256, 256, 2              ; Output image to memory channel 2, lower-left corner at (256, 256).
```
AA = ASSOC(1, BYTARR(64, 64))  
Assume file one contains a sequence of 64 x 64 byte arrays.

FOR I = 0, 63 DO TV, AA[I], I  
Display 64 images from file, from left to right and top to bottom, filling a 512 x 512 area.

**Image Scaling**

An image can be contrast enhanced so any subrange of pixel values are scaled to fill the entire range of displayed brightnesses using a variety of methods.

For example, if the image \( A \) contains an object superimposed on a varying background and the pixel values in the object range from a value of \( S \) to the brightest value in the image the IDL statement:

\[
\text{TVSCL, } A > S
\]

will use the entire range of display brightnesses to display the object. The expression \( A > S \) results in an image in which each pixel in \( A \) less than \( S \) is set to \( S \). \( S \) becomes the new minimum intensity. The TVSCL procedure scales the new image into the available number of color-table entries before loading it into the display. Again, the image \( A \) is not changed.

Another method that is more efficient, although slightly obscure, is to use the BYTSCL function to scale the array as follows:

\[
\text{TV, BYTSCL}(A, \text{MIN} = S, \text{TOP} = \!D.\text{TABLE\_SIZE})
\]

This method is more efficient because the value \( S \) is known and avoids scanning the array for the minimum and maximum values. Also, one less array operation is required.

If the object in \( A \) has values from 2.6 to 9.4, the statements

\[
\text{TVSCL, } A > 2.6 < 9.4 \quad \text{Slow method.}
\]

\[
\text{TV, BYTSCL}(A, \text{MIN} = 2.6, \text{MAX} = 9.4, \text{TOP} = \!D.\text{TABLE\_SIZE}) \quad \text{Faster method.}
\]

will truncate the image so 2.6 is the new minimum and 9.4 is the new maximum before scaling and display.

Some examples of using the TVSCL function follow.

\[
\text{TVSCL, SQRT}(A) \quad \text{Display square root of image.}
\]

\[
\text{TVSCL, } A - \text{SMOOTH}(A, 3) \quad \text{Display unsharp masked image.}
\]

\[
\text{TVSCL, } A + B, 12 \quad \text{Display scaled sum at position number 12.}
\]
Reading from the Display Device

The TVRD function reads the contents of the display device memory back into an IDL variable. One use for this capability is to build up a complex display using many IDL statements, and then read the resulting image back as a single unit for storage in a file.

The TVRD function returns the contents of the specified rectangular portion of the display subsystem's memory. The coordinate \((x_0, y_0)\) is the starting coordinate of the data to be read, and \(N_x, N_y\) is the size of the rectangle in columns and rows. This results in a byte array of dimensions \(N_x \times N_y\).

A Note on Reading Data from Windows

On some systems, when backing store is provided by the window system (RETAIN =1), reading data from a window using TVRD() may cause unexpected results. For example, data may be improperly read from the window even when the image displayed on screen is correct. Having IDL provide the backing store (RETAIN =2) ensures that the window contents will be read properly.

The TVRD function has the form:

\[
\text{TVRD}(X_0, Y_0, N_x, N_y[, \text{CHANNEL}])
\]

where

\(X_0\)
Starting column of data to read.

\(Y_0\)
Starting row of data to read.

\(N_x\)
Number of columns to read.

\(N_y\)
Number of rows to read.

\textbf{CHANNEL}
Memory channel to be read. It is assumed to be zero if not specified. This parameter is ignored on display systems that only have one memory channel.

If the system variable \(!\text{ORDER}\) is set to zero, then data are read from the bottom up; otherwise, data are read in the top-down direction.

\textbf{Example} The following statement inverts the 100 × 100 area of the display starting at (200, 300):

\[
\text{TVRD}(200, 300, 100, 100, 0)
\]
TV, NOT TVRD(200, 300, 100, 100) Reverse area.

**Ability to Read from Display**
Not all image devices are able to support reading pixels back from device memory. If the current device has this ability, IDL sets the eighth bit of !D.FLAGS.

\[
\text{TEST} = !D.\text{FLAGS} \text{ AND } 128
\]

Determine if the current device allows reading from display memory

TEST will be nonzero if the device allows such operations.

**Color Tables**
There are numerous systems for the measuring and specification of color. Most systems are three-dimensional in nature. For a complete discussion of color systems, refer to Foley and Van Dam (1982, Chapter 17). Parts of this discussion are taken from that chapter.

Most devices capable of displaying color use the RGB (red, green, and blue) color system. Other common color systems include the Munsell, HSV (hue, saturation, and value), HLS (hue, lightness, and saturation), and CMY (cyan, magenta, and yellow) color systems. Algorithms exist to convert colors from one system to another. IDL accepts color specifications in the RGB, HLS, or HSV color systems.

The RGB color system, as implemented in IDL, uses a three-dimensional Cartesian coordinate system with the value of each color ranging from 0 to 255. Each displayable color is a point within this cube, shown in Figure 14-1 (after Foley and Van Dam). The origin, (0, 0, 0), where each color coordinate is 0, is black. The point at (255, 255, 255) is white and represents an additive mixture of the full intensity of each of the three colors. Points along the main diagonal—where the 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.

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:

\[
C_i = \{r_i, g_i, b_i\}, \quad 0 \leq i < 2^m \\
0 \leq r_i, g_i, b_i < 2^n
\]

Objects containing a value, or color index, of \( i \) are displayed with a color of \( C_i \).

The IDL COLOR_CONVERT procedure can be used to convert color triples to and from the RGB color system and the HLS and HSV systems.

You can display true color images on pseudo-color displays by using the COLOR_QUAN function. This function creates the “best” pseudo-color palette for displaying the true-color image and then maps the true color image to the new palette. See “COLOR_QUAN” on page 279 of the IDL Reference Guide for more information.

---

Blue \((0,0,255)\)  
Cyan \((0,255,255)\)  
White  
Magenta \((255,0,255)\)  
Green \((0,255,0)\)  
Yellow \((255,255,0)\)  
Red \((255,0,0)\)

**Figure 14-1: RGB Color Cube.** (Note that grays are on the main diagonal.)
Loading Color Tables

IDL maintains its own internal color table which is read and written by the TVLCT procedure. When this table is modified, it is loaded into the currently selected graphics output device. A call to this procedure has the form:

TVLCT, RED, GREEN, BLUE[, START]

where

**RED, GREEN, and BLUE**

Vectors containing the intensity or value of each color for each index. Standard devices have an 8-bit color representation so the color values should range from 0 to 255. These vectors can contain up to $2^m$ elements (usually 256), assuming the display contains m bit pixels.

**START**

The starting index in the color translation table into which RED, GREEN, and BLUE will be loaded. If not specified, a value of 0 is used, causing the tables to be loaded starting at the first element of the translation vectors. START can be used to change only part of the color table.

In addition, the following keyword parameters can also be present:

**GET**

Returns the RGB values from the internal color table into the three variables.

**HLS**

Indicates that the parameters specify color using the HLS color system. The plain argument parameters are in the order H-L-S. Hue is expressed in degrees, and the lightness and saturation range from 0 to 1.

**HSV**

Indicates that the parameters specify color using the color system. The plain argument parameters are in the order H-S-V. As above, hue is in degrees, and the saturation and value range from 0 to 1.

**Example**

This example creates a graph with the axes drawn in white, then successively adds red, green, blue, and yellow lines. As there are five distinct colors, plus one color for the background, a six-element color table is created. Usually, color index 0 represents black (0, 0, 0). We arbitrarily choose color index 1 to be white (1, 1, 1), 2 as red (1, 0, 0), 3 as green (0, 1, 0), 4 as blue (0, 0, 1), and 5 as yellow (1, 1, 0). The display must have at least 3 bits per pixel to represent six colors simultaneously, and an 8-bit color table is assumed.
Using IDL

Chapter 14: Image Display Routines

RED = [0, 1, 1, 0, 0, 1]  
Specify the red component of each color.

GREEN = [0, 1, 0, 1, 0, 1]  
Specify the green component of each color.

BLUE = [0, 1, 0, 0, 1, 0]  
Specify the blue component of each color.

TVLCT, 255 * RED, 255 * GREEN, 255 * BLUE  
Load the first six elements of the color table.

PLOT, COLOR = 1, /NODATA,...  
Draw the axes in white, color index 1.

OPLLOT, COLOR = 2, ...  
Draw in red.

OPLLOT, COLOR = 3, ...  
Draw in green.

OPLLOT, COLOR = 4, ...  
Draw in blue.

OPLLOT, COLOR = 5, ...  
Draw in yellow.

The INDGEN function is handy when creating larger color tables in which each color's intensity can be expressed as a function of its index:

A = INDGEN(256)  
Straight line, A[I] = I.

TVLCT, A, A * 0, A * 0  
Display image with a linear red scale, disable green and blue.

TVLCT, A, A, A  
Display with linear black and white scale.

TVLCT, A, 2 * (A - 128) > 64, 4 * (A - 192) > 0  
Warm body temperature scale. Red is linear, green starts at 128, and blue starts at 192.

Color Table Procedures

The following IDL procedures are used to manipulate color tables:

LOADCT

Load predefined color tables. LOADCT has one parameter: the index of the predefined color table to be loaded. There are 40 pre-defined color tables in the file colors1.tbl, which is supplied with IDL. To obtain a menu listing the available color tables, call LOADCT with no parameters. Standard tables are listed below.
XLOADCT

This procedure provides a widget interface to LOADCT. Pre-defined color tables can be loaded and manipulated using this tool. Tables can be stretched and Gamma corrected interactively using this procedure.

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>Number</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Black &amp; White Linear</td>
<td>21</td>
<td>Hue Sat Value 1</td>
</tr>
<tr>
<td>1</td>
<td>Blue/White Linear</td>
<td>22</td>
<td>Hue Sat Value 2</td>
</tr>
<tr>
<td>2</td>
<td>Green-Red-Blue-White</td>
<td>23</td>
<td>Purple-Red + Stripes</td>
</tr>
<tr>
<td>3</td>
<td>Red Temperature</td>
<td>24</td>
<td>Beach</td>
</tr>
<tr>
<td>4</td>
<td>Blue-Green-Red-Yellow</td>
<td>25</td>
<td>Mac Style</td>
</tr>
<tr>
<td>5</td>
<td>Standard Gamma-II</td>
<td>26</td>
<td>Eos A</td>
</tr>
<tr>
<td>6</td>
<td>Prism</td>
<td>27</td>
<td>Eos B</td>
</tr>
<tr>
<td>7</td>
<td>Red-Purple</td>
<td>28</td>
<td>Hardcandy</td>
</tr>
<tr>
<td>8</td>
<td>Green/White Linear</td>
<td>29</td>
<td>Nature</td>
</tr>
<tr>
<td>9</td>
<td>Green/White Exponential</td>
<td>30</td>
<td>Ocean</td>
</tr>
<tr>
<td>10</td>
<td>Green-Pink</td>
<td>31</td>
<td>Peppermint</td>
</tr>
<tr>
<td>11</td>
<td>Blue-Red</td>
<td>32</td>
<td>Plasma</td>
</tr>
<tr>
<td>12</td>
<td>16 Level</td>
<td>33</td>
<td>Blue-Red 2</td>
</tr>
<tr>
<td>13</td>
<td>Rainbow</td>
<td>34</td>
<td>Rainbow 2</td>
</tr>
<tr>
<td>14</td>
<td>Steps</td>
<td>35</td>
<td>Blue Waves</td>
</tr>
<tr>
<td>15</td>
<td>Stern Special</td>
<td>36</td>
<td>Volcano</td>
</tr>
<tr>
<td>16</td>
<td>Haze</td>
<td>37</td>
<td>Waves</td>
</tr>
<tr>
<td>17</td>
<td>Blue-Pastel-Red</td>
<td>38</td>
<td>Rainbow18</td>
</tr>
<tr>
<td>18</td>
<td>Pastels</td>
<td>39</td>
<td>Rainbow + white</td>
</tr>
<tr>
<td>19</td>
<td>Hue Sat Lightness 1</td>
<td>40</td>
<td>Rainbow + black</td>
</tr>
<tr>
<td>20</td>
<td>Hue Sat Lightness 2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 14-1: Predefined Color Tables
XPALETTE
This widget procedure allows you to create your own color tables using a set of three sliders. This procedure can interpolate the space between color indices (to create “smooth” color transitions) or edit individual colors.

MODIFYCT
Saves color tables for later use by LOADCT.

HSV
Makes and loads color tables based on the HSV color system. A spiral through the single-ended HSV cone is traced. The color representation of pixel values is linearly interpolated from beginning and ending values of hue, saturation, and value.

HLS
Makes and loads color tables based on the HLS color system which is based on the Otswald color system. As with the HSV procedure, spirals are interpolated in the three-dimensional color space.

PSEUDO
Generates and loads a pseudo-color table based on the LHB (lightness, hue, and brightness) system.

STRETCH
Linearly expands the entire range of the last color table loaded to cover a given range of pixel values. STRETCH has two parameters: the pixel value to be displayed with color index 0 and the pixel value to be displayed with the maximum color index:

STRETCH, LOW, HIGH

Example
STRETCH, 100, 150

Expands the color tables so that pixels in the range of 100 to 150 fill the entire color range.

To revert to a normal color table, call STRETCH with no parameters.

Note  The window-oriented procedures will not work without a window system.

Obtaining the Color Tables
All of the IDL color-table procedures maintain the current color table in a common block called COLORS, defined as follows:
COMMON COLORS, R.ORIG, G.ORIG, B.ORIG, R.CURR, G.CURR, B.CURR

The variables are integer vectors of length equal to the number of color indices. Your program can access these variables by defining the common block. The convention is that routines that modify the current color table should read it from R.ORIG, G.ORIG, and B.ORIG, then load the color table using TVLCT and leave the resulting color table in R.CURR, G.CURR, and B.CURR.

Color Tables—Switching Between Devices
Use the SET_PLOT procedure to direct the graphics output to different devices. Because devices have differing capabilities and not all are capable of representing the same number of colors, the treatment of color tables when switching devices is somewhat tricky.

After selecting a new graphics output device, SET_PLOT will perform one of the following color-table actions depending upon which keyword parameters are specified:

- The default is to do nothing. The problem with this treatment is that the internal color tables incorrectly reflect the state of the device's color tables until TVLCT is called (usually via LOADCT).
- If the COPY keyword parameter is set, the internal color tables are copied into the device. This is straightforward if both devices have the same number of color indices. If the new device has more colors than the old device, some color indices will be invalid. If the new device has less colors than the old, not all the colors are saved. This is the preferred method if you are displaying graphics and each color index is explicitly loaded.
- When the INTERPOLATE keyword is set, the new device's table is loaded by interpolating the old color table to span the new number of color indices. This method works best when displaying images with continuous color ranges.

True-Color Displays
IDL supports the use of some true-color displays with 24 bits per pixel. True-color displays have multiple channels. That is, they store information about each primary color component (red, green, and blue) of a pixel separately. A true-color display with n bits per memory channel can display $2^n$ simultaneous colors, as opposed to the $2^m$ simultaneous colors available with a normal indexed (pseudo) color display. Images can be transferred to and from each individual memory channel, or to all channels simultaneously.

The X Window visuals TrueColor and DirectColor are among the true-color devices supported by IDL.
Configuration

The true-color display is configured as a single display with three channels:

<table>
<thead>
<tr>
<th>Channel Number</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>All colors</td>
</tr>
<tr>
<td>1</td>
<td>Red</td>
</tr>
<tr>
<td>2</td>
<td>Green</td>
</tr>
<tr>
<td>3</td>
<td>Blue</td>
</tr>
</tbody>
</table>

Table 14-2: True-Color Display Channels

Lookup Tables

Caution  Not all true-color display systems have writable color lookup tables.

Each output channel, red, green, or blue, is routed through its own 8-bit deep, 256 element lookup table. The lookup tables can be used to compensate for color inaccuracies generated by the display hardware or present in the acquisition process. Initially, each lookup table is loaded with a linear ramp, mapping its input directly to its output.

As the TrueColor lookup tables are of the same size and number of elements as those on a pseudo-color display, operation of the TVLCT procedure, which loads the lookup tables, is unchanged.

Furthermore, if the same image is loaded into each channel, operation of the display mimics that of a standard 8-bit deep pseudo-color display. Most, but not all, IDL image processing procedures written for a standard color display will run on a true-color display without modification. The routines that transfer images to the display, TV and TVSCL, write the same 8-bit data to each channel (channel 0) if no channel parameter is present. The function TVRD, which reads data from the display, returns the maximum value contained in the three-color channels for each pixel if no channel parameter is present.

Color Indices

The color index specifier can range from 0 to $2^{24} - 1$. The system variable field !D.N_COLORS, which contains the number of colors, is set to $2^{24}$ on a true-color display. The system variable field, !D.TABLE_SIZE, contains the number of RGB color table elements.

The low 8 bits, bits 0 to 7, of the color index are written to the red channel; bits 8 to 15 are written to the green; and bits 16 to 23 are written to the blue. For
example, a given RGB, the index is \( R + 256(G + 256B) \). To create a plot with a given color (assuming linear lookup tables), use the following statement:

\[ \text{PLOT}, X, Y, \text{COLOR} = R + 256L \cdot (G + 256L \cdot B) \]

**True-Color Images**

Images can be transferred to and from the display in either 8-bit or 24-bit mode. The \text{CHANNEL} parameter specifies the source or destination channel number for 8-bit images, and the \text{TRUE} keyword indicates for 24-bit images the method of channel interleaving. If neither keyword parameter is present, the 8-bit image is written to all three-color channels, yielding the same effect as if the channel parameter is specified as 0.

For example, to transfer three 8-bit images contained in the arrays R, G, and B to their respective channels, use the following statements:

\[
\begin{align*}
\text{TV}, R, 0, 0, 1 & \quad \text{Load red in channel 1.} \\
\text{TV}, G, 0, 0, 2 & \quad \text{Load green in channel 2.} \\
\text{TV}, B, 0, 0, 3 & \quad \text{Load blue in channel 3.}
\end{align*}
\]

The position parameters \((0, 0)\) above can be altered to write to any location in the window.

For 24-bit images, the RGB data can be interleaved by pixel, by line, or by image. Use the \text{TRUE} parameter to specify the method of interleaving. A \( c \) column by \( l \) line true-color image is dimensioned as follows:

<table>
<thead>
<tr>
<th>TRUE Value</th>
<th>Dimensions</th>
<th>Interleaving</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>((3, c, l))</td>
<td>Pixel</td>
</tr>
<tr>
<td>2</td>
<td>((c, 3, l))</td>
<td>Line</td>
</tr>
<tr>
<td>3</td>
<td>((c, l, 3))</td>
<td>Image</td>
</tr>
</tbody>
</table>

Table 14-3: Values for the \text{TRUE} Keyword

For example, to write a true-color, line interleaved image contained in the variable \( t \), with its lower-left corner at coordinate \((100, 200)\), use the following statement:

\[ \text{TV}, t, 100, 200, \text{TRUE} = 2 \]
Reading Images

To read from the display to an IDL variable or expression, use the TVRD function with either the CHANNEL parameter or TRUE keyword parameter. The calling sequence for TVRD is:

\[
\text{Result} = \text{TVRD}([X_0, Y_0, N_x, N_y, \text{Channel}])
\]

where \((X_0, Y_0)\) specifies the window coordinate of the lower-left corner of the rectangle to be read, and \((N_x, N_y)\) contains the number of columns and rows to read. Note that all parameters to TVRD are optional. If no arguments are supplied, the entire area of the display device is returned.

When used without the TRUE parameter, TVRD returns an \((N_x, N_y)\) byte image read from the indicated channel. If the channel number is not specified or is zero, the maximum RGB value of each pixel is returned, approximating the luminance.

If present and nonzero, the TRUE keyword indicates that a true-color image is to be read and specifies the index of the dimension over which color is interleaved. The result is a \((3, N_x, N_y)\) pixel interleaved array if TRUE is 1; or an \((N_x, 3, N_y)\) line interleaved array if TRUE is 2; or an \((N_x, N_y, 3)\) image interleaved array if TRUE is 3.

Some examples of TVRD follow.

\[
\begin{align*}
R &= \text{TVRD}(0, 0, 512, 512, 1) & \text{Read a } 512 \times 512 \text{ image, starting at } (0, 0), \text{ from the red channel into } R. \\
T &= \text{TVRD}(0, 0, 512, 512, \text{TRUE} = 2) & \text{Read a true-color } 512 \times 512, \text{ line interleaved image, starting at } (0, 0) \text{ into } T. \text{ The variable } T \text{ is now dimensioned } (512, 3, 512). \\
L &= \text{TVRD}(0, 0, 512, 512) & \text{Read the maximum RGB value of each pixel into } L.
\end{align*}
\]

Controlling the Device Cursor

The TVCRS function manipulates the cursor of the image display. Normally, the cursor is disabled and is not visible. TVCRS with one argument allows the cursor to be enabled or disabled. While TVCRS with two parameters enables the cursor and places it on pixel location \((x, y)\). TVCRS has the form

\[
\text{TVCRS}[\text{, ON_OFF}] \\
\text{TVCRS}[\text{, X, Y}]
\]

where
ON_OFF
Specifies whether the cursor should be on or off. If present and nonzero, the
cursor is enabled. If ON_OFF is zero or no parameters are specified, the cursor is
turned off.

X
The column to which the cursor will be set.

Y
The row to which the cursor will be set.

TVCRS also takes various keywords that affect how it positions the cursor.
Notably, the keywords DATA, DEVICE, and NORMAL specify the coordinate
system. For details, see the entry for TVCRS in the IDL Reference Guide.

References
Foley, J.D., and A. Van Dam (1982), Fundamentals of Interactive Computer
Chapter 15

Signal Processing

The following topics are covered in this chapter:

- Digital Signals .................................... 398
- Signal Analysis Transforms .................. 400
- The Fourier Transform ........................ 400
- Interpreting FFT Results ...................... 401
- Displaying FFT Results ....................... 402
- Using Windows .................................. 407
- Aliasing .......................................... 409
- FFT Usage Details ............................... 410
- The Hilbert Transform .......................... 411
- The Wavelet Transform ....................... 413
- Convolution ..................................... 413
- Correlation and Covariance ................. 413
- Digital Filtering ................................. 414
- Finite Impulse Response (FIR) Filters .... 414
- FIR Filter Implementation .................... 417
- Infinite Impulse Response Filters .......... 419
- Routines for Signal Processing ............. 423
- References ...................................... 423
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.

**Direct Graphics vs. Object Graphics**

The examples in this chapter are all written to take advantage of IDL Direct Graphics. Examples and techniques using IDL Object Graphics are contained in a separate volume: Objects and Object Graphics. See that volume for more information on IDL Object Graphics or see “Graphics” on page 267 of Building IDL Applications for a discussion of the differences between IDL Direct Graphics and IDL Object Graphics.

**Running the Example Code**

The example code used in this chapter is part of the IDL distribution. All of the files mentioned are located in the `doc` subdirectory of the `examples` subdirectory of the main IDL directory. 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” on page 43 of the IDL Reference Guide 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 $\text{delt}$. 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:

\[
\begin{align*}
N &= 1024 \\
delt &= 0.02 \\
u &= -0.3 + 1.0 \sin(2 \pi 2.8 \cdot \text{delt} \cdot \text{FINDGEN}(N)) + 1.0 \sin(2 \pi 6.25 \cdot \text{delt} \cdot \text{FINDGEN}(N)) + 1.0 \sin(2 \pi 11.0 \cdot \text{delt} \cdot \text{FINDGEN}(N))
\end{align*}
\]

Alternately, you can run the following batch file to create the signal:

```
@sigprc01
```

See “Running the Example Code” on page 398 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 Figure 15-1.

**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.
Enter the following commands at the IDL prompt to create the plot:

```idl
@sigprc01
Compute time data sequence u.
t = delt * FINDGEN(N)
Vector of discrete times.
t1 = 1.0
Beginning of plot time range.
t2 = 2.0
End of plot time range.
PLOT, t+delt/2, u, PSYM=10, XRANGE=[t1,t2], $
  \quad$\small
  \text{XTITLE='time in seconds', YTITLE='amplitude',}$
  \quad$\small
  \text{TITLE='Portion of Sampled Time Signal u(k)'}$
```

Alternately, you can run the following batch file to create the plot:

```idl
@sigprc02
See “Running the Example Code” on page 398 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” on page 474 of the IDL Reference Guide.

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 \( \delta \):

\[
f(m) = \frac{m}{N \delta}
\]

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 the example below).
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. Figure 15-2 displays the real and imaginary parts of the spectrum $v(m)$ of the sampled signal $u(k)$ for frequencies from $-(N/2-1)/(N \cdot \delta)$ to $(N/2)/(N \cdot \delta)$ cycles per second.

Enter the following commands at the IDL prompt to create the plot:

```idl
@sigprc01
V = FFT(U)
M = (INDGEN(N)-(N/2-1))
F = M / (N*delt)
```

- @sigprc01: Compute time sequence data.
- V = FFT(U): Compute spectrum v.
- M = (INDGEN(N)-(N/2-1)): Frequencies corresponding to m in cycles/second.

Figure 15-2: Real and Imaginary parts of the sample signal.
Using IDL
Displaying FFT Results

Set up for two plots in window.

!P.MULTI = [0, 1, 2]

PLOT, F, FLOAT(SHIFT(V,N/2-1)), $
  \text{YTITLE='real part of spectrum', }$
  \text{XTITLE='Frequency in cycles / second', }$
  \text{XRANGE=[-1,1]/(2*delt), XSTYLE=1, }$
  \text{TITLE='Spectrum of } u(k)\text{'}$

PLOT, F, IMAGINARY(SHIFT(V,N/2-1)), $
  \text{YTITLE='imaginary part of spectrum', }$
  \text{XTITLE='Frequency in cycles / second', }$
  \text{XRANGE=[-1,1]/(2*delt), XSTYLE=1}$

!P.MULTI = 0

Alternately, you can run the following batch file to create the plot:

@sigprc03

See “Running the Example Code” on page 398 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 1-dimensional vector of \( N \) real values, the result is an \( N \)-element vector of complex quantities, which takes \( 2N \) 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:

\[
\text{IMAGINARY}(v(0)) = 0.0 \\
\text{IMAGINARY}(v(N/2)) = 0.0
\]

and

\[
v(N-m) = \text{CONJ}(v(m)) \quad \text{for } m=1 \text{ to } N/2-1 \\
\text{N-2 redundant values}
\]

so that exactly \( N \) of the single- or double-precision values used to represent the frequency spectrum are redundant. This redundancy is evident in Figure 15-2 above. 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 (antisymmetric 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:

\[ F = \text{FINDGEN}(N/2+1)/(N*delt) \]
\[ f = [0.0, 1.0/(N*delt), \ldots, 1.0/(2.0*delt)] \]

**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's) and the phase is plotted in degrees, against frequency on a logarithmic scale. The magnitude and phase of our sample signal are plotted in Figure 15-3.

Enter the following commands at the IDL prompt to create the plot:

```
f = [0.0, 1.0/(N*delt), \ldots, 1.0/(2.0*delt)]
```
@sigprc01
Compute time sequence data.

V = FFT(U)
Compute spectrum v.

F = FINDGEN(N/2+1) / (N*delt)
F = [0.0, 1.0/(N*delt), ..., 1.0/(2.0*delt)].

mag = ABS(V(0:N/2))
Magnitude of first half of v.

phi = ATAN(V(0:N/2))
Phase of first half of v.

!P.MULTI = [0, 1, 2]
Set up for two plots in window.

Create log plots of magnitude in dB and phase in degrees:

PLOT, F, 20*ALOG10(mag), YTITLE='Magnitude in dB', $ 
XTITLE='Frequency in cycles / second', /XLOG, $ 
XRANGE=[1.0,1.0/(2.0*delt)], XSTYLE=1, $ 
TITLE='Spectrum of u(k)'

PLOT, F, phi/$!DTOR, YTITLE='Phase in degrees', $ 
YRANGE=[-180,180], YSTYLE=1, YTICKS=4, YMINOR=3, $ 
XTITLE='Frequency in cycles / second', /XLOG, $ 
XRANGE=[1.0,1.0/(2.0*delt)], XSTYLE=1

!P.MULTI = 0
Alternately, you can run the following batch file to create the plot:

@sigprc04

See “Running the Example Code” on page 398 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.

Figure 15-3 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 Figure 15-4.

Enter the following commands at the IDL prompt to create the plot:

```idl
@sigprc01
V = FFT(U)
F = FINDGEN(N/2+1) / (N*delt)
F = [0.0, 1.0/(N*delt), ... , 1.0/(2.0*delt)].

Create log-log plot of power spectrum:

PLOT, F, ABS(V(0:N/2))^2, YTITLE='Power Spectrum of u(k)', /YLOG, XTITLE='Frequency in cycles / second', /XLOG, XRANGE=[1.0,1.0/(2.0*delt)], XSTYLE=1
```

Alternately, you can run the following batch file to create the plot:

```
Figure 15-4: Power spectrum of the sample signal.
```
The "smearing" or "leakage" effect mentioned above 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:

\[ v_n = \text{FFT}(\text{HANNING}(N) \ast U) \]

The power spectrum of the Hanning windowed signal shows the mitigation of the truncation effect (see Figure 15-5).
Enter the following commands at the IDL prompt to create the plot:

```idl
@sigprc01
Compute time sequence data.
F = FINDGEN(N/2+1) / (N*delt)      F = [0.0, 1.0/(N*delt), ..., 1.0/(2.0*delt)].

v_n = FFT(HANNING(N)*U)

Create a log-log plot of power spectrum:
PLOT, F, ABS(v_n(0:N/2))^2, YTITLE='Power Spectrum', /YLOG, YRANGE=[1.0e-8,1.0], YSTYLE=1, YMARGIN=[4,4],
XTITLE='Frequency in cycles / second', /XLOG, XRANGE=[1.0,1.0/(2.0*delt)], XSTYLE=1,
TITLE='Power Spectrum of u(k) with Hanning Window (solid)!Cand without Window (dashed)'
OPLOT, F, ABS((FFT(U))(0:N/2))^2, LINESTYLE=2
Overplot without window.
```

Alternately, you can run the following batch file to create the plot:

```idl
@sigprc06
See “Running the Example Code” on page 398 if IDL does not find the batch file.
```

**Hamming Window**

The Hamming window is defined as:

\[
w(k) = 0.54 - 0.46 \cos \left( \frac{2\pi k}{N-1} \right)
\]

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 Figure 15-6).
v_m = FFT(HANNING(N, ALPHA=0.54)*U)

Create log-log plot of power spectrum:

PLOT, F, ABS(v_m(0:N/2))^2, YTITLE='Power Spectrum', $  
   /YLOG, YRANGE=[1.0e-8,1.0], YSTYLE=1, YMARGIN=[4,4], $  
   XTITLE='Frequency in cycles / second', $  
   /XLOG, XRANGE=[1.0,1.0/(2.0*delt)], XSTYLE=1, $  
   TITLE='Power Spectrum of u(k) with Hamming Window'  
   +'!C(solid) and without Window (dashed)'

OPLOT, F, ABS((FFT(U))(0:N/2))^2, LINESTYLE=2

Overplot without window.

Alternately, you can run the following batch file to create the plot:

@sigprc07

See “Running the Example Code” on page 398 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

![Power Spectrum of u(k) with Hamming Window (solid) and without Window (dashed)](image)

Figure 15-6: Power spectrum of the sample signal after applying a
Hamming window.

Using IDL

Aliasing
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 is shown in Figure 15-7.

![Power Spectrum with (solid) and without (dashed) Aliased 30 c/s Component](image)

Figure 15-7: 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 ($25 = 1/(2*\text{delt})$), 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.

You can run the following batch file to create the plot:

```plaintext
@sigprc08
```

See “Running the Example Code” on page 398 if IDL does not find the batch file.

**FFT Usage Details**

Unlike many implementations of the Fast Fourier Transform, IDL’s FFT algorithm does not require that the number of sampled data points be a power of 2. Neither does IDL pad the original signal with zeros to artificially increase the number of samples to a power of 2. Instead, IDL’s FFT function sacrifices computational efficiency, not accuracy, when the number of samples is not a power of 2. The result produced is always in accordance with the definition of the Discrete Fourier Transform.
IDL's implementation of the FFT is based on the Cooley-Tukey algorithm. The algorithm takes advantage of the fact that the DFT of a discrete time series with an even number of points is a sum of two DFTs, each half the length of the original. This idea is used recursively, each iteration subdividing the data into smaller sets to be transformed. If the number of data points $N$ in the original time series is a power of 2, the routine can use the same implementation for each subdivision. If the number of points in the original time series is not a power of 2, 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 must be transformed using a slow DFT. The slow DFT is mathematically equivalent to the FFT, but requires $N^2$ operations instead of $N \log_2(N)$.

This implementation means that IDL's FFT function is fastest when the number of points is a power of 2, but no accuracy is lost if it is not. The function is still computationally efficient if the prime factors of $N$ are rich in powers of two. The slowest case is when the number of samples is a large prime number; adding or removing one data point in this case can result in a huge improvement in efficiency.

### 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.

Figure 15-8 plots the complex analytic signal of a periodic time signal with a slowly varying amplitude.
Analytic Signal for r(t) Using Hilbert Transform

Figure 15-8: Analytic signal for r(t).

Enter the following commands at the IDL prompt to create the plot:

```
N = 1024  # Number of time samples in data set.
delt = 0.02  # Sampling interval in seconds.
T = delt * FINDGEN(N)  # Vector of discrete times.
f1 = 5.0 / ((N-1)*delt)
f2 = 0.5 / ((N-1)*delt)
R = SIN(2*!PI*f1*T) * SIN(2*!PI*f2*T)
PLOT_3DBOX, T, R, -FLOAT(HILBERT(R)), $  
   AX=40, AZ=15, XTICKS=5, XCHARSIZE=2, $  
   XTITLE = 'time in seconds', YTICKS=2, YCHARSIZE=2, $  
   YTITLE = 'real', YMARGIN=[4,8], ZTICKS=2, ZCHARSIZE=2, $  
   ZTITLE = 'imaginary'
XYOUTS, 0.5, 0.95, /NORMAL, ALIGNMENT=0.5, SIZE=1.5, $  
   'Analytic Signal for r(t) Using Hilbert Transform'
```

Alternately, you can run the following batch file to create the plot:
```
@sigprc09
```

See “Running the Example Code” on page 398 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” on page 1348 of the IDL Reference Guide for an example using the wavelet transform.

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 Filter section below.

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 466 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 below.

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.

Figure 15-9 plots a bandstop filter which suppresses frequencies between 7 cycles per second and 15 cycles per second for data sampled every 0.02 seconds.

Enter the following commands at the IDL prompt to create the plot:

delt = 0.02  # Sampling period in seconds
f_low = 15.  # Frequencies above f_low will be passed.
f_high = 7.  # Frequencies below f_high will be passed.
a_ripple = 50.  # Ripple amplitude will be less than -50 dB.
terms = 40  # The order of the filter.
bs_ir_k = DIGITAL_FILTER(f_low*2*delt, f_high*2*delt, $
a_ripple, nterms)  # Compute the impulse response = the filter coefficients.
The frequency response of the filter is the FFT of its impulse response:

\[ nfilt = \text{N\_ELEMENTS}(bs\_ir\_k) \]

\[ bs\_fr\_k = \text{FFT}(bs\_ir\_k) \times nfilt \]

Create a log plot of magnitude in decibels.

\[ f\_filt = \text{FINDGEN}(nfilt/2+1) / (nfilt*delt) \]

\[ mag = \text{ABS}(bs\_fr\_k(0:nfilt/2)) \]

The frequency response of the filter is the FFT of its impulse response:

\[ nfilt = \text{number of points in impulse response} \]

\[ bs\_fr\_k = \text{Scale frequency response by number of points} \]

Create a log plot of magnitude in decibels.

\[ f\_filt = \text{FINDGEN}(nfilt/2+1) / (nfilt*delt) \]

\[ mag = \text{ABS}(bs\_fr\_k(0:nfilt/2)) \]

\[ \text{PLOT, } f\_filt, 20\times\text{ALOG10}(mag), \text{YTITLE='Magnitude in dB'}, \text{XTITLE='Frequency in cycles / second'}, \text{/XLOG, } \]

\[ \text{XRANGE=[1.0,1.0/(2.0*delt)]}, \text{XSTYLE=1, } \]

\[ \text{TITLE='Frequency Response for Bandstop\textregistered CFIR Filter (Kaiser)'} \]

Alternately, you can run the following batch file to create the plot:

@sigprc10

See “Running the Example Code” on page 398 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 407), 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.

Figure 15-10 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.

Enter the following commands at the IDL prompt to create the plot:

```idl
delt = 0.02          Sampling period in seconds.
f_low = 15.           Frequencies above f_low will be passed.
f_high = 7.           Frequencies below f_high will be passed.
nfilt = 81            The length of the filter.
f_filt = FINDGEN(nfilt/2+1) / (nfilt*delt)
ideal_fr = (f_filt GT f_low) $  Pass frequencies greater than f_low.
                              OR (f_filt LT F_high)  Pass frequencies less than f_high.
ideal_fr = FLOAT(ideal_fr)  Convert from byte to floating point.
```

Figure 15-10: Bandstop filter using Hanning Window.
Replicate to obtain values for negative frequencies:

\[ \text{ideal}_\text{fr} = [\text{ideal}_\text{fr}, \text{REVERSE}(\text{ideal}_\text{fr}[1:])] \]

Now use an inverse FFT to get the impulse response of the ideal filter:

\[ \text{ideal}_\text{ir} = \text{FLOAT}(\text{FFT}(\text{ideal}_\text{fr}, /\text{INVERSE})) \]

\( \text{ideal}_\text{fr} \) is an even function, so the result is real.

\[ \text{ideal}_\text{ir} = \text{ideal}_\text{ir} / n_{\text{filt}} \]

Scale by the # of points.

\[ \text{ideal}_\text{ir} = \text{SHIFT}(\text{ideal}_\text{ir}, n_{\text{filt}}/2) \]

Shift it before applying the window.

Apply a Hanning window to the shifted ideal impulse response:

\[ \text{bs}_\text{ir}_n = \text{ideal}_\text{ir} \ast \text{HANNING}(n_{\text{filt}}) \]

These are the coefficients of the filter.

The frequency response of the filter is the FFT of its impulse response:

\[ \text{bs}_\text{fr}_n = \text{FFT}(\text{bs}_\text{ir}_n) \ast n_{\text{filt}} \]

Scale by the number of points.

Create a log plot of magnitude in decibels:

\[ \text{mag} = \text{ABS}(\text{bs}_\text{fr}_n(0:n_{\text{filt}}/2)) \]

Magnitude of Hanning bandstop filter transfer function.

