Indigo Magic™
User Interface Guidelines

Document Number 007-2167-001

CONTRIBUTORS

Written by Jackie Neider, Deb Galdes, John C. Stearns, and Arthur Evans
Illustrated by John C. Stearns, Arthur Evans, Delle Maxwell, and Doug O'Morain
Edited by Christina Cary
Style Card designed and illustrated by Kay Maitz
Principal architects of the Silicon Graphics Style: Deb Galdes, Mike Mohageg, Michael Portuesi, Rob Myers, and Betsy Zeller
Special thanks to Donna Davilla, Debbie Myers, Dave Ciemiewicz, Steve Yohanan, and Kim Rachmeler
Other contributions by Dave Story, Eva Manolis, Susan Dahlberg, Doug Young, Josie Wernecke, Susan Angebranndt, Roger Powell, Dave Mott, Tom Davis, David Curley, Ken Jones, Richard Wright, Ron Edmark, Victor Riley, Ashmeet Sidana, Ken Kershner, Joel Tesler, Bob Brown, Dan Miley, Greg Ferguson, Jerry Granucci, Ed Immenschuh, Grace Pariante, Delle Maxwell, Rebecca Underwood, James Raby, Pete Sullivan, Mike Chow, Tony Wong, Jamie Schein, Mike Keeler, Anna Wichansky, Beth Fryer, Robert Reimann

Indigo Magic™ User Interface Guidelines
Document Number 007-2167-001

Contents

About This Guide xxi
What This Guide Contains xxii
Part One: Integrating With the Indigo Magic Desktop xxii
Part Two: Interface Components xxiii
Appendix xxiv
What You Should Know Before Reading This Guide xxiv
Suggestions for Further Reading xxiv
Conventions Used in This Guide xxv
Style Guidelines xxv
Font Conventions xxvi

Part One Integrating With the Indigo Magic Desktop

1. Overview of the Indigo Magic Desktop 3
Overview of the Desktop 4
How Users Interact With Desktop Icons 6
Mouse and Keyboard Hardware 10

2. Icons 13
Designing the Appearance of Icons 14
General Icon Design: Components, Size, and Colors 15
Icon Components 16
Icon Size 19
Icon Colors 19
Icon Orientation 22
Application Icon Design 24
File Icon Design 26
Icon Appearance Design Guidelines 28
Defining the Behavior of Icons With FTRs 29
Icon Behavior Guidelines 31
Making Application Icons Accessible 32
Putting Icons Into the Icon Catalog 32
Naming and Locating Executables for the Find an Icon Tool 34
Application Icon Accessibility Guidelines 35

3. Windows in the Indigo Magic Environment 37
The Indigo Magic Look: Graphic Features and Schemes 38
Enhanced Graphics in the Indigo Magic Look 38
Schemes for Colors and Fonts 41
Indigo Magic Look Guidelines 42

Application Window Categories and Characteristics 42
Application Window Categories 42
Application Models 43
"Single Document, One Primary" Application Model 43
"Single Document, Multiple Primaries" Application Model 43
"Multiple Document, Visible Main" Application Model 44
"Multiple Document, No Visible Main" Application Model 44
Window Decorations and the Window Menu 44
Window Title Bar 47
Rules for Labeling the Title Bar in Main Primary Windows 48
Rules for Labeling the Title Bar in Windows Other Than Main 49
Window Size 50
Window Placement 52
Application Window Characteristic Guidelines 53
Keyboard Focus Across Windows 56
Single-Action Pointer Grab Model 56
Multiple-Action Pointer Grab Model 59
Guidelines for Keyboard Focus Across Windows 60
Minimized Windows 61
Choosing an Image for Your Minimized Window 61
Labeling a Minimized Window 63
Processing While Minimized 64
Using the Minimized Window to Show Status 64
Minimized Window Guidelines 65
Desks 67
Desks Guidelines 68
Session Management 68
Session Management Guidelines 70