PLOT, f_filt, 20*ALOG10(mag), YTITLE='Magnitude in dB', $
XTITLE='Frequency in cycles / second', /XLOG, $
XRANGE=[1.0,1.0/(2.0*delt)], XSTYLE=1, $
TITLE='Frequency Response for Bandstop!CFIR Filter (Hanning)'

Alternately, you can run the following batch file to create the plot:

@sigprc11

See “Running the Example Code” on page 398 if IDL does not find the batch file.

**FIR Filter Implementation**

The simplest FIR 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 Figure 15-11:

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.

Enter the following commands at the IDL prompt to create the plot:

```idl
@sigprc01
compute time data sequence u

Compute the Kaiser filter coefficients:

delt = 0.02  \hspace{1cm} \text{Sampling period in seconds}
f_low = 15.  \hspace{1cm} \text{Frequencies above } f_{\text{low}} \text{ will be passed.}
f_high = 7.  \hspace{1cm} \text{Frequencies below } f_{\text{high}} \text{ will be passed.}
a_ripple = 50. \hspace{1cm} \text{Ripple amplitude will be less than -50 dB.}
nterms = 40  \hspace{1cm} \text{The order of the filter.}
bs_ir_k = DIGITAL_FILTER(f_low*2*delt, f_high*2*delt, 
```

---

**Figure 15-11:** Digital signal before and after filtering.
Using IDL

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:

$$\frac{y(s)}{u(s)} = \frac{\left(\frac{f_0}{2\pi s} + s^2\right)}{\left(1 + 2s\left(\frac{f_0}{2\pi}\right) + s^2\right)}$$
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

\[
\frac{2}{\delta} \frac{(z-1)}{(z+1)}
\]

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:

\[
y(z) = \frac{\frac{1+c^2}{2} - 2cz + \frac{1+c^2}{2}z^2}{(c^2 - 2cz + z^2)}
\]

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:

```idl
delt = 0.02
f0 = 6.5
Notch frequency in cycles per second.
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]
```

Alternately, you can run the following batch file to compute the coefficients:

```
@sigprc13
```

See “Running the Example Code” on page 398 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

\[
y(z) = \frac{b_0 + b_1 z + \ldots + b_{nb} z^{nb}}{a_0 + a_1 z + \ldots + a_{na} z^{na}} u(z)
\]

is implemented with the difference equation
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.

Figure 15-12 plots the impulse and frequency response functions of the notch filter defined above using recursive difference equations.

Enter the following commands at the IDL prompt to create the plot:

```idl
@sigprc13

na = N_ELEMENTS(A) - 1
nb = N_ELEMENTS(B) - 1
```

- Load the coefficients for the notch filter.
- Degree of denominator polynomial.
- Degree of numerator polynomial.
Infinite Impulse Response Filters

Using IDL

N = 1024L

Create an impulse U.

U = FLTARR(N)
U[0] = FLOAT(N)
Y = FLTARR(N)

Recursively compute the filtered signal:

FOR K = 1, N-1 DO $\$
    Y(K) = ( TOTAL( B[nb-K>0:nb] * U[K-nb>0:K]) $\$

V = FFT(Y) Compute spectrum V.
F = FINDGEN(N/2+1) / (N*delt) F = [0.0, 1.0/(N*delt), ... , 1.0/(2.0*delt)]
mag = ABS(V(0:N/2)) Magnitude of first half of V.
phi = ATAN(V(0:N/2)) Phase of first half of V.

Create log plots of magnitude in decibels and phase in degrees:

!P.MULTI = [0, 1, 2] Set up for two plots in window.
PLOT, F, 20*ALOG10(mag), YTITLE='Magnitude in dB', $\$
    XTITLE='Frequency in cycles / second', /XLOG, $\$
    XRANGE=[1.0,1.0/(2.0*delt)], XSTYLE=1, $\$
    TITLE='Frequency Response Function of b(z)/a(z)'$\$
PLOT, F, phi/!DTOR, YTITLE='Phase in degrees', $\$
    XRANGE=[-180,180], YSTYLE=1, YTICKS=4, YMINOR=3, $\$
    XTITLE='Frequency in cycles / second', /XLOG, $\$
    XRANGE=[1.0,1.0/(2.0*delt)], XSTYLE=1

!P.MULTI = 0

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.

Alternately, you can run the following batch file to create the plot:

@sigprc14
See “Running the Example Code” on page 398 if IDL does not find the batch file. The same code could be used to filter any input sequence $u(k)$.

**Routines for Signal Processing**

Below is a brief description of IDL routines for signal processing. More detailed information is available in the IDL Reference Guide.

- **A_CORRELATE**(): Compute the autocorrelation or autocovariance of a sample population as a function of the lag.
- **BLK_CON()**(): Convolve an input signal with an impulse-response sequence.
- **C_CORRELATE**(): Compute the cross-correlation or cross-covariance of a sample population as a function of the lag.
- **CONVOL()**(): Convolve two vectors or arrays.
- **DIGITAL_FILTER**(): Calculate coefficients of a non-recursive digital filter.
- **FFT**(): Fast Fourier Transform.
- **HANNING()**(): Compute Hanning and Hamming windows.
- **HILBERT()**(): Construct a Hilbert matrix.
- **MEDIAN()**(): Median function and filter.
- **SMOOTH()**(): Smooth with a boxcar average.
- **WTN()**(): Wavelet transform.

**References**


Chapter 16

Mathematics

The following topics are covered in this chapter:

- IDL’s Numerical Recipes Functions ........................................... 426
- Accuracy & Floating-Point Operations .................................... 427
- Arrays and Matrices ......................................................... 429
- Correlation Analysis .......................................................... 431
- Curve and Surface Fitting .................................................... 436
- Eigenvalues and Eigenvectors .............................................. 438
- Gridding and Interpolation .................................................. 444
- Hypothesis Testing ............................................................. 446
- Integration .............................................................................. 449
- Linear Systems ................................................................. 453
- Nonlinear Equations ........................................................... 461
- Optimization ......................................................................... 462
- Sparse Arrays ...................................................................... 464
- Time-Series Analysis .......................................................... 466
- Multivariate Analysis ......................................................... 470
This chapter documents IDL'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).

This chapter also includes introductory discussions of the following topics and an overview of the way IDL handles the particular problems involved:

- Arrays and Matrices
- Correlation Analysis
- Curve and Surface Fitting
- Eigenvalues and Eigenvectors
- Gridding and Interpolation
- Hypothesis Testing
- Integration
- Linear Systems
- Nonlinear Equations
- Optimization
- Sparse Arrays
- Time Series Analysis

References are provided at the end of each section for a more detailed description and understanding of the topic.

Research Systems 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 Research Systems via:

Internet: math@rsinc.com
Fax: (303) 786-9909

**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 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.

### Accuracy & Floating-Point Operations

In a computer, real numbers are represented with finite precision. While in most cases it is safe to assume that the result of an arithmetical operation done on your computer is correct, it is important to remember that this finite-precision representation leads to unavoidable errors, especially when floating-point numbers, which are digital approximations to real numbers, are involved.

To understand why floating-point numbers are inherently inaccurate, consider the following:

- Floating-point numbers must be made to fit in a space (a string of binary digits in a computer's memory register) that can only hold an integer and a scaling factor.

- Floating-point numbers are represented by strings of a limited number of bits, but represent numbers much larger or smaller than that number of digits can be made to express.

In other words, floating-point values are finite-precision approximations of infinitely precise numbers.

### Roundoff Error

When working with floating-point arithmetic, it is helpful to consider the quantity known as the machine accuracy or the floating-point accuracy of your particular computer. This is the smallest number that, when added to 1.0, produces a floating-point result that is different from 1.0.

A useful way of thinking about machine accuracy is to consider it to be the fractional accuracy to which floating-point numbers are represented. In other words, the machine accuracy roughly corresponds to a change of the least significant bit of the floating-point mantissa—precisely what can happen if a number with more significant digits than fit in the floating-point mantissa is rounded to fit the space available. Generally speaking, every floating-point
arithmetic operation introduces an error at least equal to the machine accuracy into the result. This error is known as roundoff error.

Roundoff errors are cumulative. Depending on the algorithm you are using, a calculation involving \( n \) arithmetic operations might have a total roundoff error between \( \sqrt{n} \) times the machine accuracy and \( n \) times the machine accuracy.

Note that the machine accuracy is not the same as the smallest floating-point number your computer can represent. To find these and other machine-dependent quantities for your own computer, see “MACHAR” on page 703 of the IDL Reference Guide.

**Truncation Error**

Another type of error is also present in some numerical algorithms. Truncation error is the error introduced by the process of numerically approximating a continuous function by evaluating it at a finite number of discrete points. Often, accuracy can be increased (again at some cost of computation time) by increasing the number of discrete points evaluated.

For example, consider the process of calculating

\[
e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots + \frac{x^n}{n!}
\]

Obviously, the answer becomes more accurate as \( n \) approaches infinity. When performing the actual computation, however, a cutoff value must be specified for \( n \). Increasing \( n \) reduces truncation error at the expense of computational effort.

Several IDL routines allow you to specify cutoff values in such cases (see, for example, “INT_2D” on page 563 of the IDL Reference Guide). When writing your own routines in IDL, it is important to consider this trade-off between accuracy and computational time.

**Routines for Mathematical Error Assessment**

Below is a brief description of IDL routines for checking math error status and machine characteristics. More detailed information is available in the IDL Reference Guide.

- \texttt{CHECK_MATH( )} Return and clear accumulated math error status.
- \texttt{FINITE( )} Returns TRUE if argument is finite.
- \texttt{MACHAR( )} Returns machine-specific parameters that affect floating-point arithmetic.
Arrays and Matrices

In a computer, a multidimensional data set can be indexed in one of two ways. It can be indexed in column-major format, which means that the linear order of the data elements proceeds from the first element in the first column through the last element in the first column before beginning on the second column, and so on. This is the format used by C and Pascal to index data.

Alternatively, data can be indexed in row-major format, meaning that the linear order of the data elements proceeds from the first element of the first row through the last element of the first row before beginning on the second row, and so on. This is the format used by FORTRAN, and is traditionally associated with image processing, because it keeps all the elements of a single image scan line together. Because IDL's origins are in image processing, it indexes data in row-major format.

Note Many computer languages, such as C, Pascal, and Visual Basic, index 2-dimensional arrays in (row, column) order. IDL indexes arrays in (column, row) order.

For example, a two-by-two array A that looks like this on paper:

\[
\begin{bmatrix}
A_{0,0} & A_{1,0} \\
A_{0,1} & A_{1,1}
\end{bmatrix}
\]

would be indexed: \(A_{0,0}, A_{1,0}, A_{0,1}, A_{1,1}\) in IDL.

Remembering the difference between these two indexing schemes is crucial to effective use of IDL's matrix and array functions. To specify an individual element of a column-major array, you specify the row index first, then the column index. To specify an individual element of a row-major array, you specify the column index first, then the row index. Since many numerical algorithms assume data is indexed (row, column), while IDL indexes it (column, row), it is important to keep this distinction in mind.
IDL always allocates and references data in row-major format. In order to work with data in column-major format, use IDL’s TRANSPOSE function to interchange the order of the indices.

**Example**  Suppose you have an array A of data with each element set to the value of its one-dimensional subscript, stored in a column-major array, like this:

```
0 3
1 4
2 5
```

If you give this array as input to an IDL function that expects data in row-major format, IDL will calculate an answer other than the one you expect. Why? Because while you consider the second element in this array to be the number one (the second element in a column-major array), IDL considers the second element to be the number three (the second element in a row-major array).

```
PRINT, TRANSPOSE(A)  # Transpose the column-major array into row-major format and print.
```

### Symmetric Arrays

It is possible for an array to be indistinguishable from its transpose. In this case the number of columns and rows are identical and there is a symmetry between the rows of the array and the columns of its transpose. Arrays satisfying this condition are called symmetric. When dealing with symmetric arrays the use of the TRANSPOSE function is unnecessary, since $A^T = A$.

### Multiplying Arrays

IDL has two operators used to multiply arrays. To illustrate the difference between the two operators, consider the following two arrays:

- `array1 = [ [1, 2, 1], [2, -1, 2] ]`  # 3 column by 2 row array.
- `array2 = [ [1, 3], [0, 1], [1, 1] ]`  # 2 column by 3 row array.

#### The # Operator

The # operator computes array elements by multiplying the columns of the first array by the rows of the second array. The resulting array has the same number of
columns as the first array and the same number of rows as the second array. The second array must have the same number of columns as the first array has rows.

```
PRINT, array1#array2
```

IDL prints:

```
7   -1   7
2   -1   2
3    1   3
```

The ## Operator

The ## operator does what is commonly referred to as matrix multiplication. It computes array elements by multiplying the rows of the first array by the columns of the second array. The resulting array has the same number of rows as the first array and the same number of columns as the second array. The second array must have the same number of rows as the first array has columns.

```
PRINT, array1##array2
```

IDL prints:

```
2   6
4   7
```

**Note** The # and ## operators are order specific. Note also that A # B = B ## A.

**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.

\[
r = \frac{\text{covariance of } X \text{ and } Y}{\text{(standard deviation of } X\text{)(standard deviation of } Y)}
\]

or

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.

**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.

$X = [-1.8, 0.1, -0.1, 1.9, 0.5, 1.1, 1.9, 0.3, -0.2, -1.0]$
Using IDL

Correlation Analysis

Chapter 16: Mathematics

\[ Y = [ 1.5, -1.0, -0.6, 1.1, 0.7, -0.7, 1.1, -0.1, 0.6, -0.1 ] \]

Compute the correlation coefficient of \( X \) and \( Y \).

\[
\text{PRINT, CORRELATE}(X, Y)
\]

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 436 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.)

**Example** Define the independent (\( X \)) and dependent (\( Y \)) data.

\[
X = \begin{bmatrix}
0.477121, & 2.0, & 13.0, \\
0.477121, & 5.0, & 6.0, \\
\end{bmatrix}
\]
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.

\[
\text{PRINT, M\_CORRELATE}(X, Y)
\]

IDL prints:

0.877197

Define the five sample populations.

\[
X_0 = [30, 26, 28, 33, 35, 29]
X_1 = [0.29, 0.33, 0.34, 0.30, 0.30, 0.35]
X_2 = [65, 60, 65, 70, 70, 60]
X_3 = [2700, 2850, 2800, 3100, 2750, 3050]
Y = [37, 33, 32, 37, 36, 33]
\]

Compute the partial correlation of \(X_1\) and \(Y\) with the effects of \(X_0\), \(X_2\) and \(X_3\) removed.

\[
\text{PRINT, P\_CORRELATE}(X_1, Y, \text{REFORM}([X_0, X_2, X_3], 3, 
N\_ELEMENTS(X_1)))
\]

IDL prints:

0.996017

**Routines for Computing Correlations**

Below is a brief description of IDL routines for computing correlations. More detailed information is available in the IDL Reference Guide.

- **A\_CORRELATE()** Compute the autocorrelation or autocovariance of a sample population as a function of the lag.
- **C\_CORRELATE()** Compute the cross-correlation or cross-covariance of a sample population as a function of the lag.
- **CORRELATE()** Compute the linear correlation coefficient.
- **M\_CORRELATE()** Compute the multiple correlation coefficient.
- **P\_CORRELATE()** Compute the partial correlation coefficient.
- **R\_CORRELATE()** Compute the rank correlation of two populations.
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 \( a_1 \):

```idl
PRO funct, X, A, F, PDER
IF N_PARAMS() GE 4 THEN BEGIN
pder = FLTARR(N_ELEMENTS(X), 2)
END
```

If the function is called with four parameters, calculate the partial derivatives.

\( PDER \)'s column dimension is equal to the number of elements in \( x_i \) and its row dimension is equal to the number of parameters in the function \( F \).
pder[* , 0] = 1.0 - EXP(-A[1] * X)  
Compute the partial derivatives with respect to \( a_0 \) and place in the first row of PDER.

Compute the partial derivatives with respect to \( a_1 \) and place in the second row of PDER.

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:

\begin{align*}
X &= [0.25, 0.75, 1.25, 1.75, 2.25] \\
Y &= [0.28, 0.57, 0.68, 0.74, 0.79] \\
W &= 1.0 / Y \\
A &= [1.0, 1.0] \\
yfit &= \text{CURVEFIT}(X, Y, W, A, \text{SIGMA}_A, \text{FUNCTION}_\text{NAME} = '\text{funct}') \\
\end{align*}

Compute the parameters \( a_0 \) and \( a_1 \).

PRINT, A

IDL prints:

\begin{align*}
0.787386 & \quad 1.71602 \\
\end{align*}

Thus the nonlinear function that best fits the data is:

\[ f(x) = 0.787386 \ (1 - e^{1.71602x}) \]

Routines for Curve and Surface Fitting

Below is a brief description of IDL routines for curve and surface fitting. More detailed information is available in the IDL Reference Guide.

\begin{align*}
\text{COMFIT( )} & \quad \text{Gradient-expansion least squares fit to paired data.} \\
\text{CURVEFIT( )} & \quad \text{Non-linear least squares fit to a function.} \\
\end{align*}
Eigenvalues and Eigenvectors

Consider a system of equations that satisfies the array-vector relationship $A x = \lambda x$, where $A$ is an $n \times n$ array, $x$ is an $n$-element vector, and $\lambda$ is a scalar. A scalar $\lambda$ 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 $n \times n$ array using “Householder” transformations and the QL algorithm with implicit shifts. The eigenvalues of a real, $n \times n$ 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 $n \times n$ 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 eigenvector(s) is (are) also complex.

The following examples demonstrate how to use IDL to compute the eigenvalues and eigenvectors of real, symmetric and nonsymmetric $n \times n$ 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 $A x - \lambda x = 0$; in this case $0$ denotes an $n$-element vector, all elements of which are zero.

**Example 1: Symmetric Array with $n$ Distinct Real Eigenvalues**

To compute eigenvalues and eigenvectors of a real, symmetric, $n \times n$ array, begin with a symmetric array $A$.

**Note** The eigenvalues and eigenvectors of a real, symmetric $n \times n$ array will always be real.

$$A = \begin{bmatrix} 3.0 & 1.0 & -4.0 \\ 1.0 & 3.0 & -4.0 \\ -4.0 & -4.0 & 8.0 \end{bmatrix}$$

TRIRED, A, D, E

Compute the tridiagonal form of $A$.

Compute the eigenvalues (returned in vector D) and the eigenvectors (returned in the rows of the array $A$):

TRIQL, D, E, A

PRINT, D

IDL prints:

```
2.00000  4.76837e-07  12.0000
```

The exact values are: $[2.0, 0.0, 12.0]$.

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:

**Example 2: Nonsymmetric Array with $n$ Distinct Real and**
Complex Eigenvalues

To compute the eigenvalues and eigenvectors of a real, nonsymmetric $n \times n$ array, begin with an array $A$. In this example, there are $n$ distinct eigenvalues and $n$ linearly-independent eigenvectors.

$$A = \begin{bmatrix} 1.0 & 0.0 & 2.0 \\ 0.0 & 1.0 & -1.0 \\ -1.0 & 1.0 & 1.0 \end{bmatrix}$$

1. Reduce to upper Hessenberg format.
2. Compute the eigenvalues.
3. Print the eigenvalues.

IDL prints:

$$\begin{pmatrix} 1.00000, & -1.73205 \\ 1.00000, & 1.73205 \\ 1.00000, & 0.00000 \end{pmatrix}$$

Note that the three eigenvalues are distinct, and that two are complex. Note also that complex eigenvalues of an $n \times n$ real, nonsymmetric array always occur in complex conjugate pairs.

1. Initialize a variable to contain the residual.
2. Compute the eigenvectors and the residual for each eigenvalue/eigenvector pair, using double-precision arithmetic.
3. Print the eigenvectors, which are returned as row vectors in $\text{evecs}$.

IDL prints:

$$\begin{pmatrix} 0.68168704, & 0.18789033 \\ -0.34084352, & -0.093945164 \\ 0.16271780, & -0.59035830 \end{pmatrix}$$
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 \( A \mathbf{x} - \lambda \mathbf{x} = 0 \). 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) 
( -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.

Example 3: Repeated Eigenvalues
To compute the eigenvalues and eigenvectors of a real, nonsymmetric \( n \times n \) array, begin with an array \( A \). In this example, there are fewer than \( n \) distinct eigenvalues, but \( n \) independent eigenvectors are available.

\[
A = \begin{bmatrix} 8.0 & 0.0 & 3.0 \\ 2.0 & 2.0 & 1.0 \\ 2.0 & 0.0 & 3.0 \end{bmatrix}
\]

\[
evals = \text{HQR(ELMHES(A))}
\]
Reduce \( A \) to upper Hessenberg form and compute the eigenvalues. Note that both operations can be combined into a single command.

PRINT, evals 
Print the eigenvalues.
IDL prints:

\[
\begin{pmatrix}
9.00000 & 0.00000 \\
2.00000 & 0.00000 \\
2.00000 & 0.00000 \\
\end{pmatrix}
\]

**Note** The three eigenvalues are real, but only two are distinct.

```
residual = 1
```

Initialize a variable to contain the residual.

```
evecs = EIGENVEC(A, evals, /DOUBLE, RESIDUAL=residual)
```

Compute the eigenvectors and residual, using double-precision arithmetic.

```
PRINT, evecs[*,0]
```

Print the eigenvectors.

IDL prints:

\[
\begin{pmatrix}
0.90453403 & 0.0000000 \\
0.30151134 & 0.0000000 \\
0.30151134 & 0.0000000 \\
\end{pmatrix}
\]

```
PRINT, evecs[*,1]
```

\[
\begin{pmatrix}
-0.27907279 & 0.0000000 \\
-0.78140380 & 0.0000000 \\
0.55814557 & 0.0000000 \\
\end{pmatrix}
\]

```
PRINT, evecs[*,2]
```

\[
\begin{pmatrix}
-0.27907279 & 0.0000000 \\
-0.78140380 & 0.0000000 \\
0.55814557 & 0.0000000 \\
\end{pmatrix}
\]

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
```

```
```

```
PRINT, evecs[*,2]
```

IDL prints:

\[
\begin{pmatrix}
-0.33333333 & 0.0000000 \\
0.66666667 & 0.0000000 \\
0.66666667 & 0.0000000 \\
\end{pmatrix}
\]
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.

**Example 4: The “Defective” Case**

In the “defective” case, there are fewer than \( n \) distinct eigenvalues and fewer than \( n \) linearly-independent eigenvectors. Begin with an array \( A \):

\[
A = \begin{bmatrix} 2.0 & -1.0 \\ 1.0 & 0.0 \end{bmatrix}
\]

\[
evals = \text{HQR} ( \text{ELMHES}(A))
\]

Reduce \( A \) to upper Hessenberg form and compute the eigenvalues. Note that both operations can be combined into a single command.

\[
\text{PRINT, evals}
\]

IDL prints:

\[
( 1.00000, 0.00000) ( 1.00000, 0.00000)
\]

**Note** The two eigenvalues are real, but not distinct.

\[
evecs = \text{EIGENVEC}(A, \text{evals}, /\text{DOUBLE})
\]

Compute the eigenvectors, using double-precision arithmetic.

\[
\text{PRINT, evecs}[*,0]
\]

IDL prints:

\[
( 0.70710678, 0.0000000) ( 0.70710678, 0.0000000)
\]

\[
\text{PRINT, evecs}[*,1]
\]

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:

\[
evecs[*,1] = \text{EIGENVEC}(A, \text{evals}[1]+1.0e-6, /\text{DOUBLE})
\]

\[
\text{PRINT, evecs}[1,*]
\]
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
Below is a brief description of IDL routines for computing eigenvalues and eigenvectors. More detailed information is available in the IDL Reference Guide.

- **EIGENQL()** Compute eigenvectors of a real, symmetric array, given the array.
- **EIGENVEC( )** Compute eigenvectors of a real, nonsymmetric array, given the array and its eigenvalues.
- **ELMHES( )** Reduce a real, nonsymmetric array to upper-Hessenberg form.
- **HQR( )** Compute the eigenvalues of an upper-Hessenberg array.
- **TRIQL** Compute eigenvalues and eigenvectors of a real, symmetric, tridiagonal array.
- **TRIRED** Use Householder’s method to reduce a real, symmetric array to tridiagonal form.

References

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, nearest-neighbor, and kriging are among the interpolation methods used in IDL.

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

Below is a brief description of IDL routines for gridding and interpolation. More detailed information is available in the IDL Reference Guide.

**BILINEAR( )** Bilinear interpolation.

**GRID3( )** Smooth fit to a set of 3D scattered nodes.

**INTERPOL( )** Linear interpolation of vectors.

**INTERPOLATE( )** Compute linear, bilinear, or trilinear interpolates.

**KRIG2D( )** Interpolate regularly or irregularly gridded points using kriging.

**MIN_CURVE_SURF( )** Interpolate regularly or irregularly gridded points using minimum curvature spline surface.

**SPL_INIT( )** Establish interpolating spline for a data set (use with SPL_INTERP).

**SPL_INTERP( )** Compute cubic-spline interpolated values (use with SPL_INIT).

**POLAR_SURFACE( )** Interpolate a surface from polar coordinates to rectangular coordinates.

**SPLINE( )** Cubic spline interpolation.
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 distribution(s) of the sample population(s).

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**

Below is a brief description of IDL routines for hypothesis testing. More detailed information is available in the IDL Reference Guide.

- **CTI_TEST( )**: Construct contingency table from observed frequency data.
- **FV_TEST( )**: Compute the F-statistic for two sample populations.
- **KW_TEST( )**: Test the hypothesis that three or more sample populations have the same mean of distribution.
- **LNP_TEST( )**: Compute the Lomb Normalized Periodogram of two sample populations.
- **MD_TEST( )**: Test the hypothesis that a sample population is random.
- **R_CORRELATE( )**: Compute the rank correlation of two sample populations.
- **R_TEST( )**: Test the hypothesis that a binary population is random.
- **RS_TEST( )**: Test the hypothesis that two sample populations have the same mean of distribution.
- **S_TEST( )**: Test the hypothesis that two sample populations have the same mean of distribution.
- **TM_TEST( )**: Compute the student’s t-statistic for two sample populations.
- **XSQ_TEST( )**: Compute the Chi-square goodness-of-fit between observed and expected frequencies.

**References**

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**
  \[
  I = \int_{x=a}^{x=b} f(x) \, dx
  \]
  over an open or closed interval is possible using one of several routines based on well known methods developed by Romberg and Simpson.

- **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**
  \[
  I = \int_{x=a}^{x=b} \int_{y=p(x)}^{y=q(x)} f(x, y) \, dy \, dx
  \]
  over a regular or irregular region in the x-y plane is possible using an iterated “Gaussian Quadrature” routine.

- **Integration of a trivariate function**
  \[
  I = \int_{x=a}^{x=b} \int_{y=p(x)}^{y=q(x)} \int_{z=u(x,y)}^{z=v(x,y)} f(x, y, z) \, dz \, dy \, dx
  \]
  over a regular or irregular region in x-y-z space is possible using an iterated “Gaussian Quadrature” routine.
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.

Example 1: A Bivariate Function

Suppose that we wish to evaluate
\[ \int_{y=0}^{y=2} \int_{x=1}^{x=4} y \cos(x^5) \, dx \, dy \]

The order of integration is initially described as a dx-dy region in the x-y plane. Using the diagram at right, we can easily change the order to dy-dx. The integral is now of the form
\[ \int_{x=0}^{x=2} \int_{y=0}^{y=x^2} y \cos(x^5) \, dy \, dx \]

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 \( f(x, y) \):

```idl
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:

```idl
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
using IDL integration. 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.

```idl
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.

Example 2: 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):

```idl
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:

```idl
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:

```idl
FUNCTION uv_limits, X, Y
RETURN, [0.0, SQRT(4.0 - X^2 - Y^2)]
END
```

This is the exact solution to 9 decimal accuracy.
We can use the following IDL commands to determine the value of the above integral using 6, 10, 20 and 48 transformation points.

6 transformation points:

\[
\text{PRINT, INT}_3\text{D('fxyz', [-2.0, 2.0], 'pq_limits', 'uv_limits', 6)}
\]

IDL prints:

\textbf{57.417720}

10 transformation points:

\[
\text{PRINT, INT}_3\text{D('fxyz', [-2.0, 2.0], 'pq_limits', 'uv_limits', 10)}
\]

IDL prints:

\textbf{57.444248}

20 transformation points:

\[
\text{PRINT, INT}_3\text{D('fxyz', [-2.0, 2.0], 'pq_limits', 'uv_limits', 20)}
\]

IDL prints:

\textbf{57.446201}

48 transformation points:

\[
\text{PRINT, INT}_3\text{D('fxyz', [-2.0, 2.0], 'pq_limits', 'uv_limits', 48)}
\]

IDL prints:

\textbf{57.446265}

The exact solution to 6-decimal accuracy is 57.446267.

\textbf{Routines for Integration}

Below is a brief description of IDL routines for integration. More detailed information is available in the IDL Reference Guide.

\textbf{CRVLENGTH()} Compute the length of a curve with tabular representation.
Chapter 16: Mathematics

INT_2D( ) Evaluate the double integral of a bivariate function \( f(x,y) \).

INT_3D( ) Evaluate the triple integral of a trivariate function \( f(x,y,z) \).

INT_TABULATED( ) Integrate a tabulated data set \( \{ x_i, y_i = f(x_i) \} \).

QROMB( ) Evaluate integral over a closed interval using Romberg's method.

QROMO( ) Evaluate integral over an open interval using a modified Romberg's method.

QSIMP( ) Evaluate integral over a closed interval using Simpson's method.

References


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 invertability 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 $n \times n$ 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.

**Example 1: 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.
It is possible, however, to find a solution to this system that minimizes the residual error by orthogonally projecting the vector \( \mathbf{b} \) onto the two-dimensional subspace spanned by the columns of the array \( \mathbf{A} \). The projected vector is then used as the right-hand side of the system. The orthogonal projection of \( \mathbf{b} \) onto the column space of \( \mathbf{A} \) may be expressed with the array-vector product \( \mathbf{A}(\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{b} \), where \( \mathbf{A}(\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T \) is known as the projection matrix, \( \mathbf{P} \).

In this example, the array-vector product \( \mathbf{P}\mathbf{b} \) yields:

\[
\begin{bmatrix}
4.0 \\
5.0 \\
0.0
\end{bmatrix}
\]

and we wish to solve the linear system

\[
\begin{bmatrix}
1.0 & 2.0 \\
1.0 & 3.0 \\
0.0 & 0.0
\end{bmatrix}
\begin{bmatrix}
x_0 \\
x_1
\end{bmatrix}
= 
\begin{bmatrix}
4.0 \\
5.0 \\
0.0
\end{bmatrix}
\]

where \( \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = \begin{bmatrix} 2.0 \\ 1.0 \end{bmatrix} \)

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 \( \mathbf{b} \) onto the subspace spanned by the columns of the array \( \mathbf{A} \).

The following IDL commands use singular value decomposition to solve the system in a numerically stable manner. Begin with the array \( \mathbf{A} \):

\[
\mathbf{A} = \begin{bmatrix}
1.0 & 2.0 \\
1.0 & 3.0 \\
0.0 & 0.0
\end{bmatrix}
\]

\[
\mathbf{B} = \begin{bmatrix}
4.0 \\
5.0 \\
6.0
\end{bmatrix}
\]

\[
\text{SVDC, } \mathbf{A}, \mathbf{W}, \mathbf{U}, \mathbf{V}
\]

Define the right-hand side vector \( \mathbf{B} \).

Compute the singular value decomposition of \( \mathbf{A} \).
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 \( 1.0 \times 10^{-5} \) are reciprocated.

\[
N = \text{N\_ELEMENTS}(W)
\]
\[
WP = \text{FLTARR}(N, N)
\]
\[
\text{FOR } K = 0, N-1 \text{ DO }$
\]
\[
\text{IF } \text{ABS}(W(K)) \geq 1.0e-5 \text{ THEN } WP(K, K) = 1.0/W(K)
\]

We can now express the solution to the linear system as an array-vector product. (See Section 2.6 of Numerical Recipes for a derivation of this formula.)

\[
X = V \# WP \# \text{TRANSPOSE}(U) \# B
\]

Print the solution.

IDL Prints:

| 2.00000 |
| 1.00000 |

**Example 2: 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

It is now possible to express the solution \( x \) in terms of \( x_1 \) and \( x_3 \):

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:

\[
\begin{bmatrix}
-2 - 3x_1 - x_3 \\
x_1 \\
1 - x_3/3 \\
x_3
\end{bmatrix}
\]

\[
\begin{bmatrix}
1.0 \\
3.0 \\
1.0 \\
0.0
\end{bmatrix}
\]

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 \(A\):

\[
A = \begin{bmatrix}
1.0 & 3.0 & 3.0 & 2.0 \\
0.0 & 0.0 & 3.0 & 1.0 \\
0.0 & 0.0 & 0.0 & 0.0
\end{bmatrix}
\]

\[
B = [1.0, 5.0, 5.0]
\]

\[
\text{SVDC, } A, \ W, \ U, \ V
\]

Define the right-hand side vector \(B\).

Compute the decomposition of \(A\).
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 $1.0 \times 10^{-5}$ are reciprocated.

\[
N = \text{N}_\text{ELEMENTS}(W)
\]
\[
WP = \text{FLTARR}(N, N)
\]
\[
\text{FOR} \ K = 0, \ N-1 \ \text{DO} \$
\]
\[
\text{IF} \ \text{ABS}(W(K)) \ \text{GE} \ 1.0\text{e}-5 \ \text{THEN} \ WP(K, K) = 1.0/W(K)
\]

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.

\[
X = V \text{## WP ## TRANSPOSE(U) ## B}
\]

**IDL Prints:**

-0.211009
-0.633027
0.963303
0.110092

### Example 3: 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:

\[
\begin{bmatrix}
-1 + 0i & 1 - 3i & 2 + 0i & 3 + 3i \\
-2 + 0i & -1 + 3i & 0 + 1i & 3 + 1i \\
3 + 0i & 0 + 4i & 0 - 1i & 0 - 3i \\
2 + 0i & 1 + 1i & 2 + 1i & 2 + 1i
\end{bmatrix}
\begin{bmatrix}
z_0 \\
z_1 \\
z_2 \\
z_3
\end{bmatrix}
= \begin{bmatrix}
15 - 2i \\
12 - 1i \\
10 - 10i \\
-20 + 11i
\end{bmatrix}
\]

First we define the real part of the complex coefficient array:

```idl
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:

\[
\text{im} = \begin{bmatrix}
0 & -3 & 0 & 3 \\
0 & 3 & 1 & 1 \\
0 & 4 & -1 & -3 \\
0 & 1 & 1 & 1
\end{bmatrix}
\]

\[A = \text{COMPLEX}(\text{re}, \text{im})\]

Combine the real and imaginary parts to form a single complex coefficient array.

\[B = [\text{COMPLEX}(15,-2), \text{COMPLEX}(-2,-1), \text{COMPLEX}(-20,11), \text{COMPLEX}(-10,10)]\]

Define the right-hand side vector B.

Compute the solution using double-precision complex arithmetic.

\[z = \text{LU_COMPLEX}(A, B, /\text{DOUBLE})\]

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, \(Az-b\):

IDL prints:

\[
\begin{bmatrix}
0.00000 & 0.00000 \\
0.00000 & 0.00000 \\
0.00000 & 0.00000 \\
0.00000 & 0.00000
\end{bmatrix}
\]

**Routines for Solving Simultaneous Linear Equations**

Below is a brief description of IDL routines for solving simultaneous linear equations. More detailed information is available in the IDL Reference Guide.

- **CHOLDC**: Construct the Cholesky decomposition of an array.
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHOLSOL()</td>
<td>Solve sets of linear equations (use with CHOLDC).</td>
</tr>
<tr>
<td>COND()</td>
<td>Compute the condition number of a square array.</td>
</tr>
<tr>
<td>CRAMER()</td>
<td>Solve a linear system using Cramer’s rule.</td>
</tr>
<tr>
<td>DETERM()</td>
<td>Compute the determinant of a square array.</td>
</tr>
<tr>
<td>GS_ITER()</td>
<td>Solve a linear system using Gauss-Seidel iteration.</td>
</tr>
<tr>
<td>IDENTITY()</td>
<td>Create an identity array.</td>
</tr>
<tr>
<td>INVERT()</td>
<td>Invert a square array.</td>
</tr>
<tr>
<td>LU_COMPLEX()</td>
<td>Solve a complex linear system or invert a complex array.</td>
</tr>
<tr>
<td>LUDC</td>
<td>Construct the LU Decomposition of an array.</td>
</tr>
<tr>
<td>LUMPROVE()</td>
<td>Iteratively improve the solution vector of a set of linear equations.</td>
</tr>
<tr>
<td>LUSOL()</td>
<td>Solve sets of linear equations (use with LUDC).</td>
</tr>
<tr>
<td>NORM()</td>
<td>Compute the infinity norm of a square array or the Euclidean norm of a vector.</td>
</tr>
<tr>
<td>SVDC()</td>
<td>Construct the Singular Value Decomposition of an array.</td>
</tr>
<tr>
<td>SVSOL()</td>
<td>Use back-substitution to solve a set of simultaneous linear equations (use with SVDC).</td>
</tr>
<tr>
<td>TRACE()</td>
<td>Compute the trace of an array.</td>
</tr>
<tr>
<td>TRISOL()</td>
<td>Solve a tridiagonal system of linear equations.</td>
</tr>
</tbody>
</table>

References


Nonlinear Equations

The problem of finding a solution to a system of $n$ nonlinear equations, $F(x) = 0$, may be stated as follows:

given $F: \mathbb{R}^n \rightarrow \mathbb{R}^n$, find $x_*$ (an element of $\mathbb{R}^n$) 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 $n \times n$ linear system of equations.

The first step formulates an $n \times n$ linear system of equations ($Js = -F$) 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 \textsc{newton} and \textsc{broyden}, \textsc{idl} provides the \textsc{fx_root} and \textsc{fz_roots} functions.

**Routines for Solving Nonlinear Equations**

Below is a brief description of IDL routines for solving systems of nonlinear equations. More detailed information is available in the IDL Reference Guide.

- **\textsc{broyden}** ( )
  - Solve sets of non-linear equations using a globally-convergent Broyden’s method.
- **\textsc{fx_root}** ( )
  - Compute the real and complex roots of a univariate non-linear function using Müller’s method.
- **\textsc{fz_roots}** ( )
  - Compute the roots of a complex polynomial.
- **\textsc{newton}** ( )
  - Solve sets of non-linear equations using a globally-convergent Newton’s method.

**References**


**Optimization**

The problem of finding an unconstrained minimizer of an n-dimensional function, \( f \), may be stated as follows:

given \( f : \mathbb{R}^n \rightarrow \mathbb{R} \), find \( x^* \) (an element of \( \mathbb{R}^n \)) such that \( f(x^*) \) is a minimum of \( f \).

For example:

\[
\begin{align*}
&f(x) = (x_0 - 3)^4 + (x_1 - 2)^2 \\
&x^* = [3, 2]
\end{align*}
\]

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.

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

Below is a brief description of IDL routines for optimization. More detailed information is available in the IDL Reference Guide.

- **AMOEBA**
  Multidimensional minimization of a user supplied function using the downhill simplex method.

- **CONSTRAINED_MIN**
  Solves nonlinear optimization problems.

- **DFPMIN**
  Davidon-Fletcher-Powell minimization of a user supplied function.

- **POWELL**
  Powell minimization of a user supplied function.

### References


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 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.

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.

Example

Create a 500 by 500 element pseudo-random diagonally-dominant floating-point array in which approximately 50% of the elements are 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.)

\[
\text{N} = 500L \\
\text{A} = \text{CEIL}(\text{RANDOM}(\text{SEED}, \text{N}, \text{N})*10) + 0.0 \\
\text{I} = \text{WHERE}(\text{ABS}(\text{A}) \geq 8) \\
\text{A}[\text{I}] = 0.0 \\
\text{diag} = \text{TOTAL}(\text{ABS}(\text{A}), 1) \\
\text{A}((\text{INDGEN(N)} * (\text{N}+1)) = \text{diag} + 1.0 \\
\text{B} = \text{[REPLICATE}(1.0, 0.4*\text{N}), \text{REPLICATE}(2.0, 0.6*\text{N})] \\
\text{Create a right-hand side vector, b,}
\]
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.

\[
\begin{align*}
X &= \text{REPLICATE}(1.0, \text{N}_\text{ELEMENTS}(B)) \\
\text{start} &= \text{SYSTIME}(1) & \text{Start the timer.} \\
\text{result1} &= \text{LINBCG}(\text{SPRSIN}(A), B, X) & \text{Solve the system.} \\
\text{stop} &= \text{SYSTIME}(1) & \text{Stop the timer.} \\
\text{PRINT}, '\text{Time for Iterative Biconjugate Gradient:}', \text{stop} - \text{start}
\end{align*}
\]

IDL prints:

\textbf{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.

\[
\begin{align*}
\text{start} &= \text{SYSTIME}(1) & \text{Start the timer.} \\
\text{LUDC, A, index} & & \text{Compute the LU decomposition of A.} \\
\text{result2} &= \text{LUSOL}(A, \text{index}, B) & \text{Compute the solution.} \\
\text{stop} &= \text{SYSTIME}(1) & \text{Stop the timer.} \\
\text{PRINT}, '\text{Time for LU Decomposition:}', \text{stop} - \text{start}
\end{align*}
\]

IDL prints:

\textbf{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 \(1.0 \times 10^{-5}\), or a “-1” if the two results are identical to within five decimal places.

\[
\text{error} = \text{ABS(result1-result2)}
\]

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

Below is a brief description of IDL routines for handling sparse arrays. More detailed information is available in the IDL Reference Guide. Note that SPRSIN must be used to convert to sparse storage format before the other routines can be used.

- **FULSTR( )**: Restore a row-indexed sparse array to full storage format.
- **LINBCG( )**: Solve a system of linear equations using the iterative bi-conjugate method.
- **READ_SPR( )**: Read a row-indexed sparse array from a file.
- **SPRSAB( )**: Multiply two row-indexed sparse arrays.
- **SPRSAX( )**: Multiply a row-indexed sparse array by a vector.
- **SPRSIN( )**: Convert an array to row-indexed sparse storage format.
- **WRITE_SPR**: Write a row-indexed sparse array to a file.

### References


### 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 \( x = (x_0, x_1, x_2, ..., x_{n-1}) \), 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 stationarity. 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.

```idl
!P.MULTI=[0,1,2]   \[Set the plotting window to hold two plots.\]

PLOT, X   \[Plot the data.\]
```
Compute the sample autocorrelation function for time lagged values 0 - 20 and plot.

```idl
lag = INDGEN(21)
result = A_CORRELATE(X, lag)
PLOT, lag, result
```

Add a reference line at zero.

```idl
PLOTS, [0,20], [0,0], /DATA
!P.MULTI=0
```

Set the plotting window back to a single plot.

Add a reference line at zero.

Figure 16-1 shows the resulting graph.

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 \( \phi \) is estimated using the time-series data. The general autoregressive model of order \( p \) is expressed as

\[
x_t = \phi_1 x_{t-1} + \phi_2 x_{t-2} + ... + \phi_p x_{t-p} + \omega_t
\]

Modeling a stationary time-series as a pth order autoregressive process allows the extrapolation of data for future values of time. This process is known as forecasting.

**Routines for Time-Series Analysis**

Below is a brief description of IDL routines for time-series analysis. More detailed information is available in the IDL Reference Guide.

**A_CORRELATE( )**  
Compute the autocorrelation or autocovariance of a sample population.
Figure 16-1: 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.

C_CORRELATE()  Compute the cross-correlation or cross-covariance of two sample populations.
SMOOTH()       Smooth a time-series using a moving average.
TS_COEF()      Compute the coefficients used in an autoregressive time-series forecasting model.
TS_DIFF()      Compute forward differences of a time-series.
TS_FCAST()     Compute the future values of a stationary time-series.
TS_SMOOTH()    Compute central, backward, or forward moving averages of a time-series.
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 CLUSTER_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 CLUSTER_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 CLUSTER_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.”

References


Example. Define an array with 5 variables (columns) and 9 samples (rows):

```idl
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.

```idl
weights = CLUST_WTS(array, N_CLUSTERS = 4)
```

Compute the cluster assignments, for each sample, into one of four clusters.

```idl
result  = CLUSTER(array, weights, N_CLUSTERS = 4)
```

Display the cluster assignment and corresponding sample (row).

```idl
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.

<table>
<thead>
<tr>
<th>Sample</th>
<th>Var 1</th>
<th>Var 2</th>
<th>Var 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sample 1</td>
<td>2.0</td>
<td>1.0</td>
<td>3.0</td>
</tr>
<tr>
<td>Sample 2</td>
<td>4.0</td>
<td>2.0</td>
<td>3.0</td>
</tr>
<tr>
<td>Sample 3</td>
<td>4.0</td>
<td>1.0</td>
<td>0.0</td>
</tr>
<tr>
<td>Sample 4</td>
<td>2.0</td>
<td>3.0</td>
<td>3.0</td>
</tr>
<tr>
<td>Sample 5</td>
<td>5.0</td>
<td>1.0</td>
<td>9.0</td>
</tr>
</tbody>
</table>
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:

\[
z_1 = (0.87) \begin{bmatrix} 2.0 \\ 4.0 \\ 2.0 \\ 5.0 \end{bmatrix} + (0.70) \begin{bmatrix} 1.0 \\ 2.0 \\ 3.0 \\ 1.0 \end{bmatrix} + (0.69) \begin{bmatrix} 1.0 \\ 2.0 \\ 3.0 \\ 1.0 \end{bmatrix} = \begin{bmatrix} 5.0 \\ 4.0 \\ 4.0 \\ 5.0 \end{bmatrix}
\]

\[
z_2 = (0.01) \begin{bmatrix} 2.0 \\ 4.0 \\ 2.0 \\ 5.0 \end{bmatrix} + (-0.64) \begin{bmatrix} 1.0 \\ 2.0 \\ 3.0 \\ 1.0 \end{bmatrix} + (-0.66) \begin{bmatrix} 1.0 \\ 2.0 \\ 3.0 \\ 1.0 \end{bmatrix} = \begin{bmatrix} 2.0 \\ 4.0 \\ 2.0 \\ 5.0 \end{bmatrix}
\]

\[
z_3 = (0.49) \begin{bmatrix} 2.0 \\ 4.0 \\ 2.0 \\ 5.0 \end{bmatrix} + (0.32) \begin{bmatrix} 1.0 \\ 2.0 \\ 3.0 \\ 1.0 \end{bmatrix} + (-0.30) \begin{bmatrix} 1.0 \\ 2.0 \\ 3.0 \\ 1.0 \end{bmatrix} = \begin{bmatrix} 2.0 \\ 4.0 \\ 2.0 \\ 5.0 \end{bmatrix}
\]

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** 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**

Below is a brief description of IDL routines for multivariate analysis. More detailed information is available in the IDL Reference Guide.

- **CLUSTER()** Compute a cluster analysis classification of a multivariate data set.
- **CLUST_WTS()** Compute the cluster weights of a multivariate data set.
- **CORRELATE()** Compute the linear correlation coefficient.
- **CTI_TEST()** Construct contingency table from observed frequency data.
- **KW_TEST()** Test the hypothesis that three or more sample populations have the same mean distribution.
- **M_CORRELATE()** Compute the multiple correlation coefficient.
- **P_CORRELATE()** Compute the partial correlation coefficient.
- **PCOMP()** Compute the principal components and derived variables of a multivariate data set.
- **STANDARDIZE()** Compute standardized variables.

**References**


Index

This index is cross-referenced for the three volumes of the main IDL documentation set—Using IDL, Building IDL Applications, Object Graphics, and the IDL Reference Guide. Page numbers for Using IDL are followed by a “U”, page numbers for Building IDL Applications are followed by a “B”, page numbers for Object Graphics are followed by an “O”, and page numbers for the Reference Guide are followed by an “R”.

Symbols

! character 58R
IC system variable 46R
ID system variable 46R
ID.TABLE_SIZE system variable 47R, 1164R
ID.WINDOW system variable 1199R, 1317R, 1346R
DIR system variable 42R
IDL.M_PATH system variable 42R
IDLPI system variable 38R
IDTOR system variable 38R
EDIT_INPUT system variable 24U, 42R
ERR system variable 38R, 1114R, 1203R
ERROR_STATE system variable 143B, 149B, 39R, 745R, 780R, 1082R
MSG 147B, 1082R
SYS_MSG 147B
EXCEPT system variable 40R
HELP_PATH system variable 42R
JOURNAL system variable 43R, 582R
MAP system variable 359U, 38R
MAP1 system variable 723R
MORE system variable 43R
MOUSE system variable 40R, 332R
ORDER system variable 382U, 48R, 1154R, 1162R
IP system variable 49R
FONT system variable 66R
MULI system variable 306U, 162R

Index-1
Index-2

IP.T 3D system variable 323R
IPATH system variable 43R, 462R
IPI system variable 38R
IPROMQT system variable 45R
IQUIET system variable 45R, 745R
IRADEG system variable 38R
IVALUES system variable 38R
IVERSION system variable 45R
IWARN system variable 41R
IX system variable 52R
IY system variable 52R
IZ system variable 52R
* character 59R
# of Rows/Columns base property 202U
# operator 272U, 430U, 31B
## operator 273U, 431U, 31B
$ character 31U, 59R
& character 59R
` character 58R
* character 60R
. character 59R
.COMPILE executive command 32R
.CONTINUE executive command 33R
.EDIT executive command 33R
.GO executive command 33R
.OUT executive command 33R
.RETURN executive command 33R
.RUN executive command 34R
.SIZE executive command 35R
.STEP executive command 35R
.STEPOVER executive command 36R
.TRACE executive command 36R
.Xdefaults file 1211R
: character 59R
; character 59R
< operator 272U, 30B
> operator 272U, 30B
>> button 247U
? character 61R
? command 242U
: ternary operator 36B
@ character 32U, 60R
^ character 271U, 29B

A
A keyword 587R, 1098R
A_CORRELATE function 423U, 435U, 468U, 187R
abbreviating keywords 111B
aborting IDL 30U
about IDL 9U, 28, 140, 23R
ABOVE keyword 733R
ABS function 189R
ABSDEV keyword 598R
absolute deviation 755R
ABSOLUTE keyword 444R
absolute value 189R
accuracy
  of floating-point operations 427U
  of numerical algorithms 426U
ACMODE keyword 1145R
ACOS function 190R
action routines, Motif 74U
active command line 289B, 1365R
ACTIVE keyword 1271R
actual parameters 110B, 111B
Add method 182O, 286O, 354O, 389O, 397O
addition
  array elements 1123R
  operator 270U, 29B
AddPolygon method 376O
address, RSI postal 16U, 9B, 200, 29R
ADDSYSVAR, see Obsolete Routines
adjacency list, Delaunay triangulation 1131R
ADJCT, see Obsolete Routines

Numerics
24-bit images 1155R
2D rendering of 3D volumes 850R

3D
images
  reconstructed from 2D arrays 939R
  viewing coordinate system 323R
rendering 740R
transformations 343U, 314R, 380R, 984R,
  985R, 1095R, 1109R, 1186R
volume slices 1027R
3D text 820
64-bit data type
  long 12B
  unsigned long 12B
64-bit integer
  arrays 593R, 686R
  data type, converting to 689R
  vectors 686R

Index Numerics ... AUing IDL, Building IDL Applications, Object Graphics, Reference Guide Index
Adobe
  Font Metrics files 854R
  Type Manager 74R, 136R
ADVANCE keyword 725R
AFTER keyword 229R
A ITOFF keyword 724R
Aitoff map projection 365U, 724R
Aitoff map projection 365U, 724R
Adobe
  Font Metrics files 854R
  Type Manager 74R, 136R
ADVANCE keyword 725R
AFTER keyword 229R
A ITOFF keyword 724R
Aitoff map projection 365U, 724R
Aitoff map projection 365U, 724R
Alber's equal area conic map projection 724R
Alber's equal-area conic map projection 372U
ALBERS keyword 724R
ALIAS keyword 286O
aliasing, sampled data analysis 409U
ALIGN_BOTTOM keyword 1205R
ALIGN_CENTER keyword 1205R, 1222R, 1280R
ALIGN_LEFT keyword 1205R, 1222R, 1280R
ALIGN_RIGHT keyword 1205R, 1222R, 1280R
ALIGN_TOP keyword 1205R
aligning text 1379R
Alignment 203U
Alignment base property 203U
Alignment button property 213U
ALIGNMENT keyword 384O, 671R, 1230R, 1298R, 1379R
Alignment label property 221U
alignment of text objects 82O
Alignment table property 234U
ALL keyword 185O, 187O, 208O, 221O, 232O, 240O,
  250O, 257O, 263O, 272O, 280O, 290O, 292O,
  297O, 304O, 311O, 318O, 326O, 336O, 346O,
  356O, 357O, 362O, 371O, 383O, 391O, 394O,
ALL_EVENTS keyword 368R, 1298R
ALL KEYS keyword 537R
ALL_TABLE_EVENTS keyword 1230R
ALL_TEXT_EVENTS keyword 1231R, 1309R
ALL_VALUE keyword 246R
ALLOCATE_HEAP keyword 857R, 860R
allocated memory, returning amount of 539R
Allow Closing base property 203U
Allow Moving base property 204U
 ALOG function 191R
 ALOG10 function 192R
alpha blending 1260
alpha blending (image objects) 1150
alpha channel 1140
ALPHA keyword 529R, 677R
Altura Software 250U
AM_PM keyword 209O
AMBIENT keyword 407O
AMOEBA function 463U, 193R
ampersand 59R
analytic signal 411U
AND operator 274U, 32B
Angstrom symbol 68R
animation 343U
  compound widget 275B
  flickering images 485R
  MPEG files 757R, 758R, 760R, 762R
  widget interface 341R, 1357R
ANISOTROPY keyword 251O
ANNOTATE procedure 196R
annotations
  help topics 244U
  of displayed images 196R
  on plots 293U
  text objects 81O
anonymous structures 262U, 42B
ANOVA, see Obsolete Routines
ANOVA_UNEQUAL, see Obsolete Routines
ANSI keyword 1001R
apostrophe 58R
APP_KEYPAD keyword 1001R
APP_MBAR keyword 1206R
APP_SCROLL keyword 1252R
APPEND keyword 784R, 1232R, 1337R
AppleScript 438R
applications, written in IDL 2B
approximating models, statistical 282R
arc-cosine 190R
architecture, current version in use 45R
arc-sine 203R
arc-tangent 206R
ARG_PRESENT function 124B, 198R
arguments
  checking existence of 198R
  described 184R
arguments, described 180O
arithmetic errors 151B
ARRAY keyword 463R, 918R
array operators
  CHOLDC 267R
  CHOLSOL 268R
  COND 290R
  CRAMER 319R
  DETERM 417R
  EIGENVEC 446R
  ELMHES 448R
Index

GS_ITER 525R
HQR 551R
INVERT 576R
LU_COMPLEX 694R
LUDC 696R
LUMPROVE 697R
LUSOL 699R
NORM 769R
SVDC 1096R
SVSOL 1102R
TRIQI 1140R
TRIRED 1142R
TRISOL 1143R

arrays
and matrices 429U
changing dimensions of 945R
concatenation 273U, 31B
creating
64-bit integer
(L64INDGEN function) 593R
(LON64ARR function) 686R
any type (MAKE_ARRAY function) 705R
byte
(BINDGEN function) 220R
(BYTARR function) 233R
complex
(CINDGEN function) 270R
(COMPLEXARR function) 287R
(DCINDGEN function) 395R
(DCOMPLEXARR function) 398R
double-precision
(DBLARR function) 394R
(DCINDGEN function) 395R
(DCOMPLEXARR function) 398R
(DINDGEN function) 435R
integer
(INDGEN function) 561R
(INTARR function) 570R
longword
(LINDGEN function) 603R
(LONARR function) 687R
single-precision, floating-point
(FINDGEN function) 482R
(FLTARR function) 490R
string
(SINDGEN function) 1020R
(STARR function) 1073R
unsigned 64-bit
(UL64INDGEN function) 1169R
unsigned 64-bit integer
(UL64INDGEN function) 1169R
unsigned integer
(ULINDGEN function) 1166R
unsigned longword
(ULINDGEN function) 1170R
(ULON64ARR function) 1172R
definition 260U
efficient accessing 206B
extracting sub-arrays 467R
filling with a scalar value 949R
finding number of elements in 764R
floating-point 482R
incrementing elements 548R
interactive editing tool (XVAREDIT procedure) 1378R
memory allocation under VMS 23U
multiplying 430U
of structures 49B, 949R
operators, see array operators
resizing 291R, 461R, 935R
returning
maximum value 731R
minimum value 747R
subscripts of non-zero elements 1203R
type 1022R
reversing indices 960R
rotating 972R
searching for objects 986R, 989R
shifting elements 1013R
size 1022R
sorting 1046R
sparse 464U
stored in structure form 464U
subscripts
definition 261U
ranges 64B
returning non-zero elements 1203R
summing elements 1123R
symmetric 430U
transposing 1126R
unique elements of (UNIQ function) 1175R
updating 223R
ARROW procedure 200R
ARROW_ANGLE keyword 640R
ARROW_END keyword 640R
ARROW_SIZE keyword 640R
ARROW_START keyword 640R
ARROWSIZE keyword 488R
Index

AS_STRING keyword 438R
ASCENDING keyword 444R
ASCII keyword 1331R
ASCII TEMPLATE function 202R
ASIN function 203R
ASPECT keyword 559R, 806R
assignment 258B
operator 270U
pointers 236B
statement 85B
ASSOC function 157B, 204R
associated I/O 202B
associated variables 204R
asterisk 60R
at sign (character) 60R
ATAN function 206R
ATOL keyword 692R
atomic graphic objects 28O, 37O
ATTENUATION keyword 281O
Attribute objects 30O, 38O
attributes
  draw widget 229U
  droplist 224U
  label widget 221U
  listbox 226U
  slider 222U
  table widget 234U
autocorrelation 187R
autocovariance 187R
automatic compilation 29U, 100B
automatic structure definition 54B, 252B
autoregressive moving average filters 414U
autoregressive time-series forecasting 1147R, 1149R
AVANTGARDE keyword 116R
average
  mean 755R
  median 738R
  moving 1043R, 1151R
AVERAGE_LINES keyword 116R
axes 310U, 740, 2050, 106R
date labels for 594R
direction 2090
end points 56R
gridstyles 2100, 53R
linear 56R
location 2100
logarithmic 305U, 2110, 56R
margins 53R
range 290U, 52R, 54R
range (CRANGE, EXACT, EXTEND, RANGE) 2080
scaling 288U, 54R
style 54R
system variables for 52R
thickness 212O, 55R
thickness, (XYZ)THICK keyword 107R
titles 2130, 56R, 109R
AXIS keyword 442O, 443O
axis object 290, 2050
axis objects 740
creating 740
range values 740
tick labels 800
titles 800
using 740
AXIS procedure 310U, 207R
AY keyword 323R
azimuthal equidistant map projection 365U, 724R
AZIMUTHAL keyword 724R
azimuthal map projections 362U

B

B_LENGTH keyword 225R
B_VALUE keyword 246R
"Back" button 247U
BACK_CHARACTER keyword 402R
BACK_WORD keyword 402R
BACKCAST keyword 1149R
back-face 149O
back-face culling 1490
background color 148R
  for graphics windows 451R
BACKGROUND keyword 100R, 210R, 1193R, 1363R
BACKGROUND system variable field 49R
background tasks 1247R
  for widgets 302B
backing store 132R, 144R, 1318R
  for draw widgets 351R, 1256R, 1260R
  for zoom widgets 391R
IDLDE for Macintosh 130U
Index

- IDLDE for Motif 62U
- IDLDE for Windows 106U
- BACKPROJECT keyword 964R
- backprojection 963R
- back-substitution 1102R
- backward index list (for histograms) 546R
- BACKWARD keyword 1151R
- BAD_ID keyword 1232R, 1268R
- back-substitution 1102R
- bar charts 299U
- base 10 logarithm 192R
- base, setting attributes 202U
- base, setting events 210U
- BASE_ALIGN_BOTTOM keyword 1206R
- BASE_ALIGN_CENTER keyword 1206R
- BASE_ALIGN_LEFT keyword 1206R
- BASE_ALIGN_RIGHT keyword 1206R
- BASE_ALIGN_TOP keyword 1206R
- BASE_STYLE keyword 659R
- BASELINE keyword 384O
- BASELINES keyword 211R
- BASERANGE keyword 211R
- batch
  - mode 32U
- processing 60R
- batch files 32U
  - using as startup file 33U
    - IDLDE for Macintosh 132U
    - IDLDE for Motif 64U
    - IDLDE for Windows 108U
- BEGIN statement 90B
- Bell, ringing the terminal 18B
- BELOW keyword 733R
- benchmarks 1120R
- binomial distribution 221R
- BERNOULLI function 213R
- BESEL function 214R
- BESELY function 215R
- Bessel functions
  - BESELI 213R
  - BESELJ 214R
  - BESELY 215R
- BETAI, see Obsolete Routines
- BETAL function 216R
- incomplete 554R
- big endian byte ordering 235R, 1104R
- BLR LEVEL images 1118R
- bilinear (Tustin) transform 420U
- BILINEAR function 445U, 217R
- bilinear interpolation 445U, 217R, 935R
- BILINEAR keyword 716R, 964R
- BIN keyword 1342R
- BIN_DATE function 219R
- binary interpolation 573R
- BINARY keyword 116R, 787R
- binary SAVE and RESTORE 982R
- binary trees 248B
- BINDGEN function 220R
- binomial distribution 221R
- BINOMIAL function 221R
- BINOMIAL keyword 887R, 891R
- binomial random deviates 887R, 891R
- bins, histogram 546R
- BINSIZE keyword 544R, 546R
- bit shift operation 581R
- bitmap
  - button labels 1226R, 1228R, 1352R
  - byte array 340R
  - files
    - reading (READ_BMP) 901R
    - standard file format I/O routines 228B
    - writing (WRITE_BMP) 1320R
  - labels, creating 340R
Index

Using IDL, Building IDL Applications, Object Graphics, Reference Guide Index

Bitmap button property 213U
Bitmap Editor 174U
  opening 213U
  tools 175U
BITMAP keyword 1222R
bitmaps
  adding to buttons 174U
BITS_PER_PIXEL keyword 116R
BKMAN keyword 116R
BLAS_AXPY procedure 223R
BLEND_FUNCTION keyword 264O
BLK_CON function 423U, 225R
blob coloring 596R
BLOB keyword 488R
block convolution 225R
BLOCK field 192O
BLOCK keyword 191O, 199O, 443R, 788R, 1040R,
  1052R, 1082R, 1352R, 1361R, 1369R, 1370R,
  1371R, 1377R
blocks 90B
BLUE keyword 625R
BLUE_VALUES keyword 242O, 305O
BMP files
  adding to button widgets 174U
  displaying on buttons 213U
  reading (READ_BMP) 901R
  standard file format I/O routines 228R
  supplied 174U
  writing (WRITE_BMP) 1320R
BOLD keyword 117R
BOOK keyword 117R, 781R
Bookman font 116R
bookmark menu (online help) 244U
Boolean operators 274U, 32B
BORDER_GAP keyword 273O
BOTTOM keyword 328O, 363O, 685R, 1091R, 1361R
bottom margin, setting 53R
BOTTOM_STRETCH keyword 304O, 306O
boundaries, specifying for maps 360U
BOUNDS keyword 407O, 588R, 1052R, 1129R
BOUT keyword 1052R
box charts 299U
BOX_CURSOR procedure 227R
boxcar average 1043R
boxcar filter 417U
BREAKPOINT procedure 229R
breakpoints
  debugging with 140U
  removing 229R
returning information on 537R
  setting 230R
BREAKPOINTS keyword 537R
BRIGHT keyword 329R
Bristol Technology 28U, 250U
browse << buttons (online help) 247U
BROWSE_LINES keyword 202R
BROYDEN function 462U, 231R
Broyden's method 462U, 231R
bubble sort 245B
Buffer object 32O
buffer object 216O
buffer objects 138O
  creating 139O
buffered output 449R, 491R
buffers 491R
  flushing 459R
  type-ahead 514R
BUFFSIZE keyword 784R
bugs
  debugging in IDL 138U
  reporting math 426U
  reporting problems 13U, 5B
bulletin board bases 306B, 1218R
button
  bar (online help system) 247U
  groups 354R
  labels, creating 340R
  mouse with CURSOR procedure 332R
  widgets 273B, 1222R
    bitmap labels 1226R, 1228R, 1352R
    button release events 1224R
    events returned by 1228R
    groups 354R
    setting pointer focus 1238R
    toggle 1227R
button widgets
  adding menus to 173U
  setting attributes 213U
  setting properties 212U
  using 165U
BUTTON_EVENTS keyword 1253R
BUTTON_UVALUE keyword 354R
BY_VALUE keyword 579R
BYPASS_TRANSLATION keyword 117R
BYTARR function 233R
byte
  arguments and strings 75B
arrays 220R, 233R

data type 12B

scaling values into a range of bytes 238R

swapping 235R

swapping short integers 236R

type, converting to 234R

BYTE function 76B, 234R

BYTE keyword 561R, 705R

BYTEORDER procedure 235R

BYTSCL function 384U, 238R

C_ANNOTATION keyword 320U, 328U, 301R

C_CHARSIZE keyword 320U, 328U, 301R

C_CHARTHICK keyword 301R

C_COLOR keyword 251O

C_COLORS keyword 301R

C_CORRELATE function 423U, 435U, 469U, 240R

C>Edit, see Obsolete Routines

C_FILL_PATTERN keyword 252O

C_LABELS keyword 320U, 328U, 302R

C_LINESTYLE keyword 252O, 302R

C_ORIENTATION keyword 302R

C_SPACING keyword 302R

C_THICK keyword 252O, 302R

C_VALUE keyword 252O

C0 keyword 587R

C1 keyword 587R

CALDAT procedure 242R

CALENDAR procedure 244R

CALL_EXTERNAL 1404, 1406

CALL_EXTERNAL function 245R

CALL_FUNCTION function 129B, 252R

CALL_FUNCTION procedure 129B, 253R, 254R

CALL_VM, see Obsolete Routines

calling

external modules from IDL 245R

IDL functions from a string 252R

IDL methods from a string 253R

IDL procedures from a string 254R

mechanism for procedures 121B

routines written in other languages 245R, 606R

sequence 184R

calling sequence

function methods 1800

procedure methods 1800

CALLS keyword 1488, 537R

"Cancel" button 1232R

CANCEL keyword 143B, 202R, 255R, 424R, 530R

CANCEL_BUTTON keyword 1232R

caret 271U, 29B

carrot 271U, 29B

case folding 76B

CASE statement 92B

CAST keyword 776R, 858R

CATCH keyword 1363R

CATCH procedure 143B

catch, C++ language 255R

colors

Motif colormaps see colormaps

CD procedure 257R

CDECL keyword 246R

CDF files 1387R

CEIL function 259R

cell drawing (contour method) 319U

CELL_FILL keyword 303R

CENTER keyword 312R, 964R

CENTIMETERS keyword 1154R, 1157R

central map projection 363U, 724R

CENTRAL AZIMUTH keyword 729R

CGM driver 148R

CHANGE keyword 333R

cache value event 223U

changing directories 257R

changing widget values 301B

changing working directory, Macintosh 135U

CHANNEL keyword 265O, 100R, 451R, 1154R, 1162R

CHANNEL system variable field 49R

channels (in image objects) 114O

CHAR_DIMENSIONS keyword 3840

characters

character sets 79R

non-printable 18B

plotting in graphics windows 1379R

size 1380R

special 26U

CHARSIZE keyword 100R, 711R, 725R, 1380R

CHARSIZE system variable field 49R, 52R

CHARTHICK keyword 101R, 1380R

CHARTHICK system variable field 49R

CHEBYSHEV function 260R

CHECK keyword 232R, 417R, 767R

CHECK_MATH function 428U, 151B, 261R

checkbox widgets

creating 212U

laying out 213U

setting attributes 213U
Index

- setting properties 212U
- checkboxes 212U
- using 165U
- CHI_SQR, see Obsolete Routines
- CHI_SQR1, see Obsolete Routines
- CHILD keyword 1271R
- children, of widgets 1271R
- CHISQ keyword 678R, 1098R
- CHISQR keyword 604R
- CHISQR_CVF function 265R
- CHISQR_PDF function 266R
- Chi-square distribution 265R, 266R
- Chi-square error statistic, minimizing 604R
- Chi-square goodness-of-fit test 330R, 1375R
- CHOLDC procedure 459U, 267R
- Cholesky decomposition 267R, 268R
- CHOLSOL function 460U, 268R
- chromacoded editor, Windows 112U
- CINDGEN function 270R
- CIR_3PNT procedure 271R
- class 188O
- object 250B
- structure 251B
- structures
- zeroed 252B
- CLEANUP keyword 1364R
- CLEAR keyword 229R, 846R
- CLEAR_EVENTS keyword 1232R
- clearing breakpoints 229R
- CLIENTSERVER keyword 680R
- CLIP keyword 101R, 725R
- CLIP system variable field 49R
- Clipboard object 32O
- clipboard object 228O
- clipboard objects
- creating 139O
- clipboard support, graphics windows
  - Macintosh 120U
  - Windows 88U
- clipping planes 47O
- clipping window 49R
- clock, system 1105R
- CLOSE keyword 117R, 904R, 1322R, 1359R
- CLOSE procedure 273R
- CLOSE_DOCUMENT keyword 117R
- CLOSE_FILE keyword 117R
- CLOSED keyword 303R
- closing
  - (image processing) function 433R
  - contours 303R
  - files 161B
  - open file units 273R
  - graphics output files 117R
- CLUST_WTS function 476U, 274R
- cluster
  - analysis 274R, 275R
  - weights 274R
- CLUSTER function 476U, 275R
- CMY color system 387R
- CMY keyword 387R
- coastlines 708R
- COASTS keyword 708R
- code
  - IDL GUIBuilder generated 159U
  - modifying generated 159U
- CODE keyword 1082R
- COEFF keyword 330R
- COEFFICIENTS keyword 796R
- COLD keyword 1201R
- COLMAJOR keyword 918R
- colon (character) 59R
- color 580
  - and destination objects 59O
  - and digital data 58O
  - indexed 58O
  - palette objects 600
  - RGB 59O
  - specifying color values 600
- COLOR field 192O
- color mapping
  - voxel values 125O
- Color Model
  - draw area property 229U
- color model
  - for windows 1350
  - printers 1400
- color models 580
- COLOR system variable field 49R
COLORS common block 391U
Colors draw area property 229U
column bases 1207R
Column Labels table property 234U
COLUMN_LABELS keyword 1232R, 1299R
COLUMN_MAJOR keyword 1299R
COLUMN_WIDTHS keyword 1232R, 1271R, 1299R
column-major format 429U
COLUMN_S keyword 273O
combination 473R
combining 51O
combining contour and surface plots 344U
combining transformations 51O
COMFIT function 437U, 282R
COMMAND keyword 982R
command input buffer, displaying 539R
Command Input Line
  IDLDE for Macintosh 118U
    anchoring 129U
  IDLDE for Motif 41U
  IDLDE for Windows 86U
command line options, Motif 68U, 74U
command recall 24U
  buffer 938R
command stream substitutions, Motif 74U
command-line switches 20U
commands
  displaying previously-executed 539R
    executive 31R, 46R
      .COMPILE 32R
      .CONTINUE 33R
      .EDIT 33R
      .GO 33R
      .OUT 33R
      .RETURN 33R
.RUN 34R
.SSIZE 1393R
.SKIP 35R
.STEP 35R
.STEPOVER 36R
.TRACE 36R
COMMENT_SYMBOL keyword 898R
comments 85B
common blocks 93B
widgets and 312B
common methods 27O
COMPILE executive command See commands
compiling files
IDLDE for Macintosh 123U
IDLDE for Motif 48U
IDLDE for Windows 93U
compiling functions and procedures 29U, 952R, 954R
displaying 540R
Macintosh 135U
complex
arrays, creating 270R, 287R, 395R, 398R
arrays, rounding 288R
conjugate 293R
data type 285R, 396R
numbers 258U, 17B
returning imaginary part of 560R
returning real part of 486R
returning the magnitude of 189R
polynomials 502R
complex data type 13B
COMPLEX function 285R
COMPLEX keyword 561R, 705R
COMPLEXARR function 287R
COMPLEXROUND function 288R
Component Sizing common property 197U
Composite classes
Object Graphics 144O
composite classes 144O
COMPOSITE_FUNCTION keyword 407O
compositing 126O
compound widgets 275B, 314B
animation 275B
color manipulation 275B
CW_ANIMATE 341R
CW_ARCBALL 350R
CW_BGROUP 354R
CW_CLR_INDEX 358R
CW_COLORSEL 360R
CW_DEFROI 362R
CW_DICE 366R
CW_FIELD 368R
CW_FORM 371R
CW_FSLIDER 377R
CW_OSEM 380R
CW_PDMENU 382R
CW_RGBSLIDER 387R
CW_ZOOM 390R
data entry 276B
example 188U
handling events 199U
image manipulation 276B
in IDL GUIBuilder code 188U
orientation 276B
user interface 276B
writing 316B
COMPRESS keyword 716R
COMPRESSION keyword 1337R
compression, JPEG 907R, 1324R
COMPUTE_MESH_NORMALS function 289R
ComputeBounds method 403O
Computed Tomography 963R
ComputeDimensions method 239O, 270O
Computer Graphics Metafile 148R
computing inner products 430U
CON_COLOR keyword 726R
concatenation
array 273U, 31B
string 74B
concave polygons 71O, 375O
CONCEALED keyword 998R
COND function 460U, 290R
CONDITION keyword 230R
condition number 454U, 460U, 290R
conditional expression 36B
CONNECTIVITY keyword 1132R
constants
complex 258U, 17B
decimal 14B
double-precision 258U, 16B
floating-point 258U, 16B
hexadecimal 14B
integer 257U, 14B
ivalues 257U, 15B
octal 14B
string 258U, 259U, 17B
CONSTRAN keyword 442O, 443O
CONSTRANED_MIN procedure 463U, 294R
context object 182O
"Contents" button (online help system) 247U
contents tab (online help system) 248U
context 30U, 142B
CONTEXT keyword 781R
context number 781R
CONTIENT keyword 359U, 363U, 726R
continental boundaries 708R
continents, drawing 360U
contingency table 330R
CONTINGENT, see Obsolete Routines
CONTINUE executive command See commands
CONTINUE keyword 745R, 809R
CONTINUE ON ERROR keyword 952R
CONTINUOUS keyword 1382R
contour object 290, 247O
contour objects 86O
creating 86O
contour plots 318U, 300R
closing contours 303R
filled (FILL keyword) 303R
filling 330U
indicating direction of grade 331U
labeling 328U
overlaying with images 324U, 559R
polar 815R
smoothing 329U
with images and surface plots 1015R
CONTOUR procedure 318U, 343U, 300R
contrast, gamma correction 505R
control characters
alt-F4 61R
command-period 61R
command-q 61R
control-\ 61R
control-break 61R
control-C 61R
control-D 61R
control-Y 61R
control-Z 61R
CONTROL keyword 402R
Control Panel, IDLDE for Motif 41U
modifying 72U
controlling the device cursor 395U
controls see widgets 272B
convergence criterion 768R
CONVERGENCE keyword 678R, 1201R
CONVERT_COORD function 287U, 339U, 309R
converting
colors between color systems 277R
coordinate systems 309R
converting expressions
between host and network byte ordering 235R
to 64-bit integer type 689R
to byte type 234R
to complex type 285R, 396R
to double-precision type 442R
to integer type 484R
to longword type 688R
to single-precision floating-point type 486R
to string type 1077R
to unsigned 64-bit integer type 1174R
to unsigned integer type 1167R
to unsigned longword type 1173R
convex polygons 71O, 375O
CONVOL function 423U, 311R
convolution 225R, 311R
Cooley-Tukey algorithm 411U
COORD2TO3 function 314R
coordinate conversion 51O
coordinate systems and scaling 42O
coordinate transformations 51O
coordinates
cipping 101R
convert 287U, 399R
2D to 3D 314R
between coordinate systems 338R
map coordinates 723R
systems 309R
data 285U
defining 3D systems 323R
device 286U, 102R
normal 286U, 104R
COPY keyword 118R, 992R
copying help topics 243U
copying pixels from one window to another 118R
COR keyword 964R
CORRECTED keyword 330R
correction, gamma 505R
Using IDL, Building IDL Applications, Object Graphics, Reference Guide

Index

Index-13

CORREL_MATRIX, see Obsolete Routines
CORRELATE function 435U, 476U, 315R
correlation analysis 431U
correlation/covariance matrix 315R
Kendall's tau rank 882R
lagged autocorrelation 187R
lagged crosscorrelation 240R
multiple 701R
partial 795R
Pearson's correlation 315R
Spearman's rho rank 882R
correlation coefficient 431U
CORRELATE 315R
Kendall's 882R
M_CORRELATE 701R
multiple 701R
P_CORRELATE 795R
partial 795R
Pearson 315R
R_CORRELATE 882R
rank 882R
Spearman's 882R
COS function 317R
COSH function 318R
cosine 317R
hyperbolic 318R
inverse 190R
COSINES, see Obsolete Routines
count accumulation 548R
Count method 184O
cOUNTRIES keyword 708R
country boundaries 708R
COURIER keyword 118R
COVAR keyword 678R, 1098R
CRAMER function 460U, 319R
Cramer's rule 319R
CRAMV keyword 330R
CRANGE keyword 208O
CRANGE system variable field 52R
CREATE_INSTANCE keyword 218O, 426O
CREATE_STRUCT function 321R
CREATE_VIEW procedure 323R
creating
heap variables 231B
realizing widgets 1241R
system variables 409R
windows 1317R
creating multiple 212U
CREATOR_TYPE keyword 3010
cross correlation 240R
cross covariance 240R
CROSSP function 326R
CRVLENGTH function 438U, 452U, 327R
CT_LUMINANCE function 329R
CTI_TEST function 448U, 476U, 330R
CUBE keyword 280R
cubic convolution interpolation 573R, 821R
CUBIC keyword 291R, 573R, 821R, 964R, 970R
cubic spline interpolation 445U, 1054R, 1056R
parametric 446U
Culling 1490
current IDL session, returning information on 537R
CURRENT keyword 257R, 505R
current working directory 257R
cursor
box 227R
changing appearance 119R
controlling position 395U
displaying 1157R
graphics on Tektronix terminals 126R
hiding 1157R
hourglass 301B, 1238R
positioning 1157R
reading position of 313U, 894R
returning events from draw widgets 1255R
setting to crosshair 118R
specifying pattern 119R
type 118R
CURSOR procedure 313U, 332R
and Tektronix terminals 126R
CURSOR_CROSSHAIR keyword 118R
CURSOR_IMAGE keyword 119R
CURSOR_STANDARD keyword 119R
CURSOR_XY keyword 120R
curve fitting 282R
and surface fitting 436U
COMFIT 282R
CRVLENGTH 327R
CURVEFIT 335R
GAUSS2DFIT 508R
GAUSSFIT 511R
LADFIT 598R
LINFIT 604R
LMFIT 677R
MIN_CURVE_SURF 748R
Index-14

POLYFIT 824R
POLYFITW 831R
REGRESS 947R
SPLIT 1002R
SVDFIT 1098R
CURVEFIT function 436U, 437U, 335R
customizing
  IDLDE for Macintosh 128U
  IDLDE for Motif 57U
  IDLDE for Windows 102U
cutoff value
  Chi-square distribution 265R
  F distribution 471R
  Gaussian distribution 506R
  T distribution 1107R
CUTTING_PLANE keyword 1193R
CUTTING_PLANES keyword 408O
CV_COORD function 338R
CVTTOBM function 340R
CW_ANIMATE function 341R
CW_ANIMATE_GETP procedure 345R
CW_ANIMATE_LOAD procedure 346R
CW_ANIMATE_RUN procedure 348R
CW_ARCBALL function 350R
CW_BGROUP function 354R
CW_BSELECTOR, see Obsolete Routines
CW_CLR_INDEX function 358R
CW_COLORSEL function 360R
CW_DEFROI function 362R
CW_DICE function 316B, 366R
CW_FIELD function 368R
CW_FORM function 371R
CW_FSLIDER function 377R
CW_LOADSTATE, see Obsolete Routines
CW_ORIENTATION function 380R
CW_PDMENU function 298B, 382R, 1224R
CW_RGBSLIDER function 387R
CW_SAVERESTATE, see Obsolete Routines
CW_TMPL procedure 389R
CW_ZOOM function 390R
CYCLE keyword 346R, 1358R
cyclical fluctuation 467U
cylindrical coordinates 338R
cylindrical equidistant map projection 371U, 724R
CYLINDRICAL keyword 724R
cylindrical map projections 368U

D keyword 882R, 965R
D_VALUE keyword 246R
dangling references 239B, 255B
DARK keyword 329R
data coordinates 285U
  converting to other types 309R
data entry
  compound widgets 276B
  field widget 368R
data picking 130O, 132O
  example 132O
data types
  64-bit
    long 12B
    unsigned long 12B
  byte 12B
  complex 13B
  double-precision complex 13B
  double-precision floating-point 13B
  floating-point 13B
  integer 12B
  long integer 12B
  string 13B
  unsigned
    integer 12B
    long 12B
data, tabulated 444U
DATA_NAMES keyword 1027R
DATA_START keyword 899R
DATA_VALUES keyword 2520
DATA0 keyword 408O
DATA1 keyword 408O
DATA2 keyword 408O
DATA3 keyword 408O
DATANAMES keyword 1342R
DATAx keyword 319O, 364O
DATAY keyword 319O
date
  converting calendar to Julian 583R
  converting from string to binary 219R
  converting Julian to calendar 242R
displaying calendars 244R
  labeling axes with 594R
  returning current 1105R
DATE_FORMAT keyword 594R
Index

Daubechies wavelet filter 1348R
Davidon-Fletcher-Powell minimization 463U, 420R
day, returning current 1105R
DAYS_OF_WEEK keyword 209O
DBLARR function 394R
DCINDGEN function 395R
DCL interpreter symbols
  defining 996R
  deleting 411R
DCOMPLEX function 396R
DCOMPLEX keyword 561R, 705R
DCOMPLEXARR function 398R
decomposition
  Cholesky 267R, 268R
  LU 696R, 699R
  singular value 1096R, 1103R
DECREASE keyword 987R, 990R
default button 1233R
default font 64O, 84O, 385O
DEFAULT keyword 246R, 788R
default visual class 172R
DEFAULT BUTTON keyword 1233R
DEFAULTCANCEL keyword 1233R
DEFAULT_FONT keyword 1233R
DEFAULT_NO keyword 424R
DEFINE_KEY procedure 399R
defining
  command or help path 462R
  keys 399R
  region of interest 407R
  system variables 409R
defining method routines 261B
  note for Windows 3.11 users 263B
DEFROI function 407R
DEFSYSV procedure 409R
DEGREES keyword 338R, 675R, 742R, 1132R, 1135R
Delaunay triangulation 1131R
DELAY keyword 436R
DELAY_DESTROY keyword 1233R
DELETE keyword 784R
DELETE_CHARACTER keyword 403R
DELETE_COLUMNS keyword 1233R
DELETE_CURRENT keyword 403R
DELETE_EOL keyword 403R
DELETE_LINE keyword 403R
DELETE_ROWS keyword 1234R
DELETE_SYMBOL procedure 411R
DELETE_WORD keyword 403R
deleting
  DCL interpreter symbols 411R
  variables 413R
  windows 1199R
DELMITER keyword 383R, 899R
delimiters, string 258U, 17B
DELLOG procedure 412R
DELVAR procedure 413R
DEM1 keyword 121R
DEMO keyword 680R
DEMO_MODE function 414R
density function 546R
DEPTH_CUE keyword 392O, 408O
DEPTH_Q keyword 850R
dereference operator, pointers 236B
DERIV function 415R
derived variables 472U
DERIVSIG function 416R
DESCENT keyword 222O, 233O, 347O, 419O, 430O
desensitizing widgets 1242R
destination device 134O
Destination objects 31O
destination objects 134O
  and color 590
drawing to 134O
DESTROY keyword 300B, 1234R
destroying
  widgets 300B, 1234R
  windows 1199R
destroying objects 257B
DETA1CH keyword 1027R
DETERM function 460U, 417R
determinant of a square matrix 417R
deviation, mean absolute 736R
device
  backing store 144R
  CGM 148R
coordinates 286U
converting to other types 309R
cursor, controlling 395U
display channels 49R
flags 46R
font 124R
for graphics output 111R
graphics
  independent 268B
  output 111R
height 143R
HP-GL 150R
independent graphics 268B
LJ 151R
Macintosh (MAC) 154R
Microsoft Windows (WIN) 170R
monochrome 145R
name of 46R
Null 154R
number of color table indices 47R
number of colors 46R
offset 142R
PCL 154R
PostScript 156R
Printer 156R
Register terminals 168R
resolution of 48R
size of display 48R
Tektronix 169R
width 142R
X Windows 171R
Z-buffer 178R
Device fonts 66R
device independent graphics 230
DEVICE procedure 111R, 419R
DF keyword 330R, 592R
DF24 keyword 533R
DFPM1N procedure 463U, 420R
DFR8 keyword 533R
DIAGONAL keyword 987R, 990R
DIALOG MESSAGE function 423R
DIALOG_PARENT keyword 424R, 426R, 429R, 430R
DIALOG_PICKFILE function 426R
DIALOG_PRINTERSETUP function 277B, 430R
DIALOG_PRINTJOB 140O
DIALOG_PRINTJOB function 277B, 429R
DIALOG_PRINTSETUP 140O
dialogs
  file selection 277B
  message 277B
  message dialog box 423R
  modal 423R
  printing 277B
dialogs for printing 140O
dice widget 366R
dicer 354U, 1027R
DICOM 151O
differentiation, CONVOL function 311R
digital dissolve effect 436R
digital filters 414U
Digital Imaging Communications in Medicine 151O
digital signal processing 398U
DIGITAL_FILTER function 414U, 423U, 431R
DILATE function 432R
dilation operator 432R
DIMENSION keyword 705R
DINDGEN function 435R
Direct Graphics 268B, 230
clipboard support
  Macintosh 120U
  Windows 88U
  font use 66R
direct manipulation of object graphics 1300
DIRECT_COLOR keyword 121R
DirectColor visuals 120R
direction
  light source for shaded surface plots 994R
  of grade 331U
DIRECTION keyword 209O, 2820
directories
  changing 257R
  main directory system variable 42R
  popping 837R
  printing 843R
  pushing 861R
DIRECTORY keyword 426R, 441R
disappearing variables 142B
Discrete Fourier Transform 400U
Index

Index-17

discrete wavelet transform 413U
DISP_TEXT, see Obsolete Routines
DISPLAY environment variable 23U
Display Headers table property 234U
DISPLAY keyword 432O
displaying images 381U, 1153R
displaying text
ASCII files 1354R
true-color 1154R
with intensity scaling 1164R
distributing IDL applications 2B
DITHER keyword 280R, 908R
dithering 144R, 145R
Floyd-Steinberg 124R
ordered 129R
threshold 140R
division operator 271U, 29B
DLffDICOM
DumpElements method 156O
DLffDXF
GetPalette method 199O
DLM keyword 538R
double-clicks 1290R
double-precision
arrays, creating 394R, 435R
complex data type 13B
floating-point data type 13B
type, converting to 442R
down keyword 333R
downhill keyword 331U, 252O, 303R
drags events
for floating-point slider widgets 377R
for RGB slider widgets 387R
for slider widgets 1291R, 1296R
in draw widgets 1234R, 1255R
drag keyword 377R, 387R, 1291R
draw method 217O, 229O, 287O, 343O, 416O, 425O
draw widgets 273B, 294B, 135O, 1252R
attributes 229U
backing store 231U, 1260R
changing size 1234R, 1235R
color model 229U
colors used in 229U
events 232U
determining if set 1272R
returned by 1295R
returning 1234R
elementary application using 156U
go graphics type 230U
motion events 1255R
mouse events 232U
mouse motion events 233U
obtaining window number of 1257R
properties 229U
renderer type 230U
returning events 1234R
scrolling 231U
scrolling area 231U
using 166U
viewport change events 233U
viewport move 233U
viewport, position 1236R, 1243R
viewports position 1236R, 1243R
Using IDL, Building IDL Applications, Object Graphics, Reference Guide Index
DRAW_VIEWPORT_EVENTS keyword 1234R, 1272R
DRAW_XSIZE keyword 1234R
DRAW_YSIZE keyword 1235R
DRAWABLE keyword 196R
drawing
   arrows 200R
   continents 708R
   lines (PLOTS procedure) 809R
   objects (ANNOTATE procedure) 196R
drawing to a printer object 1400
droplist widgets 274B, 1262R
   events 224U
   events returned by 1266R
   initial value 224U
   returning
      current selection 1272R
      number of elements 1272R
   select event 225U
   setting 1243R
   setting attributes 224U
   setting properties 224U
   title 224U
   using 165U
DROPLIST_NUMBER keyword 1272R
DROPLIST_SELECT keyword 1272R
DTOGFLOAT keyword 237R
DTOL keyword 523R
DTOVAX keyword 235R
DTOXDR keyword 235R
DXF object 188O
DXF_TYPE field 192O
dynamic memory, returning amount in use 539R
dynamically loaded modules, keyword 538R

E
E_CONTINENTS keyword 726R
E_CONTOUR keyword 1015R
E_GRID keyword 762R
E_HORIZON keyword 726R
E_SURFACE keyword 1015R
earth, interpolating irregularly-sampled data over 1131R
edge detection, CONVOL function 311R
edge enhancement
   ROBERTS function 969R
   SOBEL function 1045R
   EDGE_TRUNCATE keyword 312R, 1043R
EDGE_WRAP keyword 312R
EDIT executive command See commands
EDIT keyword 377R
edit menu (online help) 243U
EDIT_CELL keyword 1235R
EDITABLE keyword 1235R, 1300R, 1310R
Editetable property 235U
Editable text property 216U
draw to a printer object 1400
droplist widgets 274B, 1262R
events 224U
events returned by 1266R
initial value 224U
returning
   current selection 1272R
   number of elements 1272R
select event 225U
setting 1243R
setting attributes 224U
setting properties 224U
title 224U
using 165U
EJECT keyword 122R
electronic mail address, RSI 16U, 9B, 200, 30R
elements, number of 764R
ELLIPSOID keyword 729R
ELLIPSOID function 444U, 444R
   complex 440U
   real 439U
   repeated 441U, 443U
EIGENVALUES keyword 796R
EIGENVEC function 444U, 444R
eigenvectors 438U, 444R, 446R, 1140R
   complex 440U
   real 439U
   repeated 443U
EIGENVECTORS keyword 444R
EIGHT keyword 596R
EIGHTBIT keyword 1001R
EJECT keyword 122R
electronic mail address, RSI 16U, 9B, 200, 30R
elements, number of 764R
ELLIPSOID keyword 729R
ELLIPSOID function 444U, 444R
EMBEDDED keyword 680R
EMPTY procedure 449R
emptying
   file buffers 491R
   graphics buffers 449R
ENABLE_FORMATTING keyword 385O, 672R
ENCAPSULATED keyword 122R, 1017R
encapsulated PostScript 160R
encapsulation 250B
<table>
<thead>
<tr>
<th>Index</th>
<th>Index-19</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ENCODING keyword</strong></td>
<td>123R</td>
</tr>
<tr>
<td><strong>END statement</strong></td>
<td>90B</td>
</tr>
<tr>
<td><strong>END_FILE keyword</strong></td>
<td>403R</td>
</tr>
<tr>
<td><strong>END_LINE keyword</strong></td>
<td>403R</td>
</tr>
<tr>
<td><strong>end-of-file</strong></td>
<td>450R, 1200R</td>
</tr>
<tr>
<td><strong>ENTER_LINE keyword</strong></td>
<td>403R</td>
</tr>
<tr>
<td><strong>Entering Procedure Definitions</strong></td>
<td>118B</td>
</tr>
<tr>
<td><strong>environment variables</strong></td>
<td></td>
</tr>
<tr>
<td>adding or changing</td>
<td>997R</td>
</tr>
<tr>
<td>DISPLAY</td>
<td>23U</td>
</tr>
<tr>
<td>HLOCAL</td>
<td>245U</td>
</tr>
<tr>
<td>HOME</td>
<td>245U</td>
</tr>
<tr>
<td>IDL_ARRAY_MEMORY_SIZE</td>
<td>23U</td>
</tr>
<tr>
<td>IDL_DEVICE</td>
<td>22U</td>
</tr>
<tr>
<td>IDL_DI_R</td>
<td>22U</td>
</tr>
<tr>
<td>IDL_DLM_PATH</td>
<td>22U</td>
</tr>
<tr>
<td>IDL_HELP_PATH</td>
<td>22U</td>
</tr>
<tr>
<td>IDL_PATH</td>
<td>22U</td>
</tr>
<tr>
<td>IDL_STARTUP</td>
<td>22U, 34U</td>
</tr>
<tr>
<td>LM_LICENSE_FILE</td>
<td>22U</td>
</tr>
<tr>
<td>PATH</td>
<td>20U</td>
</tr>
<tr>
<td>IDL Help Path</td>
<td>780R</td>
</tr>
<tr>
<td>RETURN procedure</td>
<td>1082R</td>
</tr>
<tr>
<td>setting</td>
<td>519R</td>
</tr>
<tr>
<td>TERM</td>
<td>23U</td>
</tr>
<tr>
<td>UNIX</td>
<td>519R</td>
</tr>
<tr>
<td>used by IDL</td>
<td>22U</td>
</tr>
<tr>
<td><strong>EOF function</strong></td>
<td>157B, 450R</td>
</tr>
<tr>
<td><strong>EPS files</strong></td>
<td>130R</td>
</tr>
<tr>
<td><strong>EPS machine-specific parameter</strong></td>
<td>703R</td>
</tr>
<tr>
<td><strong>EPSNEG machine-specific parameter</strong></td>
<td>703R</td>
</tr>
<tr>
<td><strong>EPSTOP keyword</strong></td>
<td>296R</td>
</tr>
<tr>
<td><strong>EQ operator</strong></td>
<td>277U, 35B</td>
</tr>
<tr>
<td>object references</td>
<td>259B</td>
</tr>
<tr>
<td>pointers</td>
<td>238B</td>
</tr>
<tr>
<td><strong>EQUAL_VARIANCE, see Obsolete Routines</strong></td>
<td></td>
</tr>
<tr>
<td><strong>equal-area map projection</strong></td>
<td>372U</td>
</tr>
<tr>
<td><strong>equivalence strings</strong></td>
<td>1145R, 1146R</td>
</tr>
<tr>
<td>Erase method</td>
<td>218O, 426O</td>
</tr>
<tr>
<td>ERASE procedure</td>
<td>451R</td>
</tr>
<tr>
<td>erasing IDL windows</td>
<td>451R</td>
</tr>
<tr>
<td>ERODE function</td>
<td>452R</td>
</tr>
<tr>
<td>erosion operator, morphologic</td>
<td>452R</td>
</tr>
<tr>
<td><strong>ERRORF function</strong></td>
<td>455R</td>
</tr>
<tr>
<td><strong>errors</strong></td>
<td></td>
</tr>
<tr>
<td>default error-handling mechanism</td>
<td>142B</td>
</tr>
<tr>
<td>error bars</td>
<td>456R, 794R, 808R</td>
</tr>
<tr>
<td>error function (ERRORF)</td>
<td>455R</td>
</tr>
<tr>
<td>floating-point</td>
<td>427U</td>
</tr>
<tr>
<td>floating-point underflow</td>
<td>151B</td>
</tr>
<tr>
<td>handling</td>
<td>142B</td>
</tr>
<tr>
<td>CATCH procedure</td>
<td>143B, 255R</td>
</tr>
<tr>
<td>input/output</td>
<td>146B</td>
</tr>
<tr>
<td>ON_ERROR procedure</td>
<td>145B, 779R</td>
</tr>
<tr>
<td>ON_IOERROR procedure</td>
<td>780R</td>
</tr>
<tr>
<td>OPEN procedure</td>
<td>784R</td>
</tr>
<tr>
<td>input/output</td>
<td>146B</td>
</tr>
<tr>
<td>Macintosh</td>
<td>136U</td>
</tr>
<tr>
<td>math</td>
<td>151B</td>
</tr>
<tr>
<td>messages</td>
<td></td>
</tr>
<tr>
<td>generating (MESSAGE procedure)</td>
<td>745R</td>
</tr>
<tr>
<td>modal widget dialog</td>
<td>423R</td>
</tr>
<tr>
<td>returning text of</td>
<td>1082R</td>
</tr>
<tr>
<td>placing error status in variable</td>
<td>784R</td>
</tr>
<tr>
<td>signalling (MESSAGE procedure)</td>
<td>147B</td>
</tr>
<tr>
<td>system variables</td>
<td>149B</td>
</tr>
<tr>
<td>system variables for</td>
<td>149B</td>
</tr>
<tr>
<td><strong>ERRPLOT procedure</strong></td>
<td>456R</td>
</tr>
<tr>
<td><strong>ESC keyword</strong></td>
<td>1071R</td>
</tr>
<tr>
<td><strong>ESCAPE keyword</strong></td>
<td>403R</td>
</tr>
<tr>
<td><strong>ESTIMATES keyword</strong></td>
<td>511R</td>
</tr>
<tr>
<td>Euclidean norm</td>
<td>769R</td>
</tr>
<tr>
<td><strong>EVEN keyword</strong></td>
<td>738R</td>
</tr>
<tr>
<td>event driven programming</td>
<td>272B</td>
</tr>
<tr>
<td><strong>EVENT FUNC keyword</strong></td>
<td>355R, 358R</td>
</tr>
<tr>
<td><strong>EVENT_HANDLER keyword</strong></td>
<td>1364R</td>
</tr>
<tr>
<td><strong>EVENT PRO keyword</strong></td>
<td>237U</td>
</tr>
<tr>
<td>basic structure returned by all widgets</td>
<td>1270R</td>
</tr>
<tr>
<td>button press</td>
<td>215U</td>
</tr>
<tr>
<td>button release</td>
<td>1224R</td>
</tr>
<tr>
<td>clearing</td>
<td>1232R</td>
</tr>
<tr>
<td>common properties</td>
<td>199U</td>
</tr>
<tr>
<td>compound, handling</td>
<td>199U</td>
</tr>
<tr>
<td>destruction</td>
<td>200U</td>
</tr>
<tr>
<td>draw area mouse</td>
<td>232U</td>
</tr>
<tr>
<td>draw area mouse motion</td>
<td>233U</td>
</tr>
<tr>
<td>draw area view changes</td>
<td>233U</td>
</tr>
<tr>
<td>draw area widget</td>
<td>232U</td>
</tr>
<tr>
<td><strong>EOF function</strong></td>
<td>157B, 450R</td>
</tr>
</tbody>
</table>

Using IDL, Building IDL Applications, Object Graphics, Reference Guide Index
using IDL, building IDL applications, object graphics, reference guide

Index

draw viewport move 233U
droplist 224U
droplist select 225U
focus 210U
handling in IDL GUIBuilder code 159U, 181U, 182U, 185U, 191U
interrupting the event loop 311B
keyboard focus 311B
kill request 210U
listbox 227U
listbox selection 227U
post creation 201U
processing 1269R
realize 200U
release for buttons 214U
released by
button widgets 1228R
draw widgets 1259R
list widgets 1290R
slider widgets 1296R
text widgets 1314R
top-level base widgets 1221R
reunicorn
base resize events 1216R
handler procedure name 1272R
keyboard focus events 1215R
returning
sending to widgets 1242R
setting button 215U
slider 223U
slider change value 223U
table data invalid 239U
set cell select 237U
table column width change 238U
table data invalid 239U
table focus 238U
table insert character 239U
table insert string 239U
table text delete 238U
table text selection 240U
text delete 218U
text focus 218U
text inserts 218U, 219U
text selection 219U
text widget 218U
timer 200U, 302B
top-level base kill events 1215R
tracking 200U
widget 287B
EXACT keyword 2100, 600R
eample 1460, 1470
eample files
norm_coord.pro 530
obj_axis.pro 750
obj_logaxis.pro 760
obj_plot.pro 1000
obj_tess.pro 710
obj_vol.pro 1220
penta.pro 690
rot_text.pro 840
sel_obj.pro 1310
surf_track.pro 1100, 4410
test_surface.pro 530
trackball_define.pro 550
EXCELL keyword 1375R
exclamation point 58R, 79R
EXCLUSIVE keyword 355R, 1208R
executive commands See commands, executive
EXFREQ keyword 331R
EXIT procedure 130B, 459R
exiting IDL 21U, 459R
EXP function 460R
EXPAND procedure 461R
EXPAND_PATH function 462R
EXPAND_TO_BYTES keyword 928R
EXPINT function 465R
explicitly formatted I/O 159B, 169B
exponential
integral 465R
natural 460R
random deviates 887R, 891R
EXPOSITIONAL keyword 282R, 587R
exponentiation operator 271U, 29B
EXPOSE_EVENTS keyword 1253R
expressions
efficiency of evaluation 130B
returning information on 537R
structure of 278U, 280U, 37B, 38B
type of 278U, 37B
EXTEK keyword 2100
EXTENDED_LEGEND keyword 3640
EXTENDEDSIZE keyword 788R
external
editors, Motif 80U
sharable object 245R
EXTRA keyword (keyword inheritance) 115B
EXTRA keyword (keyword inheritance) 114B
EXTRAC function 467R
EXTRACT_SLICE function 469R
EXTRAPOLATE keyword 112R, 113R, 1197R
EXTRUSION field 1920
EYE keyword 393O
eye position 450

F
F distribution 471R, 472R
F_CVF function 471R
F_PDF function 472R
F_TEST, see Obsolete Routines
F_TEST1, see Obsolete Routines
F_VALUE keyword 246R
F77_UNFORMATTED keyword 785R
FACT keyword 1382R, 1384R
FACTORIAL function 473R
false, definition of 103B
far clipping plane 470
Fast Fourier Transform 400U
implementation 410U
Fast Fourier transform 474R
FFT function 423U, 474R
field
plots 488R, 806R
widget 368R
FIELDFONT keyword 368R
File menu (online help) 243U
file units 162B
allocating 516R
returning information about 538R
See also logical unit numbers
setting file position pointer 813R
file, saving images to 1180
FILE_LUN keyword 1023R
FILE_UNIT keyword 1180R
FILEPATH function 478R
files
CDF 1387R
closing 161B, 117R, 273R, 493R
displaying ASCII 1354R
end-of-file 211B, 1200R
file units, See file units
filenames 123R
Macintosh differences 134U
Windows differences 113U
finding 426R, 480R
finding in IDL distribution 478R
flushing file units 210B
formats
BMP 228B
GIF 228B
Interfile 228B
JPEG 228B
NRIF 228B
PICT 228B
PNG 228B
PPM 228B
SRF 228B
TIFF 228B
X11 Bitmap 228B
XWD 228B
freeing logical unit numbers 493R
HDF 1388R
help and information 207B
IDL GuiBuilder
generated 159U
generating code 179U
generating resource 179U
IDL code 179U
regeneration 180U
resource 179U
indexed 221B
input/output 156B
locating 206B
logical unit number 162B
Macintosh path 44R
Macintosh-specific information 227B
manipulation operations 206B
Message-of-the-Day
IDLDE for Macintosh 134U
IDLDE for Windows 110U
modifying generated 159U
MPEG 119O
multiple structures 205B
NetCDF 1389R
opening 160B, 783R
pointer position 210B
CUR_PTR keyword 495R
POINT_LUN procedure 813R
under Windows 114U
printing to 841R
reading
and writing, Windows 113U
ASCII data 898R
binary data from 933R
data 895R
unformatted binary data 933R
record-oriented 219B
returning information on open 537R
selecting 426R
selection dialogs 277B
size of 495R
skipping records 1026R
special functions (IOCTL function) 578R
specifying search path
  IDLDE for Macintosh 132U, 133U
  IDLDE for Motif 66U
  IDLDE for Windows 109U
updating records (REWRITE keyword) 1344R
VMS-specific information 217B
Windows-specific information 226B
with indexed organization 896R
writing formatted output 841R
writing unformatted binary data 1344R
FILES keyword 207B, 538R
FILL keyword 252O, 303R, 1177R
FILL_COLOR keyword 273O
FILL_CONTINENTS keyword 708R
FILL_DIST system variable field 46R
FILL_PATTERN keyword 328O, 827R
FILLCONTOUR, see Obsolete Routines
filled contours 330U, 303R
filling
  plotting symbols 1177R
  polygons 298U, 826R, 829R
FILLVAL keyword 461R
FILTER keyword 426R
filtering
  convolution 225R
  digital filters 431R
  digital 431R
  filenames 426R
  frequency domain 474R
  Hanning windows 529R
  histogram equalization 544R
  Lee filter algorithm 600R
  mean 1043R
  median 738R
  morphologic dilation 432R
  morphologic erosion 452R
  Roberts 969R
  Sobel 1045R
filters
  bandpass 414U
  bandstop 414U
  boxcar 417U
  digital 414U
  highpass 414U
  lowpass 414U
  notch 419U
  rectangular 417U
find tab (online help system) 249U
FIND_BY_UNAME keyword 182U, 1273R
FINDFILE function 157B, 206B, 480R
FINDGEN function 482R
Finding an appropriate view volume 470
finding files 426R
finding text
  IDLDE for Macintosh 122U
  IDLDE for Motif 46U
  IDLDE for Windows 92U
finite
  impulse response filters 414U
  numbers 483R
FINITE function 428U, 153B, 483R
FITA keyword 678R
FIX function 484R
FIX_FILTER keyword 426R
FIXED keyword 788R
fixed pixels 326U
FIXED_SIZE keyword 227R
FLAGS system variable field 46R
flashing colormaps 67U
FLICK procedure 485R
FLOAT function 486R
FLOATING keyword 368R, 1208R
floating-point
  accuracy 427U
  arithmetic 703R
  arrays 482R, 490R
  converting type to 486R
  data type 13B
  errors 151B
  mantissa 703R
  native format 235R
  precision 703R
  slider widgets 377R
  underflow errors 151B
Index

<table>
<thead>
<tr>
<th>Page</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>235R</td>
<td>XDR format</td>
</tr>
<tr>
<td>1402</td>
<td>floating-point format</td>
</tr>
<tr>
<td>487R</td>
<td>FLOOR function</td>
</tr>
<tr>
<td>142R</td>
<td>flow control</td>
</tr>
<tr>
<td>488R</td>
<td>field, plotting</td>
</tr>
<tr>
<td>490R</td>
<td>FLTARR function</td>
</tr>
<tr>
<td>157B</td>
<td>FLUSH procedure</td>
</tr>
<tr>
<td>311B</td>
<td>focus events</td>
</tr>
<tr>
<td>282O</td>
<td>FOCUS keyword</td>
</tr>
<tr>
<td>273O</td>
<td>FONT keyword</td>
</tr>
<tr>
<td>66R</td>
<td>fonts</td>
</tr>
<tr>
<td>79R</td>
<td>character sets</td>
</tr>
<tr>
<td>66R</td>
<td>device</td>
</tr>
<tr>
<td>66R</td>
<td>Direct Graphics</td>
</tr>
<tr>
<td>1017R</td>
<td>displaying vector fonts</td>
</tr>
<tr>
<td>1356R</td>
<td>displaying X Windows fonts</td>
</tr>
<tr>
<td>443R</td>
<td>editing</td>
</tr>
<tr>
<td>86R</td>
<td>examples of TrueType fonts</td>
</tr>
<tr>
<td>89R</td>
<td>examples of vector fonts</td>
</tr>
<tr>
<td>124R</td>
<td>finding current X windows font</td>
</tr>
<tr>
<td>124R</td>
<td>finding names of</td>
</tr>
<tr>
<td>124R</td>
<td>finding number of</td>
</tr>
<tr>
<td>294U</td>
<td>hardware</td>
</tr>
<tr>
<td>65R</td>
<td>Hershey</td>
</tr>
<tr>
<td>66R</td>
<td>Object Graphics</td>
</tr>
<tr>
<td>65R</td>
<td>outline</td>
</tr>
<tr>
<td>81R</td>
<td>positioning commands</td>
</tr>
<tr>
<td>853R</td>
<td>PostScript</td>
</tr>
<tr>
<td>294U</td>
<td>selecting</td>
</tr>
<tr>
<td>65R</td>
<td>specifying</td>
</tr>
<tr>
<td>129U</td>
<td>IDLDE for Macintosh</td>
</tr>
<tr>
<td>131U</td>
<td>IDLDE for Motif</td>
</tr>
<tr>
<td>65U</td>
<td>IDLDE for Windows</td>
</tr>
<tr>
<td>74R</td>
<td>TrueType</td>
</tr>
<tr>
<td>136R</td>
<td>vector</td>
</tr>
<tr>
<td>672R</td>
<td>FONTSIZE keyword</td>
</tr>
<tr>
<td>91B</td>
<td>FOR statement</td>
</tr>
<tr>
<td>680R</td>
<td>FORCE_DEMO keyword</td>
</tr>
<tr>
<td>110B</td>
<td>formal parameters</td>
</tr>
<tr>
<td>111B</td>
<td>formal parameters</td>
</tr>
<tr>
<td>181O</td>
<td>FORMAT Codes</td>
</tr>
<tr>
<td>74B</td>
<td>FORMAT keyword</td>
</tr>
<tr>
<td>299O</td>
<td>FRAME_RATE keyword</td>
</tr>
<tr>
<td>159B</td>
<td>free format I/O</td>
</tr>
<tr>
<td>1317R</td>
<td>FREE keyword</td>
</tr>
<tr>
<td>157B</td>
<td>FREE_LUN procedure</td>
</tr>
<tr>
<td>421U</td>
<td>frequency response function</td>
</tr>
<tr>
<td>338R</td>
<td>FROM_CYLIN keyword</td>
</tr>
<tr>
<td>338R</td>
<td>FROM_POLAR keyword</td>
</tr>
<tr>
<td>338R</td>
<td>FROM_RECT keyword</td>
</tr>
<tr>
<td>338R</td>
<td>FROM_SPHERE keyword</td>
</tr>
<tr>
<td>339R</td>
<td>FROM_TYPE keyword</td>
</tr>
<tr>
<td>157B</td>
<td>FSTAT function</td>
</tr>
<tr>
<td>494R</td>
<td>FTOXDR keyword</td>
</tr>
<tr>
<td>235R</td>
<td>IDLDE for Windows</td>
</tr>
</tbody>
</table>

Using IDL, Building IDL Applications, Object Graphics, Reference Guide Index
FULL_PATH keyword 782R
FULL_TRANSLATION keyword 1146R
FULL_WINDOW keyword 1040R
FULSTR function 466U, 496R
FUNCT procedure 497R
function definition statement 99B
function keys
   defining 399R, 405R
   for different keyboards 1000R
   returning definitions 537R, 538R
FUNCTION keyword 608R
function methods
   calling sequence for 1800
FUNCTION_NAME keyword 193R, 678R, 1099R
FUNCTION_VALUE keyword 194R
functions 29U, 119B
   calling sequence for 184R
   compiled 975R
   defining 99B
   definition statements 110B
   displaying compiled 540R
   forward definition 101B
FUNCTIONS keyword 975R
FV_TEST function 448U, 498R
FVALUE keyword 1132R
FX_ROOT function 462U, 500R
FZ_ROOTS function 462U, 502R

G
gamma correction 505R
GAMA function 504R
gamma function 504R
   incomplete 557R
   logarithm of 682R
GAMA keyword 3040, 3060, 887R, 891R
gamma random deviates 887R, 891R
GAMA_CT procedure 505R
GAP keyword 273O
garbage collection 536R
GAUSS, see Obsolete Routines
GAUSS_CVF function 506R
GAUSS_PDF function 507R
GAUSS2DFIT function 508R
GAUSSFIT function 483U, 511R
Gaussian
   distribution 506R, 507R
   elimination method 576R
   integral 513R
   iterated quadrature 449U, 563R, 566R
   two-dimensional fit 508R
GAUSSIINT function 513R
Gauss-Krueger map projection 369U, 725R
Gauss-Seidel iteration 525R
gE 39R
GE operator 277U, 35B
general perspective map projection 366U, 725R
general triangles 375O
GEOM keyword 2500
GEOMETRIC keyword 282R
GEOMETRY keyword 1273R
geometry of widgets 305B
GEOMX keyword 2520
GEOMY keyword 2530
GEOMZ keyword 2530
GEOTIFF keyword 921R, 1338R
GET keyword 388U, 1160R
Get method 184O
GET_BOUNDS.PRO 48O
GET_CURRENT_FONT keyword 124R
GET_DECOMPOSED keyword 124R
GET_DRAW_VIEW keyword 1236R
GET_FONTNAMES keyword 124R
GET_FONTSNUM keyword 125R
GET_GRAPHICS_FUNCTION keyword 125R
GET_KBRD function 157B, 211B, 514R
GET_LUN keyword 785R
GET_LUN procedure 157B, 273R, 493R, 516R
GET_NAMES keyword 685R
GET_PATH keyword 427R
GET_SCREEN_SIZE function 517R
GET_SCREEN_SIZE keyword 308B, 125R
GET_SYMBOL function 518R
GET_TRANSLATION keyword 281R
GET_UVALUE keyword 1236R
GET_VALUE keyword 1237R
GET_VISUAL_DEPTH keyword 125R
GET_VISUAL_NAME keyword 125R
GET_WINDOW_POSITION keyword 126R
GET_WRITE_MASK keyword 126R
GetByName method 288O, 355O, 390O, 398O
GetContents method 1900
GetContiguousPixels method 219O, 230O, 344O, 427O
GetCTM method 2060, 2480, 2610, 2790, 2890, 3160, 3250, 3350, 3600, 3810, 4040
GetEntity method 1920
| GETENV function | 519R, 520R |
| GETHELP | see Obsolete Routines |
| GetFontnames method | 219O, 231O, 345O, 417O, 427O |
| GetPalette method | 199O |
| GetRGB method | 303O |
| GetTextDimensions method | 221O, 232O, 347O, 418O, 429O |
| GFLOATTOD keyword | 237R |
| GIF files | reading 904R, standard file format I/O routines 228B, writing 1322R |
| GIN_CHARS keyword | 126R |
| GLINESTYLE keyword | 711R, 726R |
| GLINETHICK keyword | 711R, 726R |
| GLYPH_WIDTH keyword | 273O |
| GNOMIC keyword | 724R |
| gnomonic map projection | 363U, 724R |
| GO executive command | see commands |
| GOMPertz keyword | 282R |
| GOODFIT, see Obsolete Routines |
| GOTO statement | 102B |
| GOURAUD keyword | 994R |
| Gouraud shading | 349U, 994R |
| GOUT keyword | 1052R |
| grade, direction of | 331U |
| GRAPHIC_PROPERTIES keyword | 660R |
| BMP | 901R, 1320R |
| GIF | 904R, 1322R |
| Interfile | 906R |
| JPEG | 907R, 1324R |
| NRIF | 1327R |
| PICT | 910R, 1328R |
| SRF | 916R, 1333R |
| TIFF | 921R, 1337R |
| X11 bitmap | 928R |
| XWD | 930R |
| keywords (collected) | 99R |
| modes | 268B, 230 |
| object-oriented | 268B, 240 |
| printing | 28U |
| two-dimensional arrays | 318U |
| graphics hierarchy | 360 |
| graphics tree | 360 |
| GraphicsType draw area property | 230U |
| GRAPHICS_TIMES procedure | 1120R |
| GRAPHICS_TREE keyword | 2230, 2340, 3480, 4200, 4320 |
| graphs | 860 |
| graticule | 360U |
| GRAY keyword | 433R, 453R |
| GRAYSCALE keyword | 908R |
| GREEN keyword | 626R |
| GREEN_VALUES keyword | 2420, 3060 |
| GREYSCALE keyword | 2650 |
GRID3 function 445U, 522R
gridding 1134R
   and interpolation 444U
   spherical 1052R, 1131R, 1134R
GRIDSTYLE keyword 210O
GRIDSTYLE system variable field 53R
growth trends 282R
GS keyword 588R, 1053R, 1129R
GS_ITER function 460U, 525R
GT operator 277U, 36B
guard digits 703R
guaranteed GUIBuilder
   generating files 89U
   IDLDE windows 87U
GUIBuilder, See IDL GUIBuilder
H
H_EQ_CTX function 527R
H_EQ_INT function 528R
halftoning 144R
halting program execution 1070R
HAMMER keyword 724R
Hammer-Aitoff map projection 366U, 724R
Hamming window 408U
HandiHelp 242U
Handle Events common event 199U
HANDLE_CREATE, see Obsolete Routines
HANDLE_FREE, see Obsolete Routines
HANDLE_INFO, see Obsolete Routines
HANDLE_MOVE, see Obsolete Routines
HANDLE_VALUE, see Obsolete Routines
HANNING function 423U, 529R
Hanning window 407U
hardware fonts 66R
HDF (Hierarchical Data Format) files 1388R
HDF_BROWSER function 530R
HDF_READ function 533R
HEADER keyword 899R
HEADER_DEFINE keyword 1320R
heap variables 230B, 254B
   creating 231B, 857R
   destroying 856R
garbage collection 536R
   leakage 239B, 255B
   object 230B, 250B, 254B
   pointer 233B
   saving and restoring 232B
HEAP_GC procedure 536R
HEAP_VARIABLES keyword 538R
HEIGHT keyword 1354R
Hight listbox property 226U
Hight text property 216U
help
   menu, online help viewer 246U
   online 242U
   ONLINE_HELP procedure 781R
HELP keyword 384R, 1223R
HELP procedure 252U, 157B, 537R
HELP_VM, see Obsolete Routines
Help objects 310, 380
HELVETICA keyword 126R
Hershey fonts 650, 65R
Hershey, Dr. A. V. 66R
Hessenberg array or matrix 448R, 551R
Hewlett-Packard Graphics Language, see HP-GL
   hexadecimal 14B
HHLOCAL environment variable 245U
hidden line removal 1060
HIDDEN_LINES keyword 3280, 3640
HIDE keyword 2100, 2420, 2530, 2650, 2740, 2820, 2910, 3190, 3280, 3380, 3460, 3850, 4000, 4090, 641R, 657R, 672R, 1157R
hiding cursor 1157R
hierarchy 360
HIFAC keyword 683R
highpass filters 414U
high-resolution continent outlines 377U
HILBERT function 423U, 541R
   Hilbert transform 411U
HINTS keyword 4090
HIRES keyword 377U, 708R, 727R
HIST_2D function 542R
HIST_EQUAL function 544R
histogram
   equalization
      H_EQ_CTX function 527R
      interactive (H_EQ_INT function) 528R
      plot 399U
   plotting mode 297U, 105R
HISTOGRAM function 546R
HISTOGRAM keyword 3190, 650R
HLS color system 277R, 387R, 1159R
HLS keyword 388U, 388R, 1160R
HLS procedure 391U, 550R
HLS_RGB keyword 277R
home directory
   IDLDE for Macintosh 132U
   IDLDE for Motif 66U
   IDLDE for Windows 107U
HOME environment variable 245U
homogeneous coordinates 334U
HORIZON keyword 727R
HORIZONTAL keyword 1091R
horizontal slider, See slider widgets
HOTSPOT keyword 438O
hourglass cursor 301B
   (for widgets) 1238R
   saving 1269R
HOURGLASS keyword 301B, 1238R
Householder
   method 1142R
   reductions 444R
HP9000 keyword 1001R
HP-GL
   driver 150R
   files 146R
HQR function 444U, 551R
HSIZE keyword 200R
HSV color system 277R, 387R, 1159R
HSV keyword 388U, 388R, 1160R
HSV procedure 391U, 553R
HSV_RGB keyword 277R
HSV_TO_R, see Obsolete Routines
HTHICK keyword 200R
HTML 250U, 251U, 751R
HTONL keyword 236R
HTONS keywords 236R
hyperbolic
   cosine 318R
   sine 1021R
   tangent 1113R
HYPERBOLIC keyword 283R
HyperHelp 250U
HyperText Markup Language 250U, 251U, 751R
hypothesis testing 446U
   Chi-square model validation 1375R
   contingency test for independence 330R
   F-variances test 498R
   Kruskal-Wallis H-test 591R
   Lomb frequency test 683R
 Mann-Whitney U-test 977R
   median delta test 733R
   normality test 498R, 1121R
   runs test for randomness 884R
   sign test 980R
   t-means test 1121R
   Wilcoxon rank-sum test 977R
I/O, see input/output
I_VALUE keyword 247R
IBETA function 554R
IBETA machine-specific parameter 703R
IBM keyword 1001R
ICONIC keyword 1347R
ICONIFY keyword 1238R
Iconify method 430O
iconifying
   widgets 1238R
   windows 1347R
iconifying windows 1370
icons, editing 1352R
IDENTITY function 460U, 556R
IDL
   applications,distributing 2B
   Code Profiler 141U
   Development Environment (IDLDE)
      Macintosh 118U
      Motif 40U
      Windows 84U
   Direct Graphics 268B, 230
      for Macintosh 154R
      for Windows 170R
   Object Graphics 268B, 230
   online help 242U
   pointers 233B
   runtime licensing 2B
   statements 84B
IDL documentation 150
IDL GUIBuilder 150U
   #of Rows/Columns property 202U
   Alignment label property 221U
   Alignment property 203U, 213U
   Alignment table property 234U
   Allowing Closing property 203U
   Allowing Moving property 204U
   base widget attributes 202U
   base widget events 210U
base widget properties 202U
base widgets, using 165U
Bitmap Editor 174U
Bitmap property 213U
button attributes 213U
button widgets, using 165U
buttons, adding bitmaps 174U
buttons, adding menus 173U
checkbox attributes 213U
checkbox widgets, using 165U
checkboxes, creating 212U
Color Model draw area property 229U
color table example 161U
Colors draw area property 229U
Column Labels table property 234U
common events 199U
compiling and running example 162U
Component Sizing property 197U
copying or cutting widgets 177U
creating draw area, example 156U
creating multiple checkboxes 212U
creating multiple radio buttons 212U
defining menus, example 153U
defleting widgets 178U
Display Headers table property 234U
draw area events 232U
draw widget properties 229U
draw widgets, using 166U
droplist attributes 224U
droplist events 224U
droplist properties 224U
droplists, using 165U
Editable table property 235U
Editable text property 216U
event code, example 191U
event code, handling example 182U
event code, integrating interfaces 185U
event code, understanding 181U
event handler application 153U
files, generating multiple times 180U
files, IDL code 179U
files, portable resource 179U
Floating property 204U
Frame property 197U
generating code 159U, 179U
generating resources files 179U
Graphics Type draw area property 230U
Grid layout property 204U
Handle Events common event 199U
Height listbox property 226U
Height text property 216U
horizontal slider, using 165U
Initial Value droplist property 224U
Initial Value listbox property 226U
Initial Value text property 216U
integrating multiple interfaces 185U
Label property 214U
label widget attributes 221U
label widget properties 221U
label widgets, using 165U
Layout property 205U
listbox attributes 226U
listbox events 227U
listbox properties 226U
listbox widgets, using 165U
Maximum Value slider property 222U
menus, editing 171U
Minimize/Maximize property 206U
Minimum Value slider property 222U
Modal property 206U
modifying code, example 159U
moving widgets 177U
Multiple listbox property 226U
Name property 196U
No Release property 214U
On Button draw area event 232U
On Button Press event property 215U
OnCellSelect table event 237U
OnChangeValue slider event 223U
OnColWidth table event 238U
OnDelete table event 238U
OnDestroy event property 200U
OnDestroy event property 200U
OnExposure draw area event 233U
OnFocus event property 210U
OnFocus table event 238U
OnInsertCh text event 218U
OnInsertChar table event 239U
OnInsertString table event 239U
OnKillRequest event property 210U
OnMotion draw area event 233U
OnRealize event property 200U
OnSelectValue droplist event 225U
OnSelectValue listbox event 227U
OnSizeChange event property 210U
<table>
<thead>
<tr>
<th>Page Numbers</th>
<th>Index Terms</th>
</tr>
</thead>
<tbody>
<tr>
<td>219U</td>
<td>OnTextSelect text event</td>
</tr>
<tr>
<td>200U</td>
<td>OnTimer event property</td>
</tr>
<tr>
<td>200U</td>
<td>OnTracking event property</td>
</tr>
<tr>
<td>233U</td>
<td>OnViewportMoved draw area event</td>
</tr>
<tr>
<td>177U</td>
<td>parent base, changing for widget</td>
</tr>
<tr>
<td>177U</td>
<td>pasting widgets</td>
</tr>
<tr>
<td>222U</td>
<td>Position slider property</td>
</tr>
<tr>
<td>201U</td>
<td>PostCreation event property</td>
</tr>
<tr>
<td>166U</td>
<td>Properties dialog</td>
</tr>
<tr>
<td>213U</td>
<td>radio button attributes</td>
</tr>
<tr>
<td>165U</td>
<td>radio button widgets, using</td>
</tr>
<tr>
<td>212U</td>
<td>radio buttons, creating</td>
</tr>
<tr>
<td>178U</td>
<td>redoing operations</td>
</tr>
<tr>
<td>230U</td>
<td>Renderer draw area property</td>
</tr>
<tr>
<td>235U</td>
<td>Resize Columns table property</td>
</tr>
<tr>
<td>177U</td>
<td>resizing widgets</td>
</tr>
<tr>
<td>231U</td>
<td>Retain draw area property</td>
</tr>
<tr>
<td>236U</td>
<td>Row Labels table property</td>
</tr>
<tr>
<td>235U</td>
<td>Row/Column Major table property</td>
</tr>
<tr>
<td>231U</td>
<td>Scroll draw area property</td>
</tr>
<tr>
<td>206U</td>
<td>Scroll property</td>
</tr>
<tr>
<td>236U</td>
<td>Scroll table property</td>
</tr>
<tr>
<td>217U</td>
<td>Scroll text property</td>
</tr>
<tr>
<td>176U</td>
<td>selecting widgets</td>
</tr>
<tr>
<td>198U</td>
<td>Sensitive property</td>
</tr>
<tr>
<td>215U</td>
<td>setting button events</td>
</tr>
<tr>
<td>212U</td>
<td>setting button properties</td>
</tr>
<tr>
<td>216U</td>
<td>setting text widget attributes</td>
</tr>
<tr>
<td>218U</td>
<td>setting text widget events</td>
</tr>
<tr>
<td>223U</td>
<td>slider events</td>
</tr>
<tr>
<td>222U</td>
<td>slider properties</td>
</tr>
<tr>
<td>161U</td>
<td>smooth example</td>
</tr>
<tr>
<td>207U</td>
<td>Space property</td>
</tr>
<tr>
<td>151U</td>
<td>starting</td>
</tr>
<tr>
<td>222U</td>
<td>Suppress Value slider property</td>
</tr>
<tr>
<td>207U</td>
<td>System Menu property</td>
</tr>
<tr>
<td>237U</td>
<td>table events</td>
</tr>
<tr>
<td>234U</td>
<td>table widget attributes</td>
</tr>
<tr>
<td>234U</td>
<td>table widget properties</td>
</tr>
<tr>
<td>166U</td>
<td>table widgets, using</td>
</tr>
<tr>
<td>158U</td>
<td>test mode</td>
</tr>
<tr>
<td>221U</td>
<td>Text label property</td>
</tr>
<tr>
<td>216U</td>
<td>text widgets properties</td>
</tr>
<tr>
<td>165U</td>
<td>text widgets, using</td>
</tr>
<tr>
<td>208U</td>
<td>Title Bar property</td>
</tr>
<tr>
<td>224U</td>
<td>Title droplist property</td>
</tr>
<tr>
<td>207U</td>
<td>Title property</td>
</tr>
<tr>
<td>23U</td>
<td>IDL_ARRA Y_MEMORY_SIZE logical name</td>
</tr>
<tr>
<td>22U</td>
<td>IDL_DEVICE environment variable</td>
</tr>
<tr>
<td>22U</td>
<td>IDL_DIR environment variable</td>
</tr>
<tr>
<td>22U</td>
<td>IDL_DLM_PATH environment variable</td>
</tr>
<tr>
<td>22U</td>
<td>IDL_FONT keyword</td>
</tr>
<tr>
<td>159U</td>
<td>writing event-handling code</td>
</tr>
<tr>
<td>198U</td>
<td>X Offset property</td>
</tr>
<tr>
<td>208U</td>
<td>X Pad property</td>
</tr>
<tr>
<td>231U</td>
<td>X Scroll draw area property</td>
</tr>
<tr>
<td>209U</td>
<td>X Scroll property</td>
</tr>
<tr>
<td>198U</td>
<td>X Size property</td>
</tr>
<tr>
<td>198U</td>
<td>Y Offset property</td>
</tr>
<tr>
<td>209U</td>
<td>Y Pad property</td>
</tr>
<tr>
<td>231U</td>
<td>Y Scroll draw area property</td>
</tr>
<tr>
<td>209U</td>
<td>Y Scroll property</td>
</tr>
<tr>
<td>199U</td>
<td>Y Size property</td>
</tr>
</tbody>
</table>

**Using IDL, Building IDL Applications, Object Graphics, Reference Guide**
<table>
<thead>
<tr>
<th>IDL_HELP_PATH environment variable</th>
<th>22U</th>
<th>Draw method 2170</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDL_PATH environment variable</td>
<td>22U</td>
<td>Erase method 2180</td>
</tr>
<tr>
<td>IDL_STARTUP environment variable</td>
<td>22U, 34U</td>
<td>GetFontNames method 2190, 2310, 3450, 4170, 4270</td>
</tr>
<tr>
<td>IDL_TREE example routine</td>
<td>248B</td>
<td>GetProperty method 2200</td>
</tr>
<tr>
<td>IDLDE windows Editor</td>
<td></td>
<td>GetTextDimensions method 2210</td>
</tr>
<tr>
<td>IDLDE windows Motif</td>
<td></td>
<td>Init method 2220</td>
</tr>
<tr>
<td>IDLDE windows Windows</td>
<td>87U</td>
<td>PickData method 2240</td>
</tr>
<tr>
<td>Graphics Macintosh</td>
<td>119U</td>
<td>Read method 2250</td>
</tr>
<tr>
<td>Graphics Motif</td>
<td>42U</td>
<td>Select method 2260</td>
</tr>
<tr>
<td>Graphics Windows</td>
<td>87U</td>
<td>SetProperty method 2270</td>
</tr>
<tr>
<td>GUIBuilder 87U</td>
<td></td>
<td>IDLGrBuffer class 2160</td>
</tr>
<tr>
<td>IDLffDICOM</td>
<td></td>
<td>IDLGrClipboard</td>
</tr>
<tr>
<td>IDLffDICOM Cleanup method</td>
<td>1550</td>
<td>class 320</td>
</tr>
<tr>
<td>IDLffDICOM GetChildren method</td>
<td>1570</td>
<td>Cleanup method 2290</td>
</tr>
<tr>
<td>IDLffDICOM GetDescription method</td>
<td>1580</td>
<td>Draw method 2290</td>
</tr>
<tr>
<td>IDLffDICOM GetElement method</td>
<td>1600</td>
<td>GetContiguousPixels method 2300</td>
</tr>
<tr>
<td>IDLffDICOM GetGroup method</td>
<td>1620</td>
<td>GetProperty method 2320</td>
</tr>
<tr>
<td>IDLffDICOM GetLength method</td>
<td>1630</td>
<td>GetTextDimensions method 2320</td>
</tr>
<tr>
<td>IDLffDICOM GetParent method</td>
<td>1640</td>
<td>Init method 2330</td>
</tr>
<tr>
<td>IDLffDICOM GetPreamble method</td>
<td>1650</td>
<td>PickData method 2350</td>
</tr>
<tr>
<td>IDLffDICOM GetReference method</td>
<td>1660</td>
<td>IDLGrClipboard object 2280</td>
</tr>
<tr>
<td>IDLffDICOM GetValue method</td>
<td>1680</td>
<td>IDLGrColorbar</td>
</tr>
<tr>
<td>IDLffDICOM GetVR method</td>
<td>1710</td>
<td>class 300, 2380</td>
</tr>
<tr>
<td>IDLffDICOM Init method</td>
<td>1720</td>
<td>Cleanup method 2380</td>
</tr>
<tr>
<td>IDLffDICOM Read method</td>
<td>1730</td>
<td>ComputeDimensions method 2390</td>
</tr>
<tr>
<td>IDLffDXF 1880</td>
<td></td>
<td>GetProperty method 2400</td>
</tr>
<tr>
<td>IDLffDXF 1880 Cleanup method</td>
<td>1890</td>
<td>Init method 2410</td>
</tr>
<tr>
<td>IDLffDXF 1880 GetChildren method</td>
<td>2190</td>
<td>SetProperty method 2450</td>
</tr>
<tr>
<td>IDLffDXF 1880 GetEntity method</td>
<td>2190</td>
<td>IDLGrColorbar object 2380</td>
</tr>
<tr>
<td>IDLffDXF 1880 Init method</td>
<td>2000</td>
<td>IDLGrContour</td>
</tr>
<tr>
<td>IDLffDXF 1880 PutEntity method</td>
<td>2000</td>
<td>class 290</td>
</tr>
<tr>
<td>IDLffDXF 1880 Read method</td>
<td>2010</td>
<td>Cleanup method 2480</td>
</tr>
<tr>
<td>IDLffDXF 1880 RemoveEntity method</td>
<td>2020</td>
<td>GetCTM method 2480</td>
</tr>
<tr>
<td>IDLffDXF 1880 Reset method</td>
<td>2020</td>
<td>GetProperty method 2490</td>
</tr>
<tr>
<td>IDLffDXF 1880 SetPalette method</td>
<td>2020</td>
<td>Init method 2500</td>
</tr>
<tr>
<td>IDLffDXF 1880 Write method</td>
<td>2030</td>
<td>SetProperty method 2550</td>
</tr>
<tr>
<td>IDLGrAxis</td>
<td></td>
<td>IDLGrContour object 2470</td>
</tr>
<tr>
<td>IDLGrAxis class 290, 2050</td>
<td></td>
<td>IDLGrFont</td>
</tr>
<tr>
<td>IDLGrAxis Cleanup method</td>
<td>2050</td>
<td>class 300, 2560</td>
</tr>
<tr>
<td>IDLGrAxis Cleanup method</td>
<td>2060</td>
<td>Cleanup method 2560</td>
</tr>
<tr>
<td>IDLGrAxis GetCTM method</td>
<td>2060</td>
<td>GetProperty method 2570</td>
</tr>
<tr>
<td>IDLGrAxis GetProperty method</td>
<td>2070</td>
<td>Init method 2570</td>
</tr>
<tr>
<td>IDLGrAxis Init method</td>
<td>2080</td>
<td>SetProperty method 2590</td>
</tr>
<tr>
<td>IDLGrAxis SetProperty method</td>
<td>2140</td>
<td>IDLGrImage</td>
</tr>
<tr>
<td>IDLGrBuffer class 320</td>
<td></td>
<td>class 290, 2600</td>
</tr>
<tr>
<td>IDLGrBuffer Cleanup method</td>
<td>2170</td>
<td>Cleanup method 2610</td>
</tr>
<tr>
<td>IDLGrBuffer Cleanup method</td>
<td>2170</td>
<td>GetCTM method 2610</td>
</tr>
<tr>
<td>IDLGrBuffer GetProperty method</td>
<td>2170</td>
<td>GetProperty method 2620</td>
</tr>
<tr>
<td>Class</td>
<td>Methods</td>
<td></td>
</tr>
<tr>
<td>---------------------</td>
<td>---------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>IDLgrLegend</td>
<td>Init method 2630, SetProperty method 2680</td>
<td></td>
</tr>
<tr>
<td>class 300</td>
<td>Cleanup method 2700, ComputeDimensions method 2700, GetProperty method 2710, Init method 2720, SetProperty method 2760</td>
<td></td>
</tr>
<tr>
<td>IDLgrLight</td>
<td>Cleanup method 2780, GetCTM method 2790, GetProperty method 2800, Init method 2810, SetProperty method 2840</td>
<td></td>
</tr>
<tr>
<td>class 290, 2780</td>
<td>Cleanup method 2780, GetCTM method 2790, GetProperty method 2800, Init method 2810, SetProperty method 2840</td>
<td></td>
</tr>
<tr>
<td>IDLgrModel</td>
<td>Cleanup method 2850, Init method 2900, Remove method 2920, Reset method 2930, Rotate method 2930, Scale method 2940, SetProperty method 2940, Translate method 2950</td>
<td></td>
</tr>
<tr>
<td>IDLgrMPEG</td>
<td>Cleanup method 2960, GetProperty method 2970, Init method 2980, Put method 3000, Save method 3000, SetProperty method 3010</td>
<td></td>
</tr>
<tr>
<td>class 320</td>
<td>Cleanup method 2960, GetProperty method 2970, Init method 2980, Put method 3000, Save method 3000, SetProperty method 3010</td>
<td></td>
</tr>
<tr>
<td>IDLgrMPEG object</td>
<td>Cleanup method 2960, GetProperty method 2970, Init method 2980, Put method 3000, Save method 3000, SetProperty method 3010</td>
<td></td>
</tr>
<tr>
<td>IDLgrPalette</td>
<td>Cleanup method 3020, GetProperty method 3040, GetRGB method 3030, Init method 3050, LoadCT method 3060, NearestColor method 3070, SetProperty method 3080</td>
<td></td>
</tr>
<tr>
<td>class 310, 3020</td>
<td>Cleanup method 3020, GetProperty method 3040, GetRGB method 3030, Init method 3050, LoadCT method 3060, NearestColor method 3070, SetProperty method 3080</td>
<td></td>
</tr>
<tr>
<td>IDLgrScene</td>
<td>Cleanup method 3540, GetProperty method 3560, Init method 3560, Remove method 3570</td>
<td></td>
</tr>
<tr>
<td>class 280, 3530</td>
<td>Cleanup method 3540, GetProperty method 3550, GetByName method 3550, Init method 3560, Remove method 3570</td>
<td></td>
</tr>
<tr>
<td>IDLgrPlot</td>
<td>Cleanup method 3150, GetCTM method 3160, GetProperty method 3170, Init method 3180, SetProperty method 3220</td>
<td></td>
</tr>
<tr>
<td>class 290, 3150</td>
<td>Cleanup method 3150, GetCTM method 3160, GetProperty method 3170, Init method 3180, SetProperty method 3220</td>
<td></td>
</tr>
<tr>
<td>IDLgrPolygon</td>
<td>Cleanup method 3240, GetCTM method 3250, GetProperty method 3260, Init method 3270, SetProperty method 3330</td>
<td></td>
</tr>
<tr>
<td>class 290, 3240</td>
<td>Cleanup method 3240, GetCTM method 3250, GetProperty method 3260, Init method 3270, SetProperty method 3330</td>
<td></td>
</tr>
<tr>
<td>IDLgrPolygon objects</td>
<td>Cleanup method 3240, GetCTM method 3250, GetProperty method 3260, Init method 3270, SetProperty method 3330</td>
<td></td>
</tr>
<tr>
<td>IDLgrPolyline</td>
<td>Cleanup method 3340, GetCTM method 3350, GetProperty method 3360, Init method 3370, SetProperty method 3410</td>
<td></td>
</tr>
<tr>
<td>class 290, 3340</td>
<td>Cleanup method 3340, GetCTM method 3350, GetProperty method 3360, Init method 3370, SetProperty method 3410</td>
<td></td>
</tr>
<tr>
<td>IDLgrScene</td>
<td>Cleanup method 3540, GetProperty method 3560, Init method 3560, Remove method 3570</td>
<td></td>
</tr>
<tr>
<td>class 280, 3530</td>
<td>Cleanup method 3540, GetProperty method 3550, GetByName method 3550, Init method 3560, Remove method 3570</td>
<td></td>
</tr>
<tr>
<td>Class/Method</td>
<td>Indexes</td>
<td></td>
</tr>
<tr>
<td>--------------</td>
<td>---------</td>
<td></td>
</tr>
<tr>
<td><code>IDLgrSurface</code></td>
<td>300, 3590, 3600, 3610, 3620, 3680</td>
<td></td>
</tr>
<tr>
<td><code>IDLgrSymbol</code></td>
<td>310, 3700, 3710, 3720, 3730</td>
<td></td>
</tr>
<tr>
<td><code>IDLgrTessellator</code></td>
<td>3760, 3770, 3780, 3780</td>
<td></td>
</tr>
<tr>
<td><code>IDLgrText</code></td>
<td>300, 3800, 3810, 3820, 3830, 3870</td>
<td></td>
</tr>
<tr>
<td><code>IDLgrView</code></td>
<td>3890, 3890, 3900, 3910, 3910, 3940, 3950</td>
<td></td>
</tr>
<tr>
<td><code>IDLgrViewgroup</code></td>
<td>3970, 3970, 3980, 3990, 3990, 4000, 4010</td>
<td></td>
</tr>
<tr>
<td><code>IDLgrVolume</code></td>
<td>300, 4020, 4020</td>
<td></td>
</tr>
<tr>
<td><code>IDLgrVRML</code></td>
<td>320, 4160, 4180, 4180, 4190, 4210, 4220, 4140</td>
<td></td>
</tr>
<tr>
<td><code>IDLgrWindow</code></td>
<td>320, 4240, 4250, 4250, 4260, 4270, 4280, 4290</td>
<td></td>
</tr>
<tr>
<td><code>IEXP</code></td>
<td>machine-specific parameter 703R</td>
<td></td>
</tr>
<tr>
<td><code>IF</code> statement</td>
<td>1028, avoiding 1318</td>
<td></td>
</tr>
<tr>
<td><code>IGAMMA</code> function</td>
<td>557R</td>
<td></td>
</tr>
<tr>
<td><code>IMAGE</code> keyword</td>
<td>1402, 152B, 153B</td>
<td></td>
</tr>
<tr>
<td><code>ID</code> keyword</td>
<td>355R, 374R, 384R</td>
<td></td>
</tr>
<tr>
<td><code>IEEE floating point</code></td>
<td>standard 256U, 152B, 153B</td>
<td></td>
</tr>
<tr>
<td><code>IDLffDICOM</code></td>
<td>Reset method 1740</td>
<td></td>
</tr>
<tr>
<td><code>IMAGE keyword</code></td>
<td>1320R, 1359R</td>
<td></td>
</tr>
<tr>
<td><code>image object</code></td>
<td>290, 2600</td>
<td></td>
</tr>
<tr>
<td><code>image objects</code></td>
<td>1140</td>
<td></td>
</tr>
<tr>
<td><code>alpha blending</code></td>
<td>1150</td>
<td></td>
</tr>
<tr>
<td><code>creating</code></td>
<td>1140</td>
<td></td>
</tr>
<tr>
<td><code>interleaving</code></td>
<td>1160</td>
<td></td>
</tr>
<tr>
<td><code>palettes</code></td>
<td>1160</td>
<td></td>
</tr>
</tbody>
</table>
saving to a file 118O
saving to an MPEG file 119O
using 115O
image processing 429U
IMAGE_CONT procedure 324U, 559R
IMAGE_COORD keyword 827R
IMAGE_DATA keyword 221O, 429O
IMAGE_INDEX keyword 880R, 923R
IMAGE_INTERP keyword 827R
IMAGE_SIZE keyword 362R
images 260O
annotating 196U
bi-level 1118R
color channel 265O
combining with 3D graphics 347U
copying areas 118R
defining region of interest 407R
displaying with intensity scaling 1164R
displaying, Macintosh 130U
dissolve effect 436R
image manipulation compound widgets 276B
JPEG 907R
magnified 1382R, 1384R
monochrome 145R
MPEG files 757R, 758R, 760R, 762R
orientation 382U
overlaying with contour plots 324U
overview 380U
position in display 382U
processing 380U
profiling 844R, 848R
raster 380U
reading from display 385U, 395U, 1161R
region labeling 596R
Roberts edge enhancement 969R
rotating 972R
routines 380U
scaling 384U
searching for objects 986R
sharing data 267O
size of display 383U
smoothing 1043R
Sobel edge enhancement 1045R
thinning 1118R
transfer direction 48R
true-color 1162R
warping 204R
warping to maps 715R, 718R
with surface and contour plots 1015R
zooming 390R
IMAGINARY function 560R
imaginary part of complex numbers 560R
implicit self argument 262B
INCREASE keyword 987R, 990R
incrementing array elements 548R
INDEPENDENT keyword 645R, 650R
INDEX keyword 199O, 202O, 705R
index tab (online help system) 248U
INDEX_COLOR keyword 126R
Indexed color model 580
INDEXED_COLOR keyword 613R, 626R, 650R, 666R
INDGEN function 561R
infinite impulse response filters 414U
Infinity norm 460U, 769R
infinity, undefined result 152B
INFORMATION keyword 424R
INFORMATIONAL keyword 745R
Informational Routines 124B
inheritance 253B
object 251B
INIT keyword 227R
Initial Value droplist property 224U
Initial Value listbox property 226U
Initial Value text property 216U
initialization of objects 330
INITIALSIZE keyword 789R
inner products (computing) 430U
INP, see Obsolete Routines
INPUT keyword 547R, 1135R
input/output
associated 202B
associated variables 204R
bitmap files 901R
BMP files 1320R
CDF files 1320R
...
Index

Index-34

Index

Index-34

Index

closing files 273R
emptying buffers 449R, 491R
end of file mark 1200R
error handling 146B
errors 780R
explicit format 159B, 169B
format codes 173B
format reversion 172B
formatted 158B, 841R
free_format 159B, 165B
GIF files 904R, 1322R
HDF 1388R
Interfile files 906R
JPEG files 907R, 1324R
magnetic tape 224B
netCDF 1389R
NRIF files 1327R
opening files 783R
PGM files 913R, 1331R
PICT files 910R, 1328R
portable 197B
PPM files 913R, 1331R
reading
    ASCII files 898R
      formatted data 895R
      formatted data from a string 931R
      from a prompt 896R
      from tape unit 1114R
      unformatted binary data 933R
    SRF files 916R, 1333R
    TIFF files 921R, 1337R
    unformatted 158B, 191B
      portable 197B
    string variables 192B
    UNIX FORTRAN unformatted data files 206B
    updating records (REWRITE keyword) 1344R
    wave files 926R, 1342R
writing
    to tape unit 1115R
      unformatted binary data 1344R
    X1I Bitmaps 928R
    XDR 198B
    XWD files 930R
INPUT_FOCUS keyword 1238R
INSERT_COLUMNS keyword 1238R
INSERT_OVERSTRIKE_TOGGLE keyword 403R
INSERT_ROWS keyword 1238R
instance
    object 250B

Instancing 149O
    instancing 137O, 149O
INSTANCING keyword 613R, 626R, 650R, 666R
    INT keyword 705R
    INT_2D function 450U, 453U, 563R
    INT_3D function 450U, 453U, 566R
    INT_TABULATED function 453U, 568R
    INTARR function 570R
    integer 484R
    arrays 561R, 570R
    constants 257U, 15B
    conversions, errors in 154B
    data type 12B
    data type, converting to 484R
    INTEGER keyword 369R
integration 449U
    bivariate functions 450U
    INT_2D 563R
    INT_3D 566R
    INT_TABULATED 568R
    QROM B 862R
    QROM O 864R
    QSIM P 866R
    RK4 967R
    tabulated functions 568R
    trivariate functions 451U
    univariate functions 862R, 864R, 866R
    INTENSITY keyword 282O, 505R
interactive graphics 137O
    instancing 137O
    Interactive Support 242U
    Interactive Surface Example 110O
    INTERCHANGES keyword 696R
    Interfile files
      reading 906R
      standard file format I/O routines 228B
    INTERIOR keyword 376O
    INTERLACED keyword 299O
    INTERLEAVE keyword 266O
    interleaving (image objects) 1140
    INTERPOL function 445U, 571R
    INTERPOLATE function 445U, 573R
    INTERPOLATE keyword 409O, 993R, 1193R
    interpolating
      function 445U
      interpolation 444U, 573R
      bilinear 217R, 935R
      cubic convolution 573R, 821R
cubic spline 1054R, 1058R, 1059R
INTERPOL 571R
irregularly-gridded data 1134R
irregularly-sampled data over earth 1131R
KRIG2D 586R
linear 571R
MIN_CURVE_SURF 748R
of irregularly-gridded data 586R, 748R
POLAR_SURFACE 817R
quintic 1136R
spherical 1052R
SPL_INIT 1054R
SPL_INTERP 1056R
of irregularly-gridded data 1131R, 1134R
POLAR_SURFACE 817R
quintic 1136R
spherical 1052R
SPL_INIT 1054R
SPL_INTERP 1056R
interpolation of voxel values 127O
interpret symbols, DCL
defining 996R
deleting 411R
returning values 518R
interrupting program execution 30U
INTERVAL keyword 811R, 1059R, 1202R
invalid widget ID's 1268R
inverse
cosine 190R
determinant of a complex array or matrix 694R
sine 203R
subspace iteration 446R
tangent 206R
INVERSE keyword 475R, 694R, 1348R
INVERT function 460U, 576R
IOCTL function 578R
ERROR keyword 745R
IRND machine-specific parameter 703R
IRREGULAR keyword 304R
irregularly-gridded data 1131R, 1134R
ISA keyword 185O
IsContained method 186O
ISHFT function 581R
IS Latin 1 encoding 68R
ISOLATIN1 keyword 127R
isosurfaces, displaying 351U, 1010R
ISOTROPIC keyword 360U, 304R, 729R, 800R
ISSUE_ERROR keyword 1146R
IT machine-specific parameter 703R
ITALIC keyword 127R
ITEM_COLOR keyword 274O
ITEMLineStyle keyword 274O
ITEM_NAME keyword 274O
ITEM_OBJECT keyword 274O
ITEM_THICK keyword 274O
ITEM_TYPE keyword 275O
ITER keyword 421R, 602R, 678R, 839R
iterative
biconjugategradient 601R
Gaussian quadrature 563R, 566R
improvement of a solution 697R
ITOL keyword 601R
J
JFIF, see JPEG
JMAX keyword 683R, 863R, 865R, 867R
JOIN, see Obsolete Routines
JOURNAL procedure 35U, 582R
journaling 35U
JPEG files
reading 907R
standard file format I/O routines 228B
writing 1324R
JULDAY function 583R
Julian date
converting from calendar 583R
converting to calendar 242R
JUST_REG keyword 1364R
K
K keyword 863R, 865R
Kaiser filter 414U
KBRD_FOCUS_EVENTS keyword 1209R, 1239R, 1274R, 1301R, 1311R
KEEP keyword 1382R
KEEP_PIXMAPS keyword 1359R
KENDALL keyword 882R
KENDALL's tau rank correlation 882R
kernel, convolving an array with 311R
KEY_ID keyword 896R, 934R
KEY_MATCH keyword 896R, 934R
KEY_VALUE keyword 896R, 934R
keyboard
defining keys 399R
focus events 311B, 1209R, 1239R, 1274R, 1301R, 1311R
numeric keypads 1001R
returning characters from 514R
shortcuts
IDLDE for Motif 55U
IDLDE for Windows 100U
KEYED keyword 789R
KEYS keyword 538R
keys, defining for different keyboards 1000R
KEYWORD_SET function 124B, 125B, 585R
keywords:
  arguments, checking existence of 198R
  described 181O, 185R
  graphics 99R
  identifying as set 585R
  inheritance 114B
  meaning of slash character 181O, 185R
  parameters 111B
  passing 113B
  searching 781R
  setting 111B, 181O, 185R
KEYWORDS Keyword 608R
KILL ANYWAY keyword 345R
killing widgets 300B
KMEANS, see Obsolete Routines
KRIG2D function 445U, 586R
kriging 445U, 586R
KRUSKAL_WALLIS, see Obsolete Routines
Kruskal-Wallis H-Test 591R
kurtosis 590R, 755R
KURTOSIS function 590R
KW_TEST function 448U, 476U, 591R
KX keyword 1002R

L

L64 keyword 561R
L64_VALUE keyword 247R
L64NDFGEN function 593R
L64SWAP keyword 236R
Label button property 214U
LABEL0 keyword 351R, 585R, 711R, 727R
label widgets 274B, 1280R
  attributes, setting 221U
  setting properties 221U
  using 165U
LABEL_DATE function 594R
LABEL_LEFT keyword 355R
LABEL_REGION function 596R
LABEL_TOP keyword 355R
labeling regions 596R
LADFIT function 438U, 598R
lagged
  autocorrelation 187R
  cross correlation 241R
Laguerre's method 502R
LAMBERT keyword 724R
Lambert's conformal conic map projection 371U, 724R
Lambert's equal-area map projection 365U, 724R
LANDSCAPE keyword 3480, 127R
landscape orientation 159R
  for IDL plots (LANDSCAPE keyword) 127R
laser printers 157R
LAST MESSAGE keyword 538R
LAT0 keyword 719R
LAT1 keyword 719R
LATALIGN keyword 711R, 727R
LATDEL keyword 363U, 712R, 727R
LATLAB keyword 712R, 727R
LATLON, see Obsolete Routines
LATMAX keyword 716R
LATMIN keyword 716R
LATNAMES keyword 712R
LATS keyword 712R
LAYER field 1920
LAYER keyword 1920, 1990
Layout base property 205U
LE operator 277U, 36B
leakage (in frequency plots) 405U
least absolute deviation 438U, 598R
LEEFILT function 600R
legend object 300
legend objects 970
LEGEND_PROPERTIES Keyword 662R
LEGENDRE Keyword 1099R
LEGO keyword 1091R
LEGO, see Obsolete Routines
LEN keyword 488R
length of strings 1080R
LEVELS keyword 3280, 304R
license manager 414R
lifecycle
  methods 255B
  routines 255B
LIGHT keyword 127R, 994R
light object 290, 2780
light objects 1040, 1070
Index

creating 1080

types of lights 1080

using 1080

light source 2780

changing parameters of 350U

shading 349U, 994R

lighting 1260, 1490

LIGHTING keyword 2910

LIGHTING MODEL keyword 4100

LIMIT keyword 360U, 729R

LIMSER keyword 296R

LINBCG function 466U, 601R

LINDGEN function 603R

line

drawing

method for contours 300R

PLOTS procedure 809R

editing 24U

enabling and disabling 42R

interval 46R

styles 103R

LINE keyword 650R

LINE_FILL keyword 827R

linear

algebra, See linear algebra 431U

correlation 431U

interpolation 573R

linear-log plots 801R

regression 438U, 947R

systems 453U

overdetermined 454U

underdetermined 456U

linear algebra

CHOLDC 267R

CHOLSOL 268R

COND 290R

CRAMER 319R

DETERM 417R

EIGENVEC 446R

ELMHES 448R

GS_ITER 525R

HQR 551R

INVERT 576R

LINBCG 601R

LU_COMPLEX 694R

LUDC 696R

LUMPROVE 697R

LUSOL 699R

NORM 769R

SVDC 1096R

SVSOL 1102R

TRIQL 1140R

TRIRED 1142R

TRISOL 1143R

LINEAR keyword 1128R

LINETYPE field 192O


LINETYPE system variable field 49R

linestyles, table of 103R

LINFIT function 438U, 604R

linked lists 242B

using pointers to create 242B

LINKIMAGE 1404

LINKIMAGE procedure 245R, 606R

LIST keyword 789R

list widgets 274B, 1285R

determining

currently selected element

LIST_SELECT keyword 1274R

topmost element

LIST_TOP keyword 1274R

double-clicks 1290R

events returned by 1290R

number 1274R

selecting multiple items 1274R, 1286R

setting 1243R

LIST_NUM_VISIBLE keyword 1274R

LIST_NUMBER keyword 1274R

LIST_SELECT keyword 1274R

LIST_TOP keyword 1274R

listbox widgets

attributes 226U

events 227U

initial value 226U

multiple selections, allowing 226U

selection events 227U

setting height 226U

setting properties 226U

using 165U

width 227U

LISTREP, see Obsolete Routines

LISTWISE, see Obsolete Routines

little endian byte ordering 1104R

LIVE_CONTOUR procedure 612R

LIVE_CONTROL procedure 619R

LIVE_DESTROY procedure 621R

LIVE_EXPORT procedure 623R

Using IDL, Building IDL Applications, Object Graphics, Reference Guide Index  Index L … L
LIVE_IMAGE procedure 625R
LIVE_INFO procedure 630R
LIVE_LINE procedure 640R
LIVE_LOAD procedure 644R
LIVE_PLOT procedure 649R
LIVE_PRINT procedure 655R
LIVE_RECT procedure 656R
LIVE_STYLE procedure 659R
LIVE_SURFACE procedure 665R
LIVE_TEXT procedure 671R
LJ device
   - color tables for 674R
   - driver 151R
LJLCT procedure 674R
LL_ARC_DISTANCE function 675R
LM_LICENSE_FILE environment variable 23U
LMFIT function 677R
LMGR function 680R
LMHOSTID keyword 681R
LN03, see Obsolete Routines
LNGAMMA function 682R
LNP_TEST function 448U, 683R
LOAD_FILE keyword 197R
LoadCT method 306O
LOADCT procedure 389U, 685R
loading color tables 1159R
location of graphics 420
location of text 81O
location of widgets 306B
LOCATIONS keyword 385O
LOG keyword 211O
logarithm
   - base 10 192R
   - natural 191R
   - of the gamma function 682R
logarithmic
   - scaling 305U
logarithmic plots 76O
logging an IDL session 582R
logical names (VMS)
   - defining 998R
   - deleting 412R
   - searching tables 1145R
logical names, see environment variables
logical unit numbers 162B
   - ID system variable field 47R
   - allocating 516R
   - for journal file 43R
   - freeing 493R
   - FSTAT function 494R
   - getting 785R
   - obtaining status information 494R
   - returning information about 538R
   - setting file position pointer 813R
LOGIN.COM file 20U
LOGISTIC keyword 283R
LOGSQUARE keyword 283R
Lomb Normalized Periodogram 683R
LON0 keyword 719R
LON1 keyword 719R
LON64ARR function 686R
LONALIGN keyword 712R, 727R
LONARR function 687R
LONDIES keyword 363U, 712R, 727R
LONG function 688R
long integer data type 12B
longjmp, C language 255R
longword
   - arrays 603R, 687R, 1171R
   - data type, converting to 688R
   - unsigned arrays 1170R
LOWER keyword 352U, 1011R
lower margin, setting 53R
LOWERONLY keyword 1091R
lowercase, converting strings to 1081R
lowpass filters 414U
LPF_BAND keyword 987R, 990R
LSODE function 690R
LSODE keyword 716R
LSWAP keyword 236R
LT_operator 278U, 36B
LU decomposition 694R, 696R, 699R
LU_COMPLEX function 460U, 694R
LUBKSB, see Obsolete Routines
LUDC procedure 460U, 696R
Index

LUDCMP, see Obsolete Routines
luminance 329R
LUMPROVE function 460U, 697R
LUN's (logical unit numbers) 162B, 493R
Macintosh differences 136U
LUSOL function 460U, 699R

M
M_CORRELATE function 435U, 476U, 701R
MACCREATOR keyword 787R, 1048R
MACHAR function 428U, 703R
MACHEP machine-specific parameter 703R
machine-specific parameters 703R
Macintosh
backing store 130U
color palette 130U
compiling functions and procedures 135U
display device (MAC) 112R, 154R
displaying images 130U
errors 136U
IDLDE 118U
changing working directory 135U
LUN (logical unit number) differences 136U
mouse differences 134U
optimizing performance with memory 133U
path specification 44R
pixel depth 130U
positioning file pointers 136U
reading and writing files 136U
saving/restoring files 136U
specifying file search path 132U, 133U
using batch files 132U
MACTYPE keyword 787R
magnetic tape 224B
magnifying arrays 935R
magnitude
of a complex number 189R
of spectra 404U
magnitude-based ranks 892R
main
directory
IDLDE for Macintosh 132U
IDLDE for Motif 66U
IDLDE for Windows 107U
programs 29U
window, setting preferences
IDLDE for Macintosh 128U
IDLDE for Motif 58U
IDLDE for Windows 59U, 104U
MAJOR keyword 211O, 243O
MAKE_ARRAY function 705R
Maker Interchange Format 250U
MAKE_TREE, see Obsolete Routines
MANAGE_STYLE keyword 613R, 626R, 650R, 666R
MANAGED keyword 1239, 1275R
managing the state of a widget application 312B
MANN_WHITNEY, see Obsolete Routines
Mann-Whitney U-Test 977R
MAP keyword 355R, 1209R, 1239R
map projections 358U, 723R
Aitoff 365U, 724R
Alber's equal area conic 724R
Alber's equal-area conic 372U
azimuthal 362U
azimuthal equidistant 365U, 724R
boundaries, specifying 360U
cylindrical 368U
cylindrical equidistant 371U, 724R
drawing boundaries over 708R
drawing continent boundaries 359U, 726R
drawing parallels and meridians 711R
gnomonic(cenal, gnomic) 363U, 724R
Hammer-Aitoff 366U, 724R
high-resolution outlines 377U
Lambert's conformal conic 371U, 724R
Lambert's equal area 365U, 724R
Mercator 369U, 724R
Miller 724R
Miller cylindrical 371U
Mollweide 374U, 725R
orthographic 362U, 725R
Robinson 373U
satellite 366U, 725R
sinusoidal 373U, 725R
stereographic 362U, 725R
Transverse Mercator (UTM) 369U, 725R
warping images to maps 375U, 715R, 718R
MAP_ALL keyword 281R
MAP_CONTINENTS procedure 358U, 360U, 708R
MAP_GRID procedure 358U, 360U, 711R
MAP_IMAGE function 358U, 715R
MAP_PATCH function 718R
MAP_PROJ_INFO routine 721R
MAP_SET procedure 358U, 723R
mapping widgets 1209R
MARGIN system variable field 53R
Index M ... M

Using IDL, Building IDL Applications, Object Graphics, Reference Guide Index

margins, setting 53R
marquee selector 227R
MASK keyword 438O
MATCH_PREVIOUS keyword 402R
math errors 151B
mathematical operators 270U, 28B
mathematics 426U
matrices, multiplying 272U, 31B
MATRIX keyword 1186R
matrix operators
  CHOLDC 267R
  CHOLSOL 268R
  COND 290R
  CRAMER 319R
  DETERM 417R
  EIGENVEC 446R
  ELMHES 448R
  GS_ITER 525R
  HQR 551R
  INVERT 576R
  LU_COMPLEX 694R
  LUDC 696R
  LUMPROVE 697R
  LUSOL 699R
  NORM 769R
  SVD 1096R
  SVSOL 1102R
  TRIQL 1140R
  TRIRED 1142R
  TRISOL 1143R
  See also sparse arrays
MAX function 731R
MAX keyword 238R, 390R, 547R, 747R
MAX_ARGS keyword 608R
MAX_VALUE keyword 309U, 2530, 3200, 3650,
  304R, 716R, 719R, 792R, 800R, 1005R, 1092R,
  1135R
MAXEXP machine-specific parameter 703R
MAXIMIZE keyword 296R
maximum intensity projection 126O
MAXIMUM keyword 378R, 1293R
maximum operator 272U, 30B
maximum size of drawable 136O, 424O
maximum value
  for slider widgets 1293R
  of an array 731R
Maximum Value slider property 222U
MAXIMUM_INTENSITY keyword 1193R
MAXV keyword 544R
MAXVAL keyword 461R, 878R, 913R
MBAR keyword 390R, 1209R
MD_TEST function 448U, 733R
MDC keyword 733R
MDEV keyword 756R
mean
  absolute deviation 736R
  MOMENT function 755R
  of distribution 591R
MEAN function 735R
MEANABSDEV function 736R
median
  Median Delta Test 733R
  MOMENT function 755R
  smoothing 738R
MEDIAN function 423U, 738R
MEDIAN keyword 736R
MEDIUM keyword 127R
memory
  allocation under VMS 23U
  conserving by using temporary variables 1117R
  dynamic memory in use 539R
  graphics system use 269B, 24O
  optimizing performance
    IDLDE for Macintosh 133U
    IDLDE for Windows 58U, 103U
MEMORY keyword 539R
menu bars 1206R, 1209R
Menu Editor
  opening 92U
Menu Editor, using 171U
MENUS, see Obsolete Routines
menus 295B
  editing in IDL GUIBuilder 171U
  menu bars 1206R, 1209R
  menu items
    IDLDE for Macintosh 120U
    IDLDE for Motif 43U
    IDLDE for Windows 88U
  online help 242U
  pulldown 298B, 1224R
  system, using 207U
Mercator map projection 369U, 724R
meridians, drawing 359U, 711R, 727R
mesh plots 1090R
MESH_OBJC procedure 740R
MESH_NAME keyword 1342R
Index

MESHNAMES keyword 927R
message dialogs 277B, 423R
MESSAGE keyword 227R
MESSAGE procedure 147B, 745R
message widgets 274B
message-of-the-Day file
IDLDE for Macintosh 134U
IDLDE for Windows 110U
MESSAGES keyword 539R
messages, suppressing informational 45R
METHOD keyword 557R
method overriding 263B
methods 261B
defining routines 261B
Windows 3.11 263B
invocation 258B
object 250B
Microsoft Windows
mouse differences 113U
Microsoft Windows display device (WIN) 112R, 170R
MIDEXP keyword 865R
MIDINF keyword 865R
MIDPNT keyword 865R
MIDSQL keyword 865R
MIDSQU keyword 865R
MIF 250U
Miller cylindrical map projection 371U
MILLER keyword 724R
Miller map projection 724R
MIN function 747R
MIN keyword 238R, 391R, 547R, 731R
MIN_ARGS keyword 608R
MIN_CURVE_SURF function 329U, 445U, 300R, 748R
MINEXP machine-specific parameter 703R
minimization 462U, 420R, 838R
downhill-simplex method 463U
See also optimization
Minimize/Maximize base property 206U
minimum and maximum values of plot data 950
minimum curvature surface 445U, 748R
MINIMUM keyword 378R, 1293R
minimum operator 272U, 30B
minimum value
for slider widgets (MINIMUM keyword) 1293R
of an array 747R
Minimum Value slider property 222U
MINOR keyword 2110, 243O
MINOR system variable field 53R
MINUS.ONE keyword 292R
MINV keyword 544R
MIPS keyword 1001R
MIPSEB_DBLFIXUP, see Obsolete Routines
missing data 309U
in CONTOUR plots 304R
in irregular grids 1129R, 1136R
in map projections 716R
in plots 792R, 800R, 1005R, 1092R
in reconstructed images 941R
in rotated images 971R
in velocity fields 1185R
in warped images 821R
MISSING_VALUE keyword 899R
MK_HTML_HELP procedure 251U, 751R
MLINSTYLE keyword 709R, 728R
MLINETHICK keyword 709R, 728R
Modal base property 206U
modal dialogs, creating 206U
MODE keyword 941R
model object 280, 285O
model objects 37O
MODIFYCT procedure 391U, 754R
modules
compiled 540R
dynamically loaded 538R
modulo operator 271U, 30B
MOLLWEIDE keyword 725R
Mollweide map projection 374U, 725R
MOMENT function 755R
MONTHS keyword 2110, 595R
MORE keyword 785R
morphology
dilation operator 432R
erosion operator 452R
Mosaic 251U, 751R
MOTION_EVENTS keyword 1255R
mouse
double-clicks 1290R
emulating three-button 113U, 134U
Macintosh differences 134U
reading position of 313U, 894R
reading position with the CURSOR procedure 332R
returning events from draw widgets 1255R

Using IDL, Building IDL Applications, Object Graphics, Reference Guide

Index M ... M
Index N ... N

N

N keyword 806R
N_CLUSTERS keyword 274R, 275R
N_COLORS keyword 2230, 2340, 3040, 4200
N_COLORS system variable field 46R
N_DIMENSIONS keyword 1023R
N_ELEMENTS function 112B, 124B, 125B, 764R
N_ELEMENTS keyword 1023R
N_ITERATIONS keyword 274R
N_LEVELS keyword 2530
N_PARAMS function 112B, 124B, 126B, 765R
N_TAGS function 766R
N0 keyword 884R
N1 keyword 884R
Name common property 196U
NAME system variable field 46R
named structures 262U, 42B
variables 184R, 185R
named variables 181O
names of structure tags 1111R
of variables 265U, 21B
NaN (not-a-number) 152B
NaN values 1402
NARROW keyword 127R
native format (floating-point values) 235R
natural exponential function 460R
natural logarithm 191R
Navigate menu (online help) 246U
Navigation Dialog (online help) 248U
NCALLS keyword 194R
NCAR binary encoding 127R, 128R
NCAR keyword 127R
NCAR Raster Interchange Format files, writing 1327R
NCOLS keyword 919R

Index N ... N
Index

NCOPIES keyword 3490
NE operator 278U, 368
    object references 259B
    pointers 238B
near and far clipping planes 470
NearestColor method 307O
NEGATIVE keyword 509R
NEGEP machine-specific parameter 703R
nesting of procedures and functions 537R, 540R
netCDF 1389R
Netscape 251U, 751R
new page 451R
new page (printing) 141O
NEW_AXES keyword 646R
NEW_WINDOW keyword 1383R
NewDocument method 350O
NewPage method 350O
NEWTON function 462U, 767R
Newton's method 461U, 568R, 767R
NEXT_LINE keyword 404R
NFRAMES keyword 348R
NGRD machine-specific parameter 703R
NLAT keyword 1053R
NLATIN keyword 853R
NO_ALIAS keyword 998R
NO_BALANCE keyword 448R
NO_BLOCK keyword 1358R, 1365R
NO_DIVIDE keyword 1186R
NO_HEADER keyword 1302R
NO_KILL keyword 342R
NO_LINE keyword 1240R, 1311R
NO_POLISH keyword 502R
NO_RELEASE keyword 355R, 374R, 1224R
NO_STATUS keyword 614R, 627R, 651R, 667R
NO_TOOLBAR keyword 614R, 627R, 651R, 667R
NOAUTOMODE keyword 787R
NOBORDER keyword 728R
NOCLIP keyword 103R
NOCLIP system variable field 51R
NOCLISYM keyword 1049R
NODATA keyword 104R
NOECHO keyword 402R
NOEDIT keyword 369R
NOERASE keyword 104R, 728R
NOERASE system variable field 51R
NOFILL keyword 407R
noise, filtering 738R
NOLATIN keyword 853R
NOLOGNAM keyword 1049R
NOMARK keyword 844R
NOMESHDEF keyword 1342R
NONAME keyword 745R
NONE keyword 789R
NONEXCLUSIVE keyword 355R, 1211R
noninteractive mode 32U, 34U
nonlinear equations 461U
    BROYDEN 231R
    CONSTRAINTS_MIN 294R
    FX_ROOT 500R
    FZ_ROOTS 502R
    NEWTON 767R
nonparametric tests 447U
    LNP_TEST 683R
    MD_TEST 733R
    R_TEST 884R
    RS_TEST 977R
    S_TEST 980R
    XSQ_TEST 1375R
non-printable characters 18B
NOPREFIX keyword 745R
NOPRINT keyword 746R
NOREGION keyword 407R
NORM function 460U, 769R
NORM keyword 1202R
norm Coord.pro (example file) 53O
normal
    coordinates 286U
    converting to other types 309R
    distribution (Gaussian) 506R, 507R
    random deviates 891R
Normal Computations 1480
    normal computations 92O
    NORMAL keyword 104R, 309R, 333R, 833R
    NORMALIZED keyword 200R
    normally-distributed random numbers 886R
    NORMALS keyword 329O
    NOSHEDL keyword 1048R
    NOSHOW keyword 1374R
    NOSTDIO keyword 787R

Using IDL, Building IDL Applications, Object Graphics, Reference Guide Index
and TRANSFER_COUNT 1344R
NOT operator 274U, 32B
notch filter 419U
NOTEX keyword 211O
NOTIFY keyword 1049R
NOTIFY_REALIZE keyword 1211R, 1224R, 1240R,
NOTIFY_RESET keyword 1048R
NOWAIT keyword 333R

Number of array elements 764R
numbers, random 886R, 890R
numeric keypads 1001R
numerical integration 449U, 866R
Numerical Recipes 426U
NVariables keyword 797R
NVECS keyword 488R, 1182R
NX keyword 588R, 1130R, 1136R
NY keyword 588R, 1130R, 1136R
Nyquist frequency 409U

NSteps keyword 488R, 1182R
NSUM keyword 3200, 792R, 801R
NSUM system variable field 51R
NTERMS keyword 511R
NTOH keyword 236R
NTOHS keyword 236R
Null display device (NULL) 154R
NUM_KEYPAD keyword 1001R
NUM_RECORDS keyword 899R

Object

class 250B
class structures 251B
capsulation 250B
heap variables 250B
inheritance 251B, 253B
instances 250B
lifecycle 255B
method routines 261B
naming conventions 270
persistence 251B
polymorphism 251B
Object Graphics 22O

definitions 22O
files, writing 1327R
standard file format I/O routines 228B
NROWS keyword 919R

NRF
<table>
<thead>
<tr>
<th>Object Graphics 1480</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expose Events 1480</td>
</tr>
<tr>
<td>Object Graphics 1490</td>
</tr>
<tr>
<td>object heap variables 254B</td>
</tr>
<tr>
<td>object oriented programming 250B</td>
</tr>
<tr>
<td>objects</td>
</tr>
<tr>
<td>creating 774R</td>
</tr>
<tr>
<td>arrays 778R</td>
</tr>
<tr>
<td>destroying 257B, 772R</td>
</tr>
<tr>
<td>determining</td>
</tr>
<tr>
<td>class names 771R</td>
</tr>
<tr>
<td>subclasses 773R</td>
</tr>
<tr>
<td>heap variables 230B, 254B</td>
</tr>
<tr>
<td>Object Graphics 268B, 230</td>
</tr>
<tr>
<td>clipboard support</td>
</tr>
<tr>
<td>Macintosh 120U</td>
</tr>
<tr>
<td>Windows 88U</td>
</tr>
<tr>
<td>font use 66R</td>
</tr>
<tr>
<td>object-oriented</td>
</tr>
<tr>
<td>graphics 268B, 240</td>
</tr>
<tr>
<td>references for heap variables 230B</td>
</tr>
<tr>
<td>testing existence 776R</td>
</tr>
<tr>
<td>OBJECTS keyword 539R</td>
</tr>
<tr>
<td>OBLIQUE keyword 337U, 128R, 1109R</td>
</tr>
<tr>
<td>obsolete routines and system variables 1391R</td>
</tr>
<tr>
<td>Obtaining Traceback Information 148B</td>
</tr>
<tr>
<td>OCCLUDED keyword 1202R</td>
</tr>
<tr>
<td>octal 14B</td>
</tr>
<tr>
<td>Oetli, Thomas 377U</td>
</tr>
<tr>
<td>OFAC keyword 683R</td>
</tr>
<tr>
<td>OFFSET keyword 362R</td>
</tr>
<tr>
<td>OMARGIN system variable field 53R</td>
</tr>
<tr>
<td>OMAX keyword 547R</td>
</tr>
<tr>
<td>OMIN keyword 547R</td>
</tr>
<tr>
<td>ON_ERROR procedure 142B, 145B, 745R, 779R</td>
</tr>
<tr>
<td>ON_IOERROR procedure 780R</td>
</tr>
<tr>
<td>OnButton draw area event 232U</td>
</tr>
<tr>
<td>OnButton Press event property 215U</td>
</tr>
<tr>
<td>ONCE keyword 230R</td>
</tr>
<tr>
<td>OnCellSelect table event 237U</td>
</tr>
<tr>
<td>OnChangeValue slider event 223U</td>
</tr>
<tr>
<td>OnColWidth table event 238U</td>
</tr>
<tr>
<td>OnDelete table event 238U</td>
</tr>
<tr>
<td>OnDelete text event 218U</td>
</tr>
<tr>
<td>OnDestroy property 200U</td>
</tr>
<tr>
<td>one-tailed hypothesis tests 447U</td>
</tr>
<tr>
<td>OnExpose draw area event 233U</td>
</tr>
<tr>
<td>OnFocus event property 210U</td>
</tr>
<tr>
<td>OnFocus table event 238U</td>
</tr>
<tr>
<td>OnFocus text event 218U</td>
</tr>
<tr>
<td>ONGLASS keyword 3850</td>
</tr>
<tr>
<td>OnInsertCh text event 218U</td>
</tr>
<tr>
<td>OnInsertChar table event 239U</td>
</tr>
<tr>
<td>OnInsertString table event 239U</td>
</tr>
<tr>
<td>OnInsertString text event 219U</td>
</tr>
<tr>
<td>OnInvalidData table event 239U</td>
</tr>
<tr>
<td>OnKillRequest event property 210U</td>
</tr>
<tr>
<td>online help 242U, 440R, 751R</td>
</tr>
<tr>
<td>&quot;Back&quot; button 247U</td>
</tr>
<tr>
<td>annotate 244U</td>
</tr>
<tr>
<td>bookmark menu 244U</td>
</tr>
<tr>
<td>browse buttons 247U</td>
</tr>
<tr>
<td>button bar 247U</td>
</tr>
<tr>
<td>calling from programs 781R</td>
</tr>
<tr>
<td>contents button 247U</td>
</tr>
<tr>
<td>contents tab 248U</td>
</tr>
<tr>
<td>copying a topic 243U</td>
</tr>
<tr>
<td>edit menu 243U</td>
</tr>
<tr>
<td>extending 249U</td>
</tr>
<tr>
<td>File menu 243U</td>
</tr>
<tr>
<td>find tab 249U</td>
</tr>
<tr>
<td>help menu 246U</td>
</tr>
<tr>
<td>index tab 248U</td>
</tr>
<tr>
<td>menus 242U</td>
</tr>
<tr>
<td>Navigate menu 246U</td>
</tr>
<tr>
<td>Navigation dialog 248U</td>
</tr>
<tr>
<td>printing 243U</td>
</tr>
<tr>
<td>view menu 246U</td>
</tr>
<tr>
<td>ONLINE_HELP procedure 250U, 781R</td>
</tr>
<tr>
<td>ONLY_8BIT, see Obsolete Routines</td>
</tr>
<tr>
<td>OnMotion draw area event 233U</td>
</tr>
<tr>
<td>OnRealize event property 200U</td>
</tr>
<tr>
<td>OnSelectValue droplist event 225U</td>
</tr>
<tr>
<td>OnSelectValue listbox event 227U</td>
</tr>
<tr>
<td>OnSizeChange event property 210U</td>
</tr>
<tr>
<td>OnTextSelect table event 240U</td>
</tr>
<tr>
<td>OnTextSelect text event 219U</td>
</tr>
<tr>
<td>on-the-glass text 82O</td>
</tr>
<tr>
<td>OnTimer event property 200U</td>
</tr>
<tr>
<td>OnTracking event property 200U</td>
</tr>
<tr>
<td>OnViewportMoved draw area event 233U</td>
</tr>
<tr>
<td>opacities 1192R</td>
</tr>
<tr>
<td>OPACITY keyword 4030</td>
</tr>
<tr>
<td>opacity table 124O</td>
</tr>
<tr>
<td>OPACITY_TABLE0 keyword 4100</td>
</tr>
<tr>
<td>OPACITY_TABLE1 keyword 4100</td>
</tr>
<tr>
<td>OPAQUE keyword 851R</td>
</tr>
<tr>
<td>OPEN procedures 783R</td>
</tr>
<tr>
<td>OPEN_FUNC keyword</td>
</tr>
<tr>
<td>-------------------</td>
</tr>
<tr>
<td>opening files</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>getting information on open files</td>
</tr>
<tr>
<td>opening operation, in image processing</td>
</tr>
<tr>
<td>commands, issuing</td>
</tr>
<tr>
<td>current version in use</td>
</tr>
<tr>
<td>operations on objects</td>
</tr>
<tr>
<td>operations on pointers</td>
</tr>
<tr>
<td>operators</td>
</tr>
<tr>
<td>addition</td>
</tr>
<tr>
<td>AND</td>
</tr>
<tr>
<td>array concatenation</td>
</tr>
<tr>
<td>assignment</td>
</tr>
<tr>
<td>Boolean</td>
</tr>
<tr>
<td>division</td>
</tr>
<tr>
<td>EQ</td>
</tr>
<tr>
<td>exponentiation</td>
</tr>
<tr>
<td>GE</td>
</tr>
<tr>
<td>GT</td>
</tr>
<tr>
<td>LE</td>
</tr>
<tr>
<td>LT</td>
</tr>
<tr>
<td>mathematical</td>
</tr>
<tr>
<td>matrix multiplication</td>
</tr>
<tr>
<td>maximum</td>
</tr>
<tr>
<td>minimum</td>
</tr>
<tr>
<td>modulo</td>
</tr>
<tr>
<td>multiplication</td>
</tr>
<tr>
<td>NE</td>
</tr>
<tr>
<td>NOT</td>
</tr>
<tr>
<td>OR</td>
</tr>
<tr>
<td>parentheses</td>
</tr>
<tr>
<td>precedence</td>
</tr>
<tr>
<td>relational</td>
</tr>
<tr>
<td>square brackets</td>
</tr>
<tr>
<td>subtraction and negation</td>
</tr>
<tr>
<td>XOR</td>
</tr>
<tr>
<td>OPLOT procedure</td>
</tr>
<tr>
<td>OPLOTER procedure</td>
</tr>
<tr>
<td>optimization</td>
</tr>
<tr>
<td>AMOEBA function</td>
</tr>
<tr>
<td>CONSTRAINTED_MIN</td>
</tr>
<tr>
<td>DFPMIN</td>
</tr>
<tr>
<td>POWELL</td>
</tr>
<tr>
<td>OPTIMIZE keyword</td>
</tr>
<tr>
<td>optional parameters in user-written functions</td>
</tr>
<tr>
<td>OR operator</td>
</tr>
<tr>
<td>ORDER keyword</td>
</tr>
<tr>
<td>ORDERED keyword</td>
</tr>
<tr>
<td>ordinary differential equations</td>
</tr>
<tr>
<td>LSODE function</td>
</tr>
<tr>
<td>ordinary differential equations, RK4</td>
</tr>
<tr>
<td>ORIENTATION keyword</td>
</tr>
<tr>
<td>orientation, 3-dimensional</td>
</tr>
<tr>
<td>ORIGIN system variable field</td>
</tr>
<tr>
<td>ORTHOGRAPHIC keyword</td>
</tr>
<tr>
<td>orthographic map projection</td>
</tr>
<tr>
<td>OUT executive command</td>
</tr>
<tr>
<td>OUT_VAL keyword</td>
</tr>
<tr>
<td>outer margins, setting</td>
</tr>
<tr>
<td>outline fonts</td>
</tr>
<tr>
<td>OUTLINE keyword</td>
</tr>
<tr>
<td>OUTPUT keyword</td>
</tr>
<tr>
<td>Output Log</td>
</tr>
<tr>
<td>IDLDE for Macintosh</td>
</tr>
<tr>
<td>IDLDE for Motif</td>
</tr>
<tr>
<td>IDLDE for Windows</td>
</tr>
<tr>
<td>OUTPUT_SIZE keyword</td>
</tr>
<tr>
<td>OUTPUTS keyword</td>
</tr>
<tr>
<td>overflow, integer</td>
</tr>
<tr>
<td>overlaying images and contour plots</td>
</tr>
<tr>
<td>OVERPLOT keyword</td>
</tr>
<tr>
<td>overplotting</td>
</tr>
<tr>
<td>OVERWRITE keyword</td>
</tr>
</tbody>
</table>

| P_CORRELATE function | 435U, 476U, 795R |
| P0 keyword           | 194R |
Using IDL, Building IDL Applications, Object Graphics, Reference Guide

Index

P0LAT keyword 359U
P0LON keyword 359U
P1 - P5 keywords 742R
PACKED keyword 204R
PAIRED keyword 1121R
PALATINO keyword 129R
palette object 31O, 302O, 60O, 66O
PALETTE, see Obsolete Routines
palettes 116O
pan offset 47R
parallel projection 44O
parallels, drawing 359U, 711R, 727R
parameters
actual 110B, 111B
copying 112B
finding number of 765R
formal 110B, 111B, 181O, 185R
passing mechanism 110B, 120B
PARAMETERS keyword 975R
parametric hypothesis tests 447U
PARENT_BASE keyword 614R, 627R, 651R, 667R
parentheses 270U, 28B
parents, of widgets 1275R
partial correlation coefficient 433U, 795R
PARTIAL2_COR, see Obsolete Routines
PARTIAL_COR, see Obsolete Routines
passing parameters 120B
path
definition string 462R
IDLDE for Macintosh 132U, 133U
IDLDE for Motif 65U
IDLDE for Windows 109U
on a Macintosh 44R
PATH environment variable 20U
PATH_DATA_COORDS keyword 305R
PATH_FILENAME keyword 320U, 305R
PATH_INFO keyword 306R
PATH_XY keyword 306R
PATTERN keyword 312O, 827R
pattern object 31O, 310O
pattern objects 66O
PCL driver 154R
files 146R
PCOMP function 476U, 796R
Pearson correlation coefficient 315R
penta.pro (example file) 69O
performance
analyzing 141U
optimizing with memory IDLDE for Macintosh 133U
IDLDE for Windows 58U, 103U
performance tuning with object graphics 145O
period (character) 59R
permutation 473R
PERSP keyword 323R
perspective 1109R
PERSPECTIVE keyword 337U, 1109R
perspective projection 44O
performance limiting object graphics 15O
PID keyword 129R
PLANNER keyword 234R
PLANARCONFG keyword 924R, 1340R
plane of vector-drawn text 1380R
planed vector-drawn text 1380R
Pickdata method 224O, 235O, 351O, 421O, 434O
PICKFILE, see Obsolete Routines
PickVoxel method 412O
PICT files
Pick/Move method 420O
Pick/Insert method 420O
PIXFILE see Obsolete Routines
PIXMAP keyword 1318R
pixmap objects using 136O
pixmap objects pixon
pixels depth 121R
fixed 326U
reading value of 894R
scalable 325U, 383U
PIXELS keyword 129R, 1005R
PIXMAPS keyword 342R
PIXMAP keyword 1318R
Pickdata method 224O, 235O, 351O, 421O, 434O
PICKFILE, see Obsolete Routines
PickVoxel method 412O
PICT files
Pick/Move method 420O
Pick/Insert method 420O
PIXFILE see Obsolete Routines
PIXMAP keyword 1318R
pixmap objects using 136O
pixmap objects pixon
pixels depth 121R
fixed 326U
reading value of 894R
scalable 325U, 383U
PIXELS keyword 129R, 1005R
PIXMAPS keyword 342R
PIXMAP keyword 1318R
Pickdata method 224O, 235O, 351O, 421O, 434O
PICKFILE, see Obsolete Routines
PickVoxel method 412O
PICT files
Pick/Move method 420O
Pick/Insert method 420O
PIXFILE see Obsolete Routines
<table>
<thead>
<tr>
<th>Index</th>
<th>Page Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLOTTER_ON_OFF keyword</td>
<td>129R</td>
</tr>
<tr>
<td>PLOTS procedure</td>
<td>809R</td>
</tr>
<tr>
<td>plotting 284U, 800R</td>
<td>2D fields 806R</td>
</tr>
<tr>
<td>annotation 293U</td>
<td>axes 310U</td>
</tr>
<tr>
<td>scaling 288U</td>
<td>thickness 107R</td>
</tr>
<tr>
<td>titles 109R</td>
<td>bar plots 210R</td>
</tr>
<tr>
<td>closing files (CLOSE_FILE keyword) 117R</td>
<td>combining images with graphics 347U</td>
</tr>
<tr>
<td>contour plots 300R, 559R</td>
<td>drawing axes (AXIS procedure) 207R</td>
</tr>
<tr>
<td>error bars 456R, 794R, 808R</td>
<td>filename for output (FILENAME keyword) 123R</td>
</tr>
<tr>
<td>flow field 488R</td>
<td>font selection 294U</td>
</tr>
<tr>
<td>functions of 2 variables 803R</td>
<td>height of output 143R</td>
</tr>
<tr>
<td>histogram 105R</td>
<td>histogram style 297U</td>
</tr>
<tr>
<td>keyword parameters 284U</td>
<td>landscape orientation 127R</td>
</tr>
<tr>
<td>line thickness 52R, 107R</td>
<td>lines 809R</td>
</tr>
<tr>
<td>linestyles 49R, 102R</td>
<td>location on page 307U</td>
</tr>
<tr>
<td>logarithmic axes</td>
<td>linear-log 801R</td>
</tr>
<tr>
<td>MAX_VALUE keyword</td>
<td>792R, 800R</td>
</tr>
<tr>
<td>multi-dimensional arrays 318U</td>
<td>multiple plots on a page 306U, 50R, 162R</td>
</tr>
<tr>
<td>output, positioning 148R</td>
<td>overplotting 292U, 559R, 792R</td>
</tr>
<tr>
<td>points 809R</td>
<td>polar 312U, 793R, 801R</td>
</tr>
<tr>
<td>portrait orientation 130R</td>
<td>region 51R</td>
</tr>
<tr>
<td>selecting a plotting device 992R</td>
<td>shaded surfaces 1004R</td>
</tr>
<tr>
<td>subtitles 51R, 106R</td>
<td>symbol size 106R</td>
</tr>
<tr>
<td>symbols 295U, 296U, 51R, 105R</td>
<td>text 1379R</td>
</tr>
<tr>
<td>three-dimensional lines 810R</td>
<td>titles 289U, 52R, 109R</td>
</tr>
<tr>
<td>two-dimensional arrays 318U</td>
<td>user-defined symbols 1177R</td>
</tr>
<tr>
<td>velocity field 488R</td>
<td>velocity fields 1184R</td>
</tr>
<tr>
<td>weather fronts 1201R</td>
<td>width of output 142R</td>
</tr>
<tr>
<td>wire-mesh surfaces 1090R</td>
<td>without data 104R</td>
</tr>
<tr>
<td>without erasing 51R, 104R</td>
<td>XY plots 800R</td>
</tr>
<tr>
<td>Z-coordinate for 110R</td>
<td>PM, see Obsolete Routines</td>
</tr>
<tr>
<td>PMF, see Obsolete Routines</td>
<td>PNG files</td>
</tr>
<tr>
<td>standard file format I/O routines 228B</td>
<td>PNT_LINE function 811R</td>
</tr>
<tr>
<td>POINT_LUN procedure 157B, 813R</td>
<td>pointer heap variables 254B</td>
</tr>
<tr>
<td>pointers 230B, 254B</td>
<td></td>
</tr>
</tbody>
</table>
Index

| Creating arrays | 860R |
| Destroying | 856R |
| Examples | 242B |
| Examples of using | 242B |
| File positioning | 136U |
| Macintosh | |
| Windows | 114U |
| Freeing | 242B |
| Heap variables | 230B, 233B |
| Testing existence | 858R |
| Validity | 241B |
| Poisson random deviates | 887R, 891R |
| Polynomial warping | 820R |
| POLYSHADE function | 832R |
| POLYWARP procedure | 835R |
| POPD procedure | 257R, 837R |
| PORTABLE keyword | 247R |
| Portable unformatted I/O | 197B |
| PORTRAIT keyword | 130R |
| Portrait orientation | 159R |
| Positioning | 81R |
| Cursor | 1157R |
| Graphics cursor | 332R |
| PostScript output | 159R |
| Top level base widgets | 1247R |
| Widget bases | 1219R |
| Windows (XPOS and YPOS keywords) | 1318R |
| Positioning objects | 42O |
| PostCreation event property | 201U |
| PostScript | |
| Color | 158R |
| Device | 156R |
| Encapsulated (Encapsulated PostScript Interchange) files | 130R |
| Files | 146R |
| Files with preview headers | 130R |
| Font index | 124R |
| Fonts | 158R, 853R |
| Importing graphics into other programs | 163R |
| Importing into another document | 122R |
| Multiple plots on a single page | 162R |
| Pixel bit depth | 116R |
| Positioning output | 159R |
| Scaling entire plot (SCALE_FACTOR keyword) | 132R |

Using IDL, Building IDL Applications, Object Graphics, Reference Guide Index
traceback information  538R
PROGRESSIVE keyword  624R, 1325R
PROJECT_VOL function  850R
projection  43O
  parallel  44O
  perspective  44O
PROJECTION keyword  393O
projection matrix  455U
projections
  2D from 3D datasets  850R
  Aitoff  365U, 724R
  Alber's equal-area conic  372U
  Albers  724R
azimuthal  362U
  azimuthal equidistant  365U, 724R
backprojection  963R
cylindrical  368U
cylindrical equidistant  371U, 724R
gnomonic (central, gnomic)  363U, 724R
Hammer-Aitoff  366U, 724R
Lambert's conformal conic  371U, 724R
Lambert's equal area  365U, 724R
Mercator  369U, 724R
Miller  724R
Miller cylindrical  371U
Mollweide  374U, 197R
orthographic  362U, 725R
Robinson  373U
satellite  366U, 725R
sinusoidal  373U, 725R
stereographic  362U, 725R
Transverse Mercator (UTM)  369U, 725R
prompt
  changing default  45R
  reading from  896R
PROMPT keyword  896R
PROMPT, see Obsolete Routines
properties
  draw area widget  229U
  entering multiple strings  169U
  label widget  221U
  retrieving  33O
  setting  33O
  table widget  234U
  text widget  216U
Properties dialog  166U
  #of Rows/Columns base property  202U
  Alignment base property  203U
  Alignment button property  213U
  Alignment label property  221U
  Alignment table property  234U
  Allow Moving base property  204U
  Allowing Closing base property  203U
  Bitmap button property  213U
  Color Model draw area property  229U
  Colors draw area property  229U
  Column Labels table property  234U
  Component Sizing common property  197U
  Display Headers table property  234U
draw area events  232U
draw area widget properties  229U
droplist events  224U
droplist widgets  224U
Editable table property  235U
Editable text property  216U
entering multiple strings  169U
Floating base property  204U
Frame common property  197U
Graphics Type draw area property  230U
Grid Layout base property  204U
Handle Events common event  199U
Height listbox property  226U
Height text property  216U
Initial Value droplist property  224U
Initial Value listbox property  226U
Initial Value text property  216U
Label button property  214U
Layout base property  205U
listbox events  227U
listbox properties  226U
Maximum Value slider property  222U
Minimize/Maximize base property  206U
Minimum Value slider property  222U
Modal base property  206U
Multiple listbox property  226U
Name common property  196U
No Release button property  214U
OnButton draw area event  232U
OnButtonPress button event  215U
OnCellSelect table event  237U
OnChangeValue slider event  223U
OnColWidth table event  238U
OnDelete table event  238U
OnDelete text event  218U
OnDestroy common event  200U
OnExplose draw area event  233U
OnFocus base event  210U
OnFocus tab event  238U
Index

QUERY_PICT routine 875R
QUERY_PNG routine 876R
QUERY_PPM routine 878R
QUERY_SRF routine 879R
QUERY_TIFF routine 880R
QUESTION keyword 424R
question mark
  character 61R
  online help system 242U
QuickHelp 250U
QUIET keyword 379O, 952R
quintic interpolation 1136R
QUINTIC keyword 1136R, 1197R
QUIT keyword 782R
quitting IDL 21U, 459R
quotas 139B
quotation marks 258U, 17B, 59R

R
R keyword 884R
R_CORRELATE function 435U, 448U, 882R
R_TEST function 448U, 884R
RADIANS keyword 323R, 470R
radio button widgets
  creating 212U
  creating multiple 212U
  laying out 213U
  setting attributes 213U
  setting properties 212U
  using 165U
radix 703R
Radon transform 963R
random deviates
  binomial 887R, 891R
  exponential 887R, 891R
  gamma 887R, 891R
  normal 891R
  Poisson 887R, 891R
  random 891R
random numbers
  normally-distributed 886R
  uniformly-distributed 890R
RANDOMN function 888R
RANDOMU function 890R
RANGE keyword 211O, 654R
RANGE system variable field 54R
rank correlation coefficient 882R
RANKS function 892R
rank-sum test 977R
raster images 380U
RDPIX procedure 894R
READ keyword 427R
Read method 201O, 225O, 435O
READ procedure 158B, 895R
READ_ASCII function 898R
READ_BMP function 901R
READ_DICOM 176O, 903R
READ_GIF procedure 904R
READ_INTERFILE procedure 906R
READ_JPEG function 907R
READ_KEY procedure 896R
READ_PICT procedure 910R
READ_PNG function 911R
READ_PPM function 913R
READ_SPR function 466U, 915R
READ_SRF procedure 916R
READ SYLK function 918R
READ_TABLES keyword 329R
READ_TIFF function 921R
READ_WAVE procedure 926R
READ_X11_BITMAP procedure 928R
READ_XWD function 930R
READF procedure 158B, 895R
reading
  ASCII files 898R
  BMP files 901R
  current color table 1160R
cursor position 894R
data from a string 931R
from tapes 1114R
GIF files 904R
images from the display 385U, 1161R
Interfile files 906R
JPEG files 907R
mouse position 332R
PGM files 913R
PICT files 910R
pixel values 899R
PPM files 913R
SRF files 916R
TIFF files 921R
unformatted binary data 933R
wave files 926R
X11 bitmaps 928R
<table>
<thead>
<tr>
<th>Index R...R</th>
<th>Using IDL, Building IDL Applications, Object Graphics, Reference Guide Index</th>
</tr>
</thead>
</table>

XWD files 930R
reading and writing files, Macintosh differences 136U
READ_NAMES example routine 242B
read-only system variables 409R
READS procedure 213B, 931R
READU procedure 158R, 933R
real part of complex numbers 486R
REALIZE keyword 1241R
REALIZED keyword 1275R
realizing widgets 300B, 1241R
REBIN function 935R
recall buffer
command 938R
IDLDE for Windows 58U, 103U
RECALL keyword 404R
RECALL_COMMANDS function 938R
RECALL_COMMANDS keyword 539R
recent files
in IDLDE for Macintosh 121U
in IDLDE for Motif 44U
in IDLDE for Windows 90U
RECOMPUTE_DIMENSIONS keyword 386O
RECON function 939R
reconstructions
3D from 2D images 939R
Tomographic 963R
RECORD_START keyword 899R
recording an interactive IDL session 582R
record-oriented files 219B
records
length of 495R
updating 1344R
rectangular
coordinates 338R, 817R
filter 417U
recursion 122B
RED keyword 627R
RED keyword, GREEN keyword, BLUE keyword 1340R
RED_VALUES keyword 2430, 3060
REDRAW keyword 404R
reduce operator 452R
REDUCE_COLORS procedure 944R
_REP_EXTRA keyword (keyword inheritance) 1115B
reference, parameters passed by 120B
REFORM function 945R
reformatting arrays 945R
region
labeling 596R
of interest 362R, 407R
REGION system variable field 51R, 54R
Regis device 168R
REGISTER keyword 1041R
REGRESS function 438U, 947R
REGRESS1, see Obsolete Routines
regression analysis 947R
REGRESSION, see Obsolete Routines
REGULAR keyword 587R, 1129R
REJECT keyword 3300, 994R
relational operators 275U, 34B
relaxed structure assignment 55B, 956R, 1087R
RELAXED_STRUCTURE_ASSIGNMENT keyword 955R
release, current version in use 45R
Remove method 1870, 2920, 3570, 3940, 4000
REMOVE_ALL keyword 77B, 1071R, 1074R
RemoveEntity method 2020
removing breakpoints 229R
RENDER_STEP keyword 4100
Renderer draw area property 230U
rendering 1350
3D objects 740R
3D volumes as 2D images 850R
software vs. hardware 1350
voxel 1192R
Rendering Process
rendering 380
rendering speed of volumes 1270
repeat statement 107B
REPEATS keyword 1133R
replacing text
IDLDE for Macintosh 123U
IDLDE for Motif 47U
IDLDE for Windows 93U
REPLICATE function 346U, 949R
REPLICATE_INPLACE procedure 950R
REPORT keyword 296R, 846R
reserved words 63R
reserving colors 67U
RESET keyword 337U, 4040, 746R, 846R, 1110R, 1241R
Reset method 2020, 2930, 3780, 4430
RESET_DATA keyword 2670, 3210, 3300, 3390, 3650
RESET_STRING keyword 131R
resetting widgets 1241R
RESIDUAL keyword 331R, 444R, 446R, 1376R
Resize Columns table property 235U
RESIZABLE_COLUMNS keyword 1302R
RESIZABLE_ROWS keyword 1302R
resizing arrays 291R, 461R, 935R
RESOLUTION 429O
resolution, CIA World map database 377U
RESOLVE_ALL procedure 952R
RESOLVE_ROUTINE procedure 954R
resource files, editing 67U
resources, X Window 66U
RESTORE keyword 362R, 408R
RESTORE procedure 34U, 955R, 1404
RESTORED_OBJECTS keyword 956R
restoring IDL save files 955R
restoring structures 57B
RESULT keyword 438R
RESULT_ACMODE keyword 1146R
RESULT_TABLE keyword 1146R
Retain draw area property 231U
Resized Graphics 148O
RECALL command 957R
RECALL procedure 30U
RETURN command 958R
RETURN executive command See commands
RETURN procedure 30U
RETURN_EVENTS keyword 369R
RETURN_FULL_NAME keyword 384R
RETURN_ID keyword 355R, 384R
RETURN_INDEX keyword 355R, 384R
RETURN_NAME keyword 356R, 384R
RETURN_TYPE keyword 247R
returning
  subscripts of non-zero array elements 1203R
  widget information 1271R
REVERSE function 960R
reverse index list (for histograms) 546R
REVERSE_INDICES keyword 547R
reversing array indices 960R
REWIND procedure 962R
REWRITE keyword 842R, 1344R
RGB color model 58O, 59O
RGB color system 386U, 277R, 387R, 1159R
RGB keyword 388R, 901R
RGB_HLS keyword 277R
RGB_HSV keyword 277R
RGB_TABLE0 keyword 410O
RGB_TABLE1 keyword 410O
RGB_TO_HSV, see Obsolete Routines
Rich Text Format 249U
RIEMANN procedure 963R
RIGHT keyword 1385R
right-handed coordinate system 335U
rivers 708R
RIVERS keyword 709R
RK4 function 967R
RM, see Obsolete Routines
RMF, see Obsolete Routines
RMS block mode 788R
RNAME_MBAR keyword 1213R
Roberts edge enhancement 969R
ROBERTS function 969R
Robinson map projection 373U
Romberg integration 862R, 864R
ROOT_DIR keyword 478R
roots 500R, 502R
ROT function 970R, 972R
ROT keyword 359U
ROT_INT, see Obsolete Routines
rot_text.pro (example file) 84O
ROTATE function 972R
ROTATE keyword 337U, 211R, 1110R
Rotate method 293O
rotate method 49O
rotating
  arrays 336U, 972R
  images 336U, 350R
  by arbitrary amounts 970R
  the viewing matrix 1109R
  views 343U
rotation 48O, 490
ROUND function 974R
rounding 703R
ceiling function 259R
floor function 487R
to nearest integer 974R
ROUTINE_INFO function 975R
routines
Index

accessing 29U
obsolete 1391R
saving as binary files 982R
ROUTINES keyword 540R, 982R
row bases 1213R
Row Labels table property 236U
Row/Column Major table property 235U
ROW_HEIGHTS keyword 1241R, 1275R, 1302R, 1213R
ROW_LABELS keyword 1241R, 1303R
ROW_MAJOR keyword 1303R
row-indexed sparse storage method 464U
row-major format 429U
RS_TEST function 448U, 977R
RSI electronic mail address 16U, 9B, 20O, 29R
RSI postal address 16U, 9B, 20O, 29R
RSI telephone and fax numbers 16U, 9B, 20O, 29R
RSI_GAMMAI, see Obsolete Routines
RSTRPOS function 979R
RTF 249U
RTOL keyword 692R
RUN executive command See commands
Runge-Kutta method 967R
run-length encoding 829R
running IDL 20U
runs test for randomness 884R
SAVE keyword 207R, 663R, 1006R, 1092R
Save method 300O
SAVE procedure 34U, 982R, 1404
save/restore
   binary files 982R
   files 955R
     Macintosh differences 136U
     Windows differences 114U
   heap variables 232B
SAVE_DIVIDE keyword 1186R
SAVE_HOURGLASS keyword 1269R
saved commands, displaying 539R
saving
   files
     IDLDE for Macintosh 121U
     IDLDE for Motif 44U
     IDLDE for Windows 89U
   IDL routines as binary files 982R
   IDL variables 982R
   system variables 983R
   variables 983R
saving and restoring heap variables 232B
saving and restoring windows 1370
scalable pixels 325U, 383U, 149R
Scale method 294O
scale method 500
SCALE_FACTOR keyword 132R
SCALE3 procedure 338U, 984R
SCALE3D procedure 985R
scaling 335U, 480, 500, 1109R
   axes 288U
   factors 54R
   images 384U
   values into range of bytes 238R
SCATTER keyword 653R
scene object 280, 353O
scene objects 360
SCHOOLBOOK keyword 133R
screen size, finding 308B
SCREEN_DIMENSIONS keyword 221O, 232O, 429O
scripts, AppleScript 438R
scroll bars
   for draw widgets 1256R
   for text widgets 1309R, 1312R
Scroll base property 206U
Scroll draw area property 231U
scroll offset 47R
Scroll table property 236U
Scroll text property 217U
SDEV keyword 604R, 756R
<table>
<thead>
<tr>
<th>Index</th>
<th>Index-57</th>
</tr>
</thead>
<tbody>
<tr>
<td>search path</td>
<td></td>
</tr>
<tr>
<td>IDLIDE for Macintosh</td>
<td>132U, 133U</td>
</tr>
<tr>
<td>IDLIDE for Motif</td>
<td>66U</td>
</tr>
<tr>
<td>IDLIDE for Windows</td>
<td>109U</td>
</tr>
<tr>
<td>SEARCH2D function</td>
<td>986R</td>
</tr>
<tr>
<td>SEARCH3D function</td>
<td>989R</td>
</tr>
<tr>
<td>searching, within strings</td>
<td>1084R</td>
</tr>
<tr>
<td>seasonal effect</td>
<td>467U</td>
</tr>
<tr>
<td>segmentation</td>
<td>596R</td>
</tr>
<tr>
<td>SEGMENTED keyword</td>
<td>789R</td>
</tr>
<tr>
<td>set_obj.pro (example file)</td>
<td>131O</td>
</tr>
<tr>
<td>Select method</td>
<td>2260, 4360</td>
</tr>
<tr>
<td>SELECT_TARGET keyword</td>
<td>291O</td>
</tr>
<tr>
<td>selection</td>
<td>130O</td>
</tr>
<tr>
<td>example</td>
<td>131O</td>
</tr>
<tr>
<td>graphic atoms</td>
<td>131O</td>
</tr>
<tr>
<td>IDLgrWindow object</td>
<td>130O</td>
</tr>
<tr>
<td>model objects</td>
<td>131O</td>
</tr>
<tr>
<td>views</td>
<td>1300</td>
</tr>
<tr>
<td>self argument (objects)</td>
<td>262B</td>
</tr>
<tr>
<td>semicolon</td>
<td>59R</td>
</tr>
<tr>
<td>SEND_EVENT keyword</td>
<td>1242R</td>
</tr>
<tr>
<td>Sensitive common property</td>
<td>198U</td>
</tr>
<tr>
<td>sensitizing widgets</td>
<td>301B, 1242R</td>
</tr>
<tr>
<td>SEPARATOR keyword</td>
<td>1225R</td>
</tr>
<tr>
<td>SET keyword</td>
<td>230R, 1358R</td>
</tr>
<tr>
<td>SET_BUTTON keyword</td>
<td>1242R</td>
</tr>
<tr>
<td>SET_CHARACTER_SIZE keyword</td>
<td>133R</td>
</tr>
<tr>
<td>SET_COLORMAP keyword</td>
<td>133R</td>
</tr>
<tr>
<td>SET_DRAW_VIEW keyword</td>
<td>1243R</td>
</tr>
<tr>
<td>SET_DROPLIST_SELECT keyword</td>
<td>1243R</td>
</tr>
<tr>
<td>SET_FONT keyword</td>
<td>134R</td>
</tr>
<tr>
<td>SET_GRAPHICS_FUNCTION keyword</td>
<td>137R</td>
</tr>
<tr>
<td>SET_LIST_SELECT keyword</td>
<td>1243R</td>
</tr>
<tr>
<td>SET_LIST_TOP keyword</td>
<td>1243R</td>
</tr>
<tr>
<td>SET_NATIVE_PLOT, see Obsolete Routines</td>
<td></td>
</tr>
<tr>
<td>SET_PLOT procedure</td>
<td>111R, 992R</td>
</tr>
<tr>
<td>SET_RESOLUTION keyword</td>
<td>138R</td>
</tr>
<tr>
<td>SET_SCREEN, see Obsolete Routines</td>
<td></td>
</tr>
<tr>
<td>SET_SHADING procedure</td>
<td>350U, 832R, 994R</td>
</tr>
<tr>
<td>SET_SLIDER_MAX keyword</td>
<td>1243R</td>
</tr>
<tr>
<td>SET_SLIDER_MIN keyword</td>
<td>1244R</td>
</tr>
<tr>
<td>SET_STRING keyword</td>
<td>138R</td>
</tr>
<tr>
<td>SET_SYMBOL procedure</td>
<td>996R</td>
</tr>
<tr>
<td>SET_TABLE_SELECT keyword</td>
<td>1244R</td>
</tr>
<tr>
<td>SET_TABLE_VIEW keyword</td>
<td>1244R</td>
</tr>
<tr>
<td>SET_TEXT_SELECT keyword</td>
<td>1244R</td>
</tr>
<tr>
<td>SET_TEXT_TOP_LINE keyword</td>
<td>1244R</td>
</tr>
<tr>
<td>SET_TRANSLATION keyword</td>
<td>138R</td>
</tr>
<tr>
<td>SET_UNAME keyword</td>
<td>1245R</td>
</tr>
<tr>
<td>SET_UVALUE keyword</td>
<td>1245R</td>
</tr>
<tr>
<td>SET_VALUE keyword</td>
<td>356R, 1245R</td>
</tr>
<tr>
<td>SET_VIEW.PRO</td>
<td>48O</td>
</tr>
<tr>
<td>SETVIEWPORT, see Obsolete Routines</td>
<td></td>
</tr>
<tr>
<td>SET_WRITE_MASK keyword</td>
<td>1245R</td>
</tr>
<tr>
<td>SET_XY, see Obsolete Routines</td>
<td></td>
</tr>
<tr>
<td>SetCurrentCursor method</td>
<td>4370</td>
</tr>
<tr>
<td>SetENV procedure</td>
<td>520R, 997R</td>
</tr>
<tr>
<td>setjmp, C language</td>
<td>255R</td>
</tr>
<tr>
<td>SETLOG procedure</td>
<td>988R</td>
</tr>
<tr>
<td>SetPalette method</td>
<td>2020</td>
</tr>
<tr>
<td>SetProperty method</td>
<td>214O, 227O, 245O, 255O, 290O, 2680, 2760, 2840, 2940, 3010, 3080, 3130, 3220, 3330, 3410, 3520, 3580, 3680, 3730, 3870, 3950, 4010, 4130, 4220, 4380, 4390</td>
</tr>
<tr>
<td>SETRGB method</td>
<td>3080</td>
</tr>
<tr>
<td>setting</td>
<td></td>
</tr>
<tr>
<td>breakpoints</td>
<td>230R</td>
</tr>
<tr>
<td>keywords</td>
<td>111B, 181O, 185R, 585R</td>
</tr>
<tr>
<td>properties</td>
<td>330</td>
</tr>
<tr>
<td>properties at initialization</td>
<td>330</td>
</tr>
<tr>
<td>properties of existing objects</td>
<td>330</td>
</tr>
<tr>
<td>the current window</td>
<td>1346R</td>
</tr>
<tr>
<td>values</td>
<td>1245R</td>
</tr>
<tr>
<td>setting the cursor in a window object</td>
<td>1370</td>
</tr>
<tr>
<td>SETUP keyword</td>
<td>655R</td>
</tr>
<tr>
<td>SETUP_KEYS procedure</td>
<td>399R, 1000R</td>
</tr>
<tr>
<td>SFIT function</td>
<td>438U, 1002R</td>
</tr>
<tr>
<td>SGI keyword</td>
<td>1001R</td>
</tr>
<tr>
<td>SGML</td>
<td>250U</td>
</tr>
<tr>
<td>SH keyword</td>
<td>1049R</td>
</tr>
<tr>
<td>SHADE_RANGE keyword</td>
<td>3300, 3650</td>
</tr>
<tr>
<td>SHADE_SURF procedure</td>
<td>1004R</td>
</tr>
<tr>
<td>SHADE_SURF_IRR procedure</td>
<td>1008R</td>
</tr>
<tr>
<td>SHADE_VOLUME procedure</td>
<td>351U, 834R, 1010R</td>
</tr>
<tr>
<td>shaded surfaces</td>
<td>1004R</td>
</tr>
<tr>
<td>changing position of light source</td>
<td>994R</td>
</tr>
<tr>
<td>from polygons</td>
<td>832R</td>
</tr>
<tr>
<td>plotting</td>
<td>349U</td>
</tr>
<tr>
<td>The SHADE_SURF Procedure</td>
<td>349U</td>
</tr>
<tr>
<td>SHADERS keyword</td>
<td>352U, 833R, 1006R, 1011R, 1092R</td>
</tr>
<tr>
<td>shading</td>
<td>349U, 994R</td>
</tr>
<tr>
<td>changing position of light source</td>
<td>994R</td>
</tr>
<tr>
<td>parameters</td>
<td>350U</td>
</tr>
</tbody>
</table>
volumes 832R
SHADING keyword 3300, 3390, 3650
SHARE_DATA keyword 2670, 3210, 3310, 3400, 3650
shared colormap 138R, 140R
Motif IDLDE 67U
SHARED keyword 789R
sheet feeder 122R
shells, spawning 1047R
SHIFT function 1013R
shifting
array elements 1013R
bit 581R
SHORT keyword 1340R
short word swap 236R
shortcuts, keyboard
IDLDE for Motif 55U
IDLDE for Windows 100U
SHOW keyword 1246R
SHOW_AXIS keyword 2430
SHOW_FILL keyword 2750
SHOW_FULL keyword 1041R
SHOW_OUTLINE keyword 2430, 2750
SHOW_SKIRT keyword 366O
SHOW3 procedure 1015R
SHOWFONT procedure 1017R
showing
images 1153R
windows 1347R
SHOWLOAD keyword 1358R
shrink operator 452R
shrinking
arrays 935R
windows 1347R
SIBLING keyword 1275R
SIG_AB keyword 650R
SIGMA keyword 283R, 678R, 1099R
SIGMA, see Obsolete Routines
sign test 980R
SIGN_TEST, see Obsolete Routines
signal
analysis transforms 400U
filtering 225R
processing 398U
CONVOL function 311R
SIGNED keyword 823R
significant bits 121R
SILENT keyword 685R, 1362R
simple polygons 710, 3750

SIMPLEX keyword 194R
SIMPSON, see Obsolete Routines
Simpson's rule 866R
simultaneous linear equations, solving 453U
SIN function 1019R
SINDGEN function 1020R
sine 1019R
hyperbolic 1021R
inverse 203R
single-precision
arrays 482R, 490R
conversion values to 486R
SINGULAR keyword 1099R
Singular Value Decomposition 438U, 454U, 1096R,
1103R
SINH function 1021R
SINOSORT example routine 245B
SINUSOIDSAL keyword 725R
sinusoidal map projection 373U, 725R
SIZ keyword 436R
size
of arrays 1022R
of widgets 306B
SIZE,executive command 1393R
SIZE function 124B, 127B, 1022R
SIZE keyword 2590, 373O, 351R, 706R
sizing widgets 304B
sizing windows
graphics
IDLDE for Windows 106U
IDLDE for Macintosh 130U
IDLDE for Motif 61U
sketches of bi-level images 1118R
skewness 755R, 1025R
SKINESS function 1025R
SKIP, executive command See commands
SKIPF procedure 1026R
SKIRT keyword 366O, 1093R
skirts (surface objects) 1060
slash character 181O, 185R
SLICER procedure 354U, 522R
SLICER, see Obsolete Routines
SLICER3 procedure 1027R
SLIDE_IMAGE procedure 1040R
SLIDE_WINDOW keyword 1041R
slider widgets 223U, 274B, 1291R
changing maximum value 1243R
changing minimum value 1244R
displayed values 222U
drag events 1296R
draggable 1291R
events returned by 1296R
floating-point 377R
initial position 222U
maximum value 222U, 1293R
minimum value 222U, 1293R
properties 222U
returning minimum and maximum values 1276R
setting attributes 222U
setting events 223U
title 223U
using 165U
SLIDER_MIN_MAX keyword 1276R
smearing (in frequency plots) 405U
SMOOTH function 423U, 469U, 1043R
smoothing 1043R
contours 329U
CONVOL function 311R
element 161U
median 738R
MIN_CURVE_SURF function 300R
Sobel function 1047R
SOLID keyword 200R
SOLID_WALLS keyword 804R
SORT function 1046R
sorting
arrays 1046R
SINKSORT example 245B
SOURCE keyword 976R
SOURCE_FILES keyword 540R
Space base property 207U
SPACE keyword 356R, 1214R
spaces, removing from a string 77B
SPACING keyword 313O, 709R, 827R
sparse arrays 464U
FULSTR 496R
LINBCG 601R
READ_SPR 915R
SPRSAB 1061R
SPRSAX 1063R
SPRSIN 1064R
WRITE_SPR 1332R
SPARSE keyword 694R
spawning a shell process 1047R
SPAWN procedure 31U, 1047R
SPEARMAN, see Obsolete Routines
Spearman's rho rank correlation 882R
special characters 26U
displaying in plots 68R
special functions
BETA 216R
IBETA 554R
Specifying the Location of a Plot 307U
SPH_4PNT procedure 1051R
SPH_SCAT function 1052R
SPHERE keyword 1133R, 1136R
spherical coordinates 338R
spherical gridding 1052R, 1131R, 1134R
spherical interpolation 1052R
SPHERICAL keyword 587R
spherical triangulation 1131R
SPL_INIT function 445U, 1054R
SPL_INTERP function 445U, 1056R
spline
cubic interpolation 1054R, 1058R, 1059R
thin-plate surface 748R
SPLINE function 445U, 1058R
SPLINE keyword 320U
SPLINE_P procedure 446U, 1059R
spreading (spreadsheet data files) 918R, 1335R
SPRSAB function 466U, 1061R
SPRSAX function 466U, 1063R
SPRSIN function 466U, 1064R
SORT function 1066R
square brackets 270U, 28B
See arrays, concatenation
square root 1066R
SRF files
reading 916R
standard file format I/O routines 228B
writing 1333R
SSCALE keyword 1016R
SSWAP keyword 236R
stacked histogram plots (LEGO keyword) 1091R
standard
deviation 755R
image file formats 227B
input 514R
standard deviation 1069R
Standard Generalized Markup Language, SGML 250U
STANDARD keyword 438O
STANDARD_PARALLELS keyword 730R
STANDARDIZE function 476U, 1067R
STANDARDIZE keyword 797R
standardized variables 472U, 1067R
START_COLOR keyword 358R
START_OF_LINE keyword 404R
Using IDL, Building IDL Applications, Object Graphics, Reference Guide Index
Index

- STARTCOL keyword 919R, 1335R
- starting IDL 20U
- STARTROW keyword 919R, 1335R
- startup file 33U
- IDLDE for Macintosh 132U
- IDLDE for Motif 64U
- IDLDE for Windows 108U
- preferences IDLDE for Macintosh 132U
- IDLDE for Motif 64U
- IDLDE for Windows 107U
- switches 20U
- statement labels 84B
- statements 84B
- STATIC_COLOR keyword 139R
- STATIC_GRAY keyword 139R
- stationarity 467U
- STATIONARY keyword 1202R
- statistics 426U
- approximating models 282R
- fitting data
- growth trends 282R
- least absolute deviation regression 598R
- moving averages 1043R
- multiple linear regression 947R
- nonlinear least-squares regression 335R
- outlying data regression 598R
- kurtosis 590R
- testing 446U
- tools absolute deviation 755R
- Chi-square error, minimizing 604R
- combinations 473R
- contingency table 330R
- cumulative sum 1123R
- factorial 473R
- frequency tables 546R
- histogram 546R
- kurtosis 590R, 755R
- Lomb normalized periodogram 683R
- magnitude-based ranking 892R
- maximum 731R
- mean 735R, 755R
- mean absolute deviation 736R
- median 755R
- minimum 747R
- number generators 840R, 886R, 890R
- permutations 473R
- skewness 755R, 1025R
- sort 1046R
- standard deviation 755R, 1069R
- T-statistic, Student’s 1121R
- variance 755R, 1179R
- STATISTICS keyword 299O
- Status Bar
- IDLDE for Motif 42U
- IDLDE for Windows 86U
- STDDEV function 1069R
- STDEV, see Obsolete Routines
- STEP executive command See commands
- STEP keyword 1193R
- step plot 399U
- STEPMAX keyword 232R, 421R, 768R
- STEPOVER executive command See commands
- stepping through, debugging 139U
- STEPWISE, see Obsolete Routines
- STEREOGRAPHIC keyword 725R
- stereographic map projection 362U, 725R
- STIRLING keyword 473R
- STOP keyword 348R
- STOP procedure 130B, 1070R
- stopping program execution 30U, 229R, 1070R
- STR_SEP function 1071R
- STRARR function 1073R
- STRCOMPRESS function 77B, 78B, 1074R
- STREAM keyword 789R
- streamlines 1182R
- STRETCH procedure 391U, 1075R
- string data type 13B
- STRING function 74B, 75B, 212B, 1077R
- STRING keyword 369R, 561R, 706R
- strings 258U, 259U, 17B
- calling IDL functions from 252R
- IDL methods from 253R
- IDL procedures from 254R
- case folding 76B
- concatenation 74B
- converting to lowercase 1081R
- converting to uppercase 1089R
- creating arrays 1020R
- creating string arrays 1073R
- data type, converting to 1077R
- executing contents of 458R
- extracting substrings from 1083R
- finding substrings within 979R, 1084R
- formatting data 74B

Index S ... S Using IDL, Building IDL Applications, Object Graphics, Reference Guide Index
<table>
<thead>
<tr>
<th>Index</th>
<th>Index-61</th>
</tr>
</thead>
<tbody>
<tr>
<td>inserting strings into 1085R</td>
<td></td>
</tr>
<tr>
<td>length of 79B, 1080R</td>
<td></td>
</tr>
<tr>
<td>nonstring arguments to routines 73B</td>
<td></td>
</tr>
<tr>
<td>operations 72B, 979R, 1071R</td>
<td></td>
</tr>
<tr>
<td>reading data from 931R</td>
<td></td>
</tr>
<tr>
<td>removing whitespace from 1074R, 1086R</td>
<td></td>
</tr>
<tr>
<td>separating into parts 1071R</td>
<td></td>
</tr>
<tr>
<td>substrings 79B</td>
<td></td>
</tr>
<tr>
<td>whitespace 77B</td>
<td></td>
</tr>
<tr>
<td>STRINGS keyword 386O</td>
<td></td>
</tr>
<tr>
<td>STRLEN function 79B, 1080R</td>
<td></td>
</tr>
<tr>
<td>STRLOWCASE function 76B, 1081R</td>
<td></td>
</tr>
<tr>
<td>STRM ESSION function 1082R</td>
<td></td>
</tr>
<tr>
<td>STRMID function 79B, 81B, 1083R</td>
<td></td>
</tr>
<tr>
<td>STRPOS function 79B, 1084R</td>
<td></td>
</tr>
<tr>
<td>STRPUT procedure 79B, 80B, 1085R</td>
<td></td>
</tr>
<tr>
<td>STRTRIM function 77B, 78B, 1086R</td>
<td></td>
</tr>
<tr>
<td>STRUCT_ASSIGN procedure 55B, 1087R</td>
<td></td>
</tr>
<tr>
<td>STRUCTURE keyword 1023R</td>
<td></td>
</tr>
<tr>
<td>Structure objects 280</td>
<td></td>
</tr>
<tr>
<td>structure of subarrays 65B</td>
<td></td>
</tr>
<tr>
<td>STRUCTURE_NAME keyword 1111R</td>
<td></td>
</tr>
<tr>
<td>structures 540R</td>
<td></td>
</tr>
<tr>
<td>advanced 53B</td>
<td></td>
</tr>
<tr>
<td>arrays of 49B</td>
<td></td>
</tr>
<tr>
<td>arrays stored in structure form 464U</td>
<td></td>
</tr>
<tr>
<td>automatic definition 54B, 252B</td>
<td></td>
</tr>
<tr>
<td>concatenating 321R</td>
<td></td>
</tr>
<tr>
<td>creating and defining 262U, 42B, 54B, 321R</td>
<td></td>
</tr>
<tr>
<td>creating arrays of 949R</td>
<td></td>
</tr>
<tr>
<td>definition 55B, 1087R</td>
<td></td>
</tr>
<tr>
<td>displaying information on currently-defined 540R</td>
<td></td>
</tr>
<tr>
<td>FSTAT 494R</td>
<td></td>
</tr>
<tr>
<td>input/output 51B</td>
<td></td>
</tr>
<tr>
<td>introduction to 262U, 42B</td>
<td></td>
</tr>
<tr>
<td>number of fields in 53B</td>
<td></td>
</tr>
<tr>
<td>parameter passing 264U, 47B</td>
<td></td>
</tr>
<tr>
<td>references 263U, 45B</td>
<td></td>
</tr>
<tr>
<td>relaxed definition 956R, 1087R</td>
<td></td>
</tr>
<tr>
<td>relaxed definition 55B</td>
<td></td>
</tr>
<tr>
<td>restoring 57B</td>
<td></td>
</tr>
<tr>
<td>returned by widgets 1270R</td>
<td></td>
</tr>
<tr>
<td>returning length of 766R</td>
<td></td>
</tr>
<tr>
<td>returning number of tags 766R</td>
<td></td>
</tr>
<tr>
<td>tag names 321R, 1111R</td>
<td></td>
</tr>
<tr>
<td>using help with 264U, 47B</td>
<td></td>
</tr>
<tr>
<td>zoned 43B, 252B</td>
<td></td>
</tr>
<tr>
<td>STRUCTURES keyword 540R</td>
<td></td>
</tr>
<tr>
<td>structuring element 432R</td>
<td></td>
</tr>
<tr>
<td>STRUPCASE function 76B, 1089R</td>
<td></td>
</tr>
<tr>
<td>STUDENT keyword 681R</td>
<td></td>
</tr>
<tr>
<td>STUDENT_T, see Obsolete Routines</td>
<td></td>
</tr>
<tr>
<td>Student's t distribution 1107R, 1108R</td>
<td></td>
</tr>
<tr>
<td>Student's T-statistic 1121R</td>
<td></td>
</tr>
<tr>
<td>STUDENT1_T, see Obsolete Routines</td>
<td></td>
</tr>
<tr>
<td>STUDRANGE, see Obsolete Routines</td>
<td></td>
</tr>
<tr>
<td>STYLE system variable field 54R</td>
<td></td>
</tr>
<tr>
<td>STYLES keyword 220O, 231O, 345O, 417O, 4280</td>
<td></td>
</tr>
<tr>
<td>SUB_RECT keyword 924R</td>
<td></td>
</tr>
<tr>
<td>SUBDIRECTORY keyword 478R</td>
<td></td>
</tr>
<tr>
<td>SUBMIT keyword 790R</td>
<td></td>
</tr>
<tr>
<td>submitting a print job 141O</td>
<td></td>
</tr>
<tr>
<td>subscripts 60B</td>
<td></td>
</tr>
<tr>
<td>array valued 66B</td>
<td></td>
</tr>
<tr>
<td>arrays 261U</td>
<td></td>
</tr>
<tr>
<td>examples 61B</td>
<td></td>
</tr>
<tr>
<td>of scalars 63B</td>
<td></td>
</tr>
<tr>
<td>ranges 64B</td>
<td></td>
</tr>
<tr>
<td>ranges, combined with arrays 67B</td>
<td></td>
</tr>
<tr>
<td>subscript arrays 87B, 89B</td>
<td></td>
</tr>
<tr>
<td>SUBSTITUTE keyword 2590</td>
<td></td>
</tr>
<tr>
<td>substrings 79B</td>
<td></td>
</tr>
<tr>
<td>SUBTICKLEN 211O</td>
<td></td>
</tr>
<tr>
<td>SUBTICKLEN keyword 211O, 2430</td>
<td></td>
</tr>
<tr>
<td>Subtitle keyword 106R</td>
<td></td>
</tr>
<tr>
<td>SUBTITLE system variable field 51R</td>
<td></td>
</tr>
<tr>
<td>subtraction operator 271U, 29B</td>
<td></td>
</tr>
<tr>
<td>SUBTYPE keyword 647R</td>
<td></td>
</tr>
<tr>
<td>summation, array elements 1123R</td>
<td></td>
</tr>
<tr>
<td>SUN keyword 1001R</td>
<td></td>
</tr>
<tr>
<td>Sun raster files</td>
<td></td>
</tr>
<tr>
<td>reading 916R</td>
<td></td>
</tr>
<tr>
<td>writing 1333R</td>
<td></td>
</tr>
<tr>
<td>SUPERCLASS keyword 771R</td>
<td></td>
</tr>
<tr>
<td>SUPERSEDE keyword 790R</td>
<td></td>
</tr>
<tr>
<td>Suppress Value slider property 222U</td>
<td></td>
</tr>
<tr>
<td>SUPPRESS_ERROR keyword 579R</td>
<td></td>
</tr>
<tr>
<td>SUPPRESS_VALUE keyword 378R, 1294R</td>
<td></td>
</tr>
<tr>
<td>suppressing information messages 45R</td>
<td></td>
</tr>
<tr>
<td>surf_track.pro (example file) 110O, 4410</td>
<td></td>
</tr>
<tr>
<td>surface fitting 436U</td>
<td></td>
</tr>
<tr>
<td>SFIT 1002R</td>
<td></td>
</tr>
<tr>
<td>surface object 300, 359O</td>
<td></td>
</tr>
<tr>
<td>surface objects 104O</td>
<td></td>
</tr>
<tr>
<td>creating 104O</td>
<td></td>
</tr>
<tr>
<td>example 1100</td>
<td></td>
</tr>
</tbody>
</table>
hidden line removal 1060
rendering style 1050
shading 1050
skirts 1060
texture mapping 1070
using 1050
surface plots 331U, 1377R
  with images and contours 1015R
SURFACE procedure 331U, 1090R
  duplicating transformations 1095R
SURFACE_FIT, see Obsolete Routines
surfaces, shaded 740R, 1004R, 1008R
SURFR procedure 1095R
suspending execution 140U
SVBKSB, see Obsolete Routines
SVD, see Obsolete Routines
SVD_fit function 438U, 1098R
SVSOL function 460U, 1102R
SWAP_ENDIAN function 1104R
SWAP_ENDIAN keyword 786R
SWAP_IF_BIG_ENDIAN keyword 236R, 786R
SWAP_IF_LITTLE_ENDIAN keyword 236R, 786R
swapping the order of bytes 235R
switches, command-line 20U
SX keyword 488R, 848R
SY keyword 488R, 848R
SYLK files 918R, 1335R
SYM_HT keyword 1202R
SYM_LEN keyword 1202R
SYM_BOL keyword 3210, 3400, 139R
symbol object 310, 370O
symbol objects 67O
symbolic link files 918R, 1335R
symmetric
  array or matrix 1140R, 1142R
  arrays 430U
SYM_SIZE keyword 106R
system
  clock 1105R
  files 139B
SYSTEM keyword 846R, 976R
System Menu base property 207U
system variable fields
  BACKGROUND 49R
  BLOCK 39R
  CHANNEL 49R
  CHARSIZE 49R, 52R
  CHARTICK 49R
  CLIP 49R
  CODE 39R
  COLOR 49R
  CRANGE 52R
  FILL_DIST 46R
  FLAGS 46R
  FONT 49R
  GRIDSTYLE 53R
  LINESTYLE 49R
  MARGIN 53R
  MINOR 53R
  MSG 39R
  MSG_PREFIX 39R
  MULTI 50R
  N_COLORS 46R
  NAME 39R, 46R
  NOCLIP 51R
  NOERASE 51R
  NSUM 51R
  OMARGIN 53R
  ORIGIN 47R
  POSITION 51R
  PSYM 51R
  RANGE 54R
  REGION 51R, 54R
  S 54R
  STYLE 54R
  SUBTITLE 51R
  SYS_CODE 39R
  SYS_MSG 39R
  T 51R
  T3D 51R
  TABLE_SIZE 47R
  THICK 52R, 55R
  TICKFORMAT 55R
  TICKLEN 52R, 55R
  TICKNAME 55R
  TICKS 56R
  TICKV 56R
  TITLE 52R, 56R
  TYPE 56R
  UNIT 47R
  WINDOW 48R, 56R
  X_CH_SIZE 48R
  X_PIXELS_CM 48R
  X_SIZE 48R
  X_VSIZE 48R
  Y_CH_SIZE 48R
creating 409R
displaying information on currently-defined 540R
for axes 52R
for errors 149B
for graphics 46R
obsolete 1391R
read-only 409R
saving 983R
SYSTEM_VARIABLES keyword 540R, 983R
SYSTIME function 1105R
SZ keyword 488R

T
system variable field 51R
_T_CVF function 1107R
_T_PDF function 1108R
T3D keyword 107R, 310R, 833R, 1009R
T3D procedure 336U, 1109R
T3D system variable field 51R
TABLE keyword 998R, 1146R
table widgets 237U, 275B, 1298R
alignment of text 234U
attributes 234U
cell select events 237U
column labels 234U
column width change events 238U
data invalid events 239U
data transfer to 235U
editing cells 235U
events 237U
focus events 238U
heading display 234U
height 235U
insert character events 239U
insert string events 239U
keyboard focus events 1301R
row labels 236U
scroll height 237U
scroll width 236U
scrolling 236U
sizing columns 235U
text delete events 238U
text selection events 240U
using 166U
width 235U
TABLE_ALL_EVENTS keyword 1276R
TABLE_EDIT_CELL keyword 1276R
TABLE_EDITTABLE keyword 1276R
TABLE_SELECT keyword 1276R
TABLE_SIZE system variable field 47R
TABLE_VIEW keyword 1276R
TABLE_XSIZE keyword 1246R
TABLE_YSIZE keyword 1246R
tabs, removing from a string 77B
tabulated data 444U
TAG_NAMES function 1111R
tags, number in a structure 766R
TAN function 1112R
TAN0 keyword 1059R
tangent 1112R
inverse 206R
TANH function 1113R
tapes
reading from 1114R
rewinding 962R
skipping records 1026R
writing data to 1115R
writing EOF mark 1200R
TAPRD procedure 1114R
TAPWRT procedure 1115R
TEK_COLOR procedure 1116R
Index-64

Index

TEK_COLORS keyword 197R
TEK4014 keyword 139R
TEK4100 keyword 140R
Tektronix device 169R
telephone and fax numbers for RSI 16U, 9B, 200, 29R
TEMP_DIRECTORY keyword 2990
TEMP PLATE keyword 533R, 899R
TEMPORARY function 137B, 1117R
temporary variables 1117R
TERM environment variable 23U
TERMINAL keyword 478R, 998R
TERMINATE keyword 402R
ternary operator, ?: 36B
tessellation 1131R
Tessellate method 378O
tessellator object 31O, 375O
tessellator objects 71O
test functions 683R
CTI_TEST 330R
FV_TEST 498R
KW_TEST 591R
LNP_TEST 683R
MD_TEST 733R
R_TEST 884R
RS_TEST 977R
S_TEST 980R
TM_TEST 1121R
XSQ_TEST 1375R
test mode, IDL GUIBuilder 158U
test_surface.pro (example file) 53O
TESTCONTRAST, see Obsolete Routines
text
aligning 1379R
character
  height 48R
  size 49R
  thickness 49R, 1380R
  width 48R
displaying 1354R
font index 102R
font selection 49R
plane of 1380R
plotting in graphics windows 1379R
positioning 81R
replacing
  IDLDE for Macintosh 123U
  IDLDE for Motif 47U
  IDLDE for Windows 93U
searching for

IDLDE for Macintosh 122U
IDLDE for Motif 46U
IDLDE for Windows 92U
selecting, IDLDE for Windows 111U
size 100R
size of characters 1380R
widgets, See text widgets
width of 1380R
TEXT keyword 1354R
Text label property 221U
text object 300, 3800
text objects 80O
  3D 82O
  alignment 82O
  baseline and upward direction 82O
  creating 81O
  font 840
  location 81O
  on the glass 820
  using 81O
text widgets 275B, 1309R
  appending text to 1232R
  changing selected text 1249R
  converting
    character offsets to column/line form 1277R
    line/column positions to character
      offsets 1277R
delete events 218U
determining
  if all events are being returned 1276R
  if text widget is editable 1276R
editable 1310R
  making editable after creation 1235R
events returned by 1231R, 1309R, 1314R
focus events 218U
keyboard focus events 1311R
properties 216U
returning
  line number of top line in viewport 1277R
  number of characters 1277R
  offsets of text selection 1277R
  selected text 1249R
selection events 219U
setting
  text selection 1244R
  top line 1244R
setting attributes 216U
setting editable state 216U
setting height 216U

Index T ... T

Using IDL, Building IDL Applications, Object Graphics, Reference Guide Index
setting initial displays 216U
setting keyboard focus to 1238R
setting scrolling 217U
setting width 217U
setting word wrapping 217U
string insert events 219U
suppressing newline characters 1240R
text insert events 218U
using 165U
TEXT_ALL_EVENTS keyword 1276R
TEXT_AXES keyword 1380R
TEXT_COLOR keyword 275O
TEXT_EDITABLE keyword 1276R
TEXT_NUMBER keyword 1277R
TEXT_OFFSET_TO_XY keyword 1277R
TEXT_SELECT keyword 1277R
TEXT_TOP_LINE keyword 1277R
TEXT_XY_TO_OFFSET keyword 1277R
TEXTANGLE keyword 673R
TEXTPOS keyword 211O
texture mapping 107O
TEXTURE_COORD keyword 331O, 366O
TEXTURE_INTERP keyword 331O, 366O
TEXTURE_MAP keyword 331O, 367O
THICK keyword 2120, 2430, 313O, 321O, 332O,
THICK system variable field 52R, 55R
THICKNESS field 192O
THICKNESS keyword 259O
thickness of characters 1380R
THIN function 1118R
thinning images 1118R
thin-plate-spline surface 748R
THREED keyword 244O
THREED procedure 1119R
THREED procedure 1119R
three-dimensional
three-dimensional
graphics 334U
transformations
array transforms 1186R
coordinates 314R, 380R
duplicating SURFACE transforms 1095R
implementing transforms 1109R
matrices 335U
PLOT and CONTOUR 343U
plotting 314R, 380R
scaling 964R, 985R
specifying orientation 350R
T3D keyword 51R
THRESH keyword 1061R, 1064R
THRESHOLD keyword 140R, 340R
throw, C++ language 255R
tick labels 80O
tick marks 302U
annotation 55R, 109R
data values for 56R, 109R
getting values of 109R
intervals 56R, 109R
length 52R, 108R
length on individual axes 55R, 108R
linestyles 302U, 102R
minor 53R, 103R
string labels for 55R
styles 53R
suppressing 56R, 109R
TICKDIR keyword 2120
TICKFORMAT keyword 2120, 244O
TICKFORMAT system variable field 55R
TICKFRM_DATA keyword 2120, 244O
TICKINTERVAL keyword 254O
TICKLEN keyword 302U, 2130, 2440, 2540, 108R
TICKLEN system variable field 52R, 55R
TICKNAME system variable field 55R
TICKS system variable field 56R
TICKTEXT keyword 2130, 244O
TICKV system variable field 56R
TICKVALUES keyword 2130, 244O
TIFF files
reading 921R
standard file format I/O routines 228B
writing 1337R
TIFF_DUMP, see Obsolete Routines
TIFF_READ, see Obsolete Routines
TIFF_WRITE, see Obsolete Routines
TILT keyword 509R
time
converting from string to binary 219R
returning current 1105R
TIME_TEST procedure 1120R
timed demo mode 414R
timer events (for widgets) 302B
TIMER keyword 302B, 1247R
TIMES keyword 140R
time-series analysis 466U
autocorrelation 187R
autocovariance 187R
autoregressive modeling 1147R, 1149R
cross correlation 240R
Index T ... T

Using IDL, Building IDL Applications, Object Graphics, Reference Guide

Index

cross covariance 240R
forward differencing 1148R
Title Bar base property 208U
Title base property 207U
Title droplist property 224U
Title slider property 223U
TITLE system variable field 52R, 56R
titles 74O
titles, multiline 289U
TLB_FRAME_ATTR keyword 1215R
TLB_GET_OFFSET keyword 1247R
TLB_GET_SIZE keyword 1248R
TLB_KILL_REQUEST_EVENTS keyword 1215R, 1248R
TM_TEST function 448U, 1121R
t:mean test 1121R
TMP keyword 478R
TNAME keyword 1023R
TO_CYLIN keyword 338R
TO_DATA keyword 310R
TO_DEVICE keyword 310R
TO_NORMAL keyword 310R
TO_POLAR keyword 339R
TO_RECT keyword 339R
TO_SPHERE keyword 339R
toggle buttons 1227R
TOL keyword 602R, 679R
TOLF keyword 232R, 768R
TOLMIN keyword 232R, 768R
TOLX keyword 232R, 768R
Tomographic reconstructions 963R
tools
IDL GUIBuilder 164U
IDLDE for Motif 41U, 72U
IDLDE for Windows 85U
top margin, setting 53R
TOP_ID keyword 1041R
TOP_STRETCH keyword 3040, 3060
top-level base 1205R
TOTAL function 1123R
TQLI, see Obsolete Routines
TRACE executive command See commands
TRACE function 460U, 1125R
traceback information 148B
displaying 540R
returning 538R
TRACEBACK keyword 540R, 746R
TRACK keyword 342R, 391R, 1358R
TrackBall
class 310
Init method 442O
Reset method 443O
Update method 444O
TrackBall object 440O
trackball_define.pro (example file) 55O
transformations
rotation 49O
TRANS keyword 851R
TRANSFER_COUNT keyword 933R, 1344R
TRANFORM keyword 292O
transformation matrices 335U, 51R
transformations 500, 510
coordinate 510
example 530
model objects 480
rotation 480
scaling 480
translation 480, 490
transforms
Fourier 400U, 474R
Helbert 411U
wavelet 413U
TRANSLATE keyword 337U, 1110R
Translate method 295O
translate method 49O
translation 335U, 480, 490, 1109R
TRANSLATION keyword 140R, 281R
translation tables, bypassing 117R
transparency of voxels 124O
TRANSPARENT keyword 3570, 3930, 828R, 911R, 1329R
<table>
<thead>
<tr>
<th>Index</th>
<th>Index-67</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRANSPOSE function 430U, 1126R</td>
<td>TT_FONT keyword 141R, 1017R</td>
</tr>
<tr>
<td>transposing arrays 1126R</td>
<td>TTY keyword 141R</td>
</tr>
<tr>
<td>Transverse Mercator map (UTM) projection 369U, 725R</td>
<td>TUPLE_CNT keyword 366R</td>
</tr>
<tr>
<td>TRANSVERSE_MERCATOR keyword 725R</td>
<td>TUPLE_PERIOD keyword 366R</td>
</tr>
<tr>
<td>TRED2, see Obsolete Routines</td>
<td>TV procedure 381U, 1153R</td>
</tr>
<tr>
<td>tree 360</td>
<td>TVCRS procedure 395U, 1157R</td>
</tr>
<tr>
<td>TREE_EXAMPLE example routine 248B</td>
<td>TVDELETE, see Obsolete Routines</td>
</tr>
<tr>
<td>trees 242B</td>
<td>TVLCT procedure 388U, 1159R</td>
</tr>
<tr>
<td>binary 248B</td>
<td>TVRD function 385U, 395U, 1161R</td>
</tr>
<tr>
<td>trend analysis 467U</td>
<td>TVRDC, see Obsolete Routines</td>
</tr>
<tr>
<td>TRI_SURF function 1128R</td>
<td>TVSCL procedure 381U, 1164R</td>
</tr>
<tr>
<td>TRIAL keyword 681R</td>
<td>TVSET, see Obsolete Routines</td>
</tr>
<tr>
<td>Triangle Fans 1460</td>
<td>TVSHOW, see Obsolete Routines</td>
</tr>
<tr>
<td>Triangle strips 1470</td>
<td>TVWINDOW, see Obsolete Routines</td>
</tr>
<tr>
<td>TRIANGULATE keyword 719R</td>
<td>TWO_PASS_QUANTIZE keyword 908R</td>
</tr>
<tr>
<td>TRIANGULATE procedure 446U, 1131R</td>
<td>TWO_SIDED keyword 4100</td>
</tr>
<tr>
<td>triangulation 1131R, 1134R</td>
<td>two-dimensional Gaussian fit 508R</td>
</tr>
<tr>
<td>spherical 1131R</td>
<td>two-tailed hypothesis tests 447U</td>
</tr>
<tr>
<td>TRIANGULATION keyword 307R</td>
<td>Type button property 214U</td>
</tr>
<tr>
<td>TRIDAG, see Obsolete Routines</td>
<td>type conversion</td>
</tr>
<tr>
<td>tridiagonal array or matrix 1140R, 1142R, 1143R</td>
<td>to 64-bit integer 689R</td>
</tr>
<tr>
<td>TRIGRID function 446U, 1134R</td>
<td>to byte 234R</td>
</tr>
<tr>
<td>trilinear interpolation 573R</td>
<td>to complex 285R, 396R</td>
</tr>
<tr>
<td>TRIM keyword 1071R</td>
<td>to double-precision 442R</td>
</tr>
<tr>
<td>trimming strings 1086R</td>
<td>to integer 484R</td>
</tr>
<tr>
<td>TRIOL procedure 444U, 1140R</td>
<td>to longword 688R</td>
</tr>
<tr>
<td>TRIRED procedure 444U, 1142R</td>
<td>to single-precision, floating-point 486R</td>
</tr>
<tr>
<td>TRISOL function 460U, 1143R</td>
<td>to string 1077R</td>
</tr>
<tr>
<td>TRNLOG function 1145R</td>
<td>to unsigned 64-bit integer 1174R</td>
</tr>
<tr>
<td>TRUE keyword 908R, 1154R, 1162R, 1325R</td>
<td>to unsigned integer 1167R</td>
</tr>
<tr>
<td>true map scale, setting 360U</td>
<td>to unsigned longword 1173R</td>
</tr>
<tr>
<td>TRUE_COLOR keyword 141R</td>
<td>TYPE system variable field 56R</td>
</tr>
<tr>
<td>true-color</td>
<td>type-ahead buffer 514R</td>
</tr>
<tr>
<td>displays 392U</td>
<td>typestyle 84O</td>
</tr>
<tr>
<td>images</td>
<td>typographical conventions 27O</td>
</tr>
<tr>
<td>converting to pseudo-color 279R</td>
<td>U</td>
</tr>
<tr>
<td>displaying 1154R</td>
<td>UDF_BLOCK keyword 790R</td>
</tr>
<tr>
<td>PostScript 158R</td>
<td>UI_VALUE keyword 247R</td>
</tr>
<tr>
<td>reading 1162R</td>
<td>UINDGEN function 1166R</td>
</tr>
<tr>
<td>visuals 120R</td>
<td>UINT function 1167R</td>
</tr>
<tr>
<td>TrueType 74R, 136R</td>
<td>UINT keyword 562R</td>
</tr>
<tr>
<td>TrueType fonts 64O, 65R, 86R</td>
<td>UINTARR function 1168R</td>
</tr>
<tr>
<td>TRUNCATE_ON_CLOSE keyword 790R</td>
<td>UL_VALUE keyword 247R</td>
</tr>
<tr>
<td>TS_COEF function 469U, 1147R</td>
<td>UL64 keyword 562R</td>
</tr>
<tr>
<td>TS_DIFF function 469U, 1148R</td>
<td>UL64_VALUE keyword 247R</td>
</tr>
</tbody>
</table>
UL64INDGEN function 1169R
ULINDGEN function 1170R
ULON64ARR function 1171R
ULONG function 1173R
ULONG keyword 562R
ULONG64 function 1174R
unconstrained minimizer 462U
underflow errors 151B
UNEQUAL keyword 1122R
unformatted binary data 933R, 1344R
unformatted I/O 158B, 191B
uniform random deviates 891R
uniformly-distributed random numbers 890R
UNIQ function 1175R
Unix environment variables 519R
UNIX, OS-specific file I/O information 213B
unmapping widgets 1209R
UNRESOLVED keyword 976R
unsharp masking 384U
unsigned 64-bit integer arrays 1169R
unsigned arrays
  longword 1170R
  data type, converting to 1174R
unsigned data type
  integer 12B
  long 12B
unsigned integer arrays 1166R
  data type, converting to 1167R
UNSIGNED keyword 924R
unsigned longword arrays 1172R
  data type, converting to 1173R
UP keyword 333R
UPDATE keyword 309R, 351R, 1249R, 1279R
Update method 444O
UPDATE_DATA keyword 620R
updating files (OPENU procedure) 783R
UPDIR keyword 386O
upper margin, setting 53R
UPPER_ONLY keyword 1093R
uppercase, converting strings to 1089R
USA keyword 709R, 728R
USE_CURRENT keyword 1362R
USE_TABLE_SELECT keyword 1249R, 1279R
USE_TEXT_COLOR keyword 213O
USE_TRIANGLES keyword 367O
USE_ZVALUE keyword 321O
USEDOUBLES keyword 919R
USELONGS keyword 919R
user interface compound widgets 276B
user values
  for widgets 286B
  setting 1245R
USER_FONT keyword 141R
user-defined plotting symbols 1177R
USERSYM procedure 296U, 1177R
using external modules 245R
UTM (Transverse Mercator) map projection 369U, 725R
UX keyword 977R
UY keyword 978R
V
VALID_DATA keyword 406O
VALID_ID keyword 1279R
value
  parameters passed by 120B
  widgets 284B
VALUES keyword 434R, 453R, 994R
VARIABLE keyword 790R
VariableWatch Window 145U
VARIABLE_WTS keyword 274R
variables 264U, 21B
  associated 204R
<table>
<thead>
<tr>
<th>Index</th>
<th>Index-69</th>
</tr>
</thead>
<tbody>
<tr>
<td>attributes of</td>
<td>264U, 21B</td>
</tr>
<tr>
<td>deleting</td>
<td>413R</td>
</tr>
<tr>
<td>derived</td>
<td>472U</td>
</tr>
<tr>
<td>disappearing</td>
<td>142B</td>
</tr>
<tr>
<td>displaying current</td>
<td>145U</td>
</tr>
<tr>
<td>interactive editing tool (XVEDIT procedure)</td>
<td>1378R</td>
</tr>
<tr>
<td>named</td>
<td>181O, 184R, 185R</td>
</tr>
<tr>
<td>names of</td>
<td>265U, 21B</td>
</tr>
<tr>
<td>reading display images into (TVRD function)</td>
<td>1161R</td>
</tr>
<tr>
<td>returning information on</td>
<td>537R</td>
</tr>
<tr>
<td>saving</td>
<td>983R</td>
</tr>
<tr>
<td>standardized</td>
<td>472U</td>
</tr>
<tr>
<td>system</td>
<td>266U, 22B</td>
</tr>
<tr>
<td>temporary</td>
<td>1117R</td>
</tr>
<tr>
<td>VARIABLES keyword</td>
<td>976R, 983R</td>
</tr>
<tr>
<td>variance</td>
<td>498R, 755R</td>
</tr>
<tr>
<td>VARIANCE function</td>
<td>1179R</td>
</tr>
<tr>
<td>VARIANCE keyword</td>
<td>1099R</td>
</tr>
<tr>
<td>VARIANCES keyword</td>
<td>797R</td>
</tr>
<tr>
<td>VAX_CALL_EXT routine</td>
<td>1406</td>
</tr>
<tr>
<td>VAX_FLOAT function</td>
<td>1180R</td>
</tr>
<tr>
<td>VAX_FLOAT keyword</td>
<td>247R, 786R, 1406</td>
</tr>
<tr>
<td>VAXTOF keyword</td>
<td>236R</td>
</tr>
<tr>
<td>VECTOR keyword</td>
<td>1342R</td>
</tr>
<tr>
<td>vector-drawn fonts</td>
<td>65R, 89R</td>
</tr>
<tr>
<td>! character</td>
<td>81R</td>
</tr>
<tr>
<td>displaying</td>
<td>1017R</td>
</tr>
<tr>
<td>editing (EFONT procedure)</td>
<td>443R</td>
</tr>
<tr>
<td>special characters</td>
<td>68R</td>
</tr>
<tr>
<td>vectors</td>
<td>drawing arrowheads</td>
</tr>
<tr>
<td>subscripting</td>
<td>64B</td>
</tr>
<tr>
<td>VEL procedure</td>
<td>1182R</td>
</tr>
<tr>
<td>velocity field, plotting</td>
<td>488R, 1182R, 1184R</td>
</tr>
<tr>
<td>VELOVect procedure</td>
<td>1184R</td>
</tr>
<tr>
<td>VERSION keyword</td>
<td>1279R</td>
</tr>
<tr>
<td>VERT_COLORS keyword</td>
<td>321O, 3320, 3400, 3670</td>
</tr>
<tr>
<td>VERT_T3D function</td>
<td>1186R</td>
</tr>
<tr>
<td>VERTICAL keyword</td>
<td>378R, 388R, 1295R</td>
</tr>
<tr>
<td>vertical slider, See slider widgets</td>
<td></td>
</tr>
<tr>
<td>VERTICAL_ALIGNMENT keyword</td>
<td>3860</td>
</tr>
<tr>
<td>view area</td>
<td>420</td>
</tr>
<tr>
<td>view menu (online help viewer)</td>
<td>246U</td>
</tr>
<tr>
<td>view object</td>
<td>280, 3880</td>
</tr>
<tr>
<td>view objects</td>
<td>370</td>
</tr>
<tr>
<td>view volume</td>
<td>460</td>
</tr>
<tr>
<td>finding an appropriate</td>
<td>470</td>
</tr>
<tr>
<td>viewgroup object</td>
<td>280, 3960</td>
</tr>
<tr>
<td>viewgroup objects</td>
<td>370</td>
</tr>
<tr>
<td>viewplane rectangle</td>
<td>460</td>
</tr>
<tr>
<td>VIEWPLANE_RECT</td>
<td>460</td>
</tr>
<tr>
<td>route for determining</td>
<td>470</td>
</tr>
<tr>
<td>viewport</td>
<td>420</td>
</tr>
<tr>
<td>Viewport Columns table property</td>
<td>236U</td>
</tr>
<tr>
<td>Viewport Rows table property</td>
<td>237U</td>
</tr>
<tr>
<td>VIEWPOINT_EVENTS keyword</td>
<td>1258R</td>
</tr>
<tr>
<td>virtual memory</td>
<td>124B, 135B</td>
</tr>
<tr>
<td>minimizing</td>
<td>137B</td>
</tr>
<tr>
<td>running out of</td>
<td>136B</td>
</tr>
<tr>
<td>system parameters</td>
<td>138B</td>
</tr>
<tr>
<td>Visible base property</td>
<td>208U</td>
</tr>
<tr>
<td>Visible property</td>
<td>208U</td>
</tr>
<tr>
<td>VISUALIZATION_IN keyword</td>
<td>624R, 642R, 647R, 658R, 673R</td>
</tr>
<tr>
<td>VISUALIZATION_PROPERTIES keyword</td>
<td>663R</td>
</tr>
<tr>
<td>VMS</td>
<td></td>
</tr>
<tr>
<td>Open VMS</td>
<td></td>
</tr>
<tr>
<td>virtual memory performance</td>
<td></td>
</tr>
<tr>
<td>VMS file I/O information</td>
<td>217B</td>
</tr>
<tr>
<td>VMS logical name</td>
<td>412R</td>
</tr>
<tr>
<td>VMS logical name tables</td>
<td>1145R</td>
</tr>
<tr>
<td>VMS logical tables</td>
<td>1146R</td>
</tr>
<tr>
<td>VMS text libraries</td>
<td>462R</td>
</tr>
<tr>
<td>VMSCODE, see Obsolete Routines</td>
<td></td>
</tr>
<tr>
<td>VOIGT function</td>
<td>1188R</td>
</tr>
<tr>
<td>volume object</td>
<td>300, 4020</td>
</tr>
<tr>
<td>volume objects</td>
<td>1220</td>
</tr>
<tr>
<td>attributes</td>
<td>1240</td>
</tr>
<tr>
<td>color values</td>
<td>1250</td>
</tr>
<tr>
<td>composting</td>
<td>1260</td>
</tr>
<tr>
<td>creating</td>
<td>1220</td>
</tr>
<tr>
<td>interpolating values</td>
<td>1270</td>
</tr>
<tr>
<td>lighting</td>
<td>1260</td>
</tr>
<tr>
<td>opacity table</td>
<td>1240</td>
</tr>
<tr>
<td>rendering speed</td>
<td>1270</td>
</tr>
<tr>
<td>using</td>
<td>1220</td>
</tr>
<tr>
<td>zbuffering</td>
<td>1260</td>
</tr>
<tr>
<td>volume slices</td>
<td>1027R</td>
</tr>
<tr>
<td>VOLUME_SELECT keyword</td>
<td>4110</td>
</tr>
<tr>
<td>volumes</td>
<td></td>
</tr>
<tr>
<td>extracting slices</td>
<td>469R</td>
</tr>
<tr>
<td>rendering</td>
<td>850R</td>
</tr>
<tr>
<td>searching for objects</td>
<td>989R</td>
</tr>
</tbody>
</table>
visualizing 351U, 832R, 850R, 1010R, 1192R
volumetric reconstruction 939R
VORONOI procedure 446U, 1190R
voxel rendering 1192R
voxel values (volume object) 1220
VOXEL_PROJ function 1192R
VRML object 320, 414O
VRML objects 141O
creating 141O
VT200 keyword 1001R
VT240 keyword 141R
VT240 terminal 168R
VT330 terminal 168R
VT340 keyword 141R
VT340 terminal 168R
W
WAIT keyword 333R
WAIT procedure 130B, 1196R
WARM keyword 1202R
WARP_TRI function 1197R
warping
images 820R
to maps 715R, 718R
polynomial 820R
using the Z-buffer 828R
Wavefront Advanced Data Visualizer 926R, 1342R
Wavefront files
reading 926R
writing 1342R
wavelet transform 413U, 1348R
DELETE procedure 144R, 1199R
weather fronts, plotting 1201R
WEIGHT keyword 1099R
WEIGHTS keyword 283R, 679R
WEOF procedure 1200R
WF_DRAW procedure 1201R
WHERE function 1203R
while statement 1088
whitespace, removing from strings 77B, 1074R, 1086R
WIDGET, see Obsolete Routines
Widget Browser 169U, 191U
WIDGET_BASE function 306B, 1205R
WIDGET_BUTTON function 1222R
WIDGET_CONTROL procedure 300B, 301B, 1230R
WIDGET_DRAW function 1252R
WIDGET_DROPLIST function 1262R
WIDGET_EVENT function 1268R
WIDGET_INFO function 1271R
WIDGET_KILL_REQUEST event 1215R
WIDGET_LABEL function 1280R
WIDGET_LIST function 1285R
WIDGET_MESSAGE, see Obsolete Routines
WIDGET_SLIDER function 1291R
WIDGET_TABLE function 1298R
WIDGET_TEXT function 1309R
widgets 202U, 210U, 272B
3D orientation 276B
aligning (ALIGN_XXX keywords) 1205R
animation 341R
annotation 196R
application errors 282B
tips 315B
background tasks (TIMER keyword) 1247R
base 273B, 1205R
base focus events 210U
base, alignment 203U
base, allow moving 204U
base, allowing closing 203U
base, displaying titlebars 208U
base, floating 204U
base, grid layouts 204U
base, kill request events 210U
base, layouts 205U
base, menus, using system 207U
base, modal 206U
base, resizing 206U
base, rows and columns 202U
base, scroll area size 209U
base, scrolling 206U
base, setting properties 202U
base, spacing of contained widgets 207U
base, spacing of widgets in 208U, 209U
base, titles 207U
base, visibility 208U
bases, using 165U
blocking (MODAL keyword) 1365R
Browser 169U
button, press events 215U
button, release events 214U
button, setting properties 212U
buttons 273B, 1222R
bitmap labels 928R
groups 354R
release events 1224R
buttons, adding menus 173U
buttons, displaying bitmaps 213U
buttons, labels 214U
buttons, using 165U
callbacks 1209R, 1211R
changing values 301B
checkboxes, using 165U
color
index 358R, 387R
resources 1213R
selection 360R
common blocks and 312B
common events 199U
template for creating 389R
compound, adding 188U
compound, example 188U
compound, handling events for 199U
controlling 300B
controlling visibility 191U
creating in IDL GUIBuilder 164U
creating with IDL GUIBuilder 150U
default font for 1233R
destroy events 200U
destroying 1234R
determining if widgets are realized
(ACTIVE keyword) 1271R
(REALIZED keyword) 1275R
disabling and enabling screen updates (UPDATE keyword) 1249R
displaying 191U
draw 273B, 294B, 1252R
draw area properties 229U
draw area, color model 229U
draw events 232U
draw, attributes 229U
draw, backing store 231U
draw, changing view events 233U
draw, colors used in 229U
draw, graphic type 230U
draw, mouse events 232U
draw, mouse motion events 233U
draw, render type 230U
draw, scrolling 231U
draw, scrolling area 231U
draw, viewport move events 233U
draw, viewport move events 233U
droplist 274B, 1262R
droplist attributes 224U
droplist events 224U
droplist properties 224U
droplist, select events 225U
droplist, title 224U
droplists, initial value 224U
droplists, using 165U
dynamic resizing 306B
enabled or disabled state 198U
events 287B, 1268R
CLEAR_EVENTS keyword 1232R
draw, using 166U
elements 224U
exclusive buttons 1208R
explicit size 305B
field 368R
finding screen size 308B
form 371R
frames, using 197U
geometry 305B
 getting user values 1236R
height 199U
help buttons 1223R
hiding and showing 1246R
hierarchies 300B
horizontal size, changing 1242R, 1250R
hourglass cursor 301B
iconifying 1238R
interrupting the event loop 311B
invalid IDs 1232R, 1268R
killing hierarchies 300B
label 274B, 1280R
label, setting properties 221U
labels, using 165U
lifecycle 281B
list 274B, 1285R
listbox attributes 226U
listbox events 227U
listbox properties 226U
listbox, height 226U
listbox, initial value 226U
listbox, multiple selections 226U
listbox, selection events 227U
listbox, using 165U
listbox, width 227U
location 306B
main event loop for 1363R
managing the state of applications 312B
mapping 1209R
  mapping and unmapping 1239R
menu bars 1206R, 1209R
menus 295B
message 274B
message dialog box 423R
modal 423R
modal dialogs 206U
naming 196U
natural size 305B
non-exclusive buttons 1211R
portability 316B
positioning 198U, 1218R, 1219R
post creation events 201U
preventing layout flicker 308B
properties for IDL GUIBuilder 166U
pulldown menu 382R
pulldown menus 298B
  separators 1225R
radio buttons, using 165U
realize events 200U
realizing 1241R
  hierarchies 300B
region of interest 362R
registered 1374R
registering with XMANAGER 1363R
resetting all widgets 1241R
resizing (DYNAMIC_RESIZE keyword) 1222R, 1262R, 1280R
restarting after an error 282B
retreiving values 301B
returning
  children of 1271R
  information about 1271R
  name of event handler procedure 1273R
  parent of 1275R
  siblings of 1275R
  size of (GEOMETRY keyword) 1273R
  tracking event status 1278R
  type of 1275R, 1278R
  validity of 1279R
sending event to (SEND_EVENT keyword) 1242R
sensitizing 301B
setting button events 215U
setting buttons 1242R
setting label attributes 221U
showing and hiding 1246R
size 306B
  changing
    horizontal 1242R, 1250R
    vertical 1242R, 1250R
  dynamic resizing 306B
  explicit 305B
  natural 305B
  sizing 304B
sizing, default or explicit 197U
slider 274B, 377R, 1291R
  attributes 222U
  properties 222U
  setting events 223U
 , displayed values 222U
  , initial position 222U
  , maximum value 222U
  , minimum value 222U
  , setting events 223U
  , title 223U
  , using 165U
  space between children 1214R
  table 275B, 1298R
  attributes 234U
  events 237U
  properties 234U
  , alignment 234U
  , cell select events 237U
  , column labels 234U
  , column width events 238U
  , data transfer to 235U
  , editing 235U
  , focus events 238U
  , heading display 234U
  , height 235U
  , insert character events 239U
  , insert string events 239U
  , invalid data events 239U
  , row labels 236U
  , scroll height 237U
  , scroll width 236U
  , scrolling 236U
  , sizing columns 235U
  , text delete events 238U
  , text selection events 240U
  , using 166U
  , width 235U
  template for creating 1369R
Write method 2030
WRITE_32 keyword 1333R
WRITE_BMP procedure 1320R
WRITE_GIF procedure 1322R
WRITE_JPEG procedure 1324R
WRITE_NRIF procedure 1327R
WRITE_PICT procedure 1328R
WRITE_PNG function 1331R
WRITE_PPM procedure 1335R
WRITE_SPR procedure 1337R
WRITE_SRF procedure 1342R
WRITE_SYLK function 1344R
WRITE_TIFF procedure 1346R
WRITE_WAVE procedure 1348R
WRITEU procedure 1352R
writing
a compound widget 316B
BMP files 1320R
files (OPENW procedure) 783R
GIF files 1322R
JPEG files 1324R
NRIF files 1327R
PGM files 1331R
PICT files 1328R
PPM files 1335R
SRF files 1342R
TIFF files 1346R
wave files 1352R
WSET procedure 144R, 1346R
WSHOW procedure 144R, 1347R
WSIZE keyword 848R
WTN function 423U, 1348R

X
X keyword 300R
X Offset common property 198U
X Pad base property 208U
X resources 66U
  widget colors 1213R
X Scroll base property 209U
X Scroll draw area property 231U
X Size common property 198U
X vs. Y Plots 287U
X Windows
  bitmap files, reading 928R
  Dump files, reading 930R
  fonts 1356R
  X Windows device 171R
  DirectColor visual 121R
  PseudoColor visual 131R
  StaticColor visual 139R
  StaticGray visual 139R
  TrueColor visual 141R
  visuals 171R
X_BITMAP_EXTRA keyword 1226R, 1250R
X_CH_SIZE system variable field 48R
X_PX_CM system variable field 48R
X_SIZE system variable field 48R
X_TICKNAME keyword 617R, 648R, 654R, 670R
X_VSIZE system variable field 48R
X_ZSIZE keyword 391R
X0 keyword 408R, 436R
X11 Bitmap, standard file format I/O routines 228B
XANIMATE, see Obsolete Routines
XAXIS keyword 207R
XAXIS_IN keyword 648R, 653R, 670R
XBACKREGISTER, see Obsolete Routines
XBME_EDIT procedure 1228R, 1352R
XCHARSIZE keyword 100R
XDICE procedure 323B
XDISPLAYFILE procedure 1354R
XDL, see Obsolete Routines
XDR 197B
XDR files 160B, 983R
XDR format (floating point values) 235R
XDR keyword 786R, 983R
XDRTOD keyword 237R
XDRTOF keyword 237R
XFONT function 1356R
XGRID keyword 587R, 1129R, 1136R
XGRIDSTYLE keyword 102R
XINDEPENDENT keyword 616R, 669R
XINTERANIMATE procedure 1357R
XLOADCT procedure 390U, 1361R
XMANAGER procedure 289B, 292B, 315B, 1268R, 1363R
XMANAGERTOOL, see Obsolete Routines
YP0 keyword 1054R
YPAD keyword 356R, 1217R
YPN_1 keyword 1054R
YPOS keyword 1318R
YRESOL keyword 1340R
YSTART keyword 719R, 844R
YSTYLE keyword 106R
YTHICK keyword 302U, 107R
YTICK_GET keyword 109R
YTICKFORMAT keyword 107R
YTICKLEN keyword 108R
YTICKNAME keyword 303U, 109R
YTICKS keyword 303U, 109R
YTICKV keyword 303U, 109R
YTITLE keyword 109R, 211R, 1119R
YVALUES keyword 588R, 1129R
YVISIBLE keyword 1042R
YZ_Plane keyword 804R
YZEACH keyword 338U, 1110R
YZSTYLE keyword 804R

Z
Z keyword 110R, 300R
ZAPFCHANTERY keyword 143R
ZAPFDINGBATS keyword 143R
ZAXIS keyword 207R, 307R, 1093R
ZAXIS_PROPERTIES keyword 664R
Z-buffer
  closing 117R
  using with POLYFILL 827R
  using with POLYSHADE 832R
  warping images to polygons 828R
Z-buffer device 178R
ZBUFFER keyword 4120, 1194R
ZBUFFER_DATA keyword 2210
zbuffering, volume objects 1260
ZCHARSIZE keyword 100R
ZCLIP keyword 3940
ZCOORD_CONV keyword 2140, 2450, 2550, 2670, 2760, 2830, 3220, 3320, 3410, 3680, 3870.