4. Indigo Magic Desktop Services 73
Software Installation 74
Software Installation Guideline 75

Online Help 76
Providing Help Using SGIHelp 76
Types of Online Help 77
Context-Sensitive Information 79
Overview Information 80
Task-Oriented Information 80
Index of Help Topics 80
Keyboard Shortcut Information 81
Product Information 81
Providing Help through a Help Menu 82
Providing Help Through a Help Button 83
Guidelines for Designing Online Help 85
Writing Online Help Content for SGIHelp 88
Learning About SGIHelp 88
Creating Help Cards 89
Writing Context-Sensitive Help 89
Writing Overview Information 90
Writing Task Information 90
Writing Index Information 91
Writing Keyboard Shortcut Information 91
Writing for Windows With Help Buttons 92
Guidelines for Creating SGIHelp Content 92
Online Documentation 94
Desktop Variables 95
Scheme Setting 95
Auto Window Placement Setting 95
Language Setting 95
Mouse Double-Click Speed Setting 96
Editor Preference Setting 96
Desktop Variables Guidelines 99
File Alteration Monitor (FAM) 100
File Monitoring Guideline 100

5. **Data Exchange on the Indigo Magic Desktop** 101
Supporting the Clipboard Transfer Model 102
Supporting the Primary Transfer Model 104
Data Types Supported for Inter-Application Transfer 108
Data Exchange Guidelines 109

Part Two Interface Components

6. **Application Windows** 113
Application Models 114
Window Types 114
Standard Application Models 115
"Single Document, One Primary" Application Model 116
"Single Document, Multiple Primaries" Application Model 117
"Multiple Document, Visible Main" Application Model 118
"Multiple Document, No Visible Main" Application Model 120
Application Model Guidelines 120
Main and Co-Primary Windows 121
Menu Bars in Primary Windows 123
Scrollable Work Areas in Primary Windows 124
Control Areas in Primary Windows 124
Status Areas in Primary Windows 126
Splitting Primary Windows Into Panes 126
Popup Menus in Primary Windows 126
Primary Window Guidelines 127
Support Windows 128
General Support Window Design 128
A Specific Standard Support Window: The Indigo Magic Color Chooser 131
Support Window Guidelines 133
Pointer Behavior in a Window 133
Pointer Behavior Guidelines 134

7. Focus, Selection, and Drag and Drop 135
Keyboard Focus and Navigation 135
Keyboard Focus Policy and Navigation Within a Window 136
Keyboard Navigation 138
Mouse Navigation 140
Keyboard Focus and Navigation Guidelines 141
Selection 142
Selection Models—What Can Be Selected and How To Select It 142
Highlighting a Selection 145
Multiple Collections in One Application Window 145
Selection Guidelines 146
Drag and Drop 147
Two Models of Drag and Drop 147
Drag and Drop for Non-Text Objects 148
Drag and Drop for Text 149
Pointers for Drag Operations 149
Drag and Drop Guidelines 150

8. Menus 151
Types of Menus 151
Menu Traversal and Activation 154
Menu Traversal and Activation Guidelines 156

The Menu Bar and Pull-Down Menus 156
Standard Menus 158
File Menu 159
Selected Menu 162
Edit Menu 162
View Menu 165
Tools menu 166
Options menu 166
Help menu 166
What to Put in the Pull-Down Menus 167
Naming Menus in the Menu Bar 168
Naming Menu Entries in the Pull-Down Menus 169
Ordering Menus and Menu Entries in the Pull-Down Menus 170
Using Cascading Menus 171
Using Radio Buttons and Checkboxes in Pull-Down Menus 172
Choosing Mnemonics 172
Choosing Keyboard Accelerators 173
Disabling Menu Entries 174
Dynamic Menu Entries 175
Pull-Down Menu Guidelines 175
Popup Menus 179
What to Put in Popup Menus 180
Disabling Popup Menu Entries 180
Popup Menu Guidelines 181

9. Controls 183
Pushbuttons 184
Pushbutton Guidelines 184
Option Buttons 186
Option Button Guidelines 186
Checkboxes 187
Checkbox Guidelines 188

Radio Buttons 189
Radio Button Guidelines 189
Lists 190
List Guidelines 191
Text Fields 192
Text Field Guidelines 193
Scrollbars 194
Scrollbar Guidelines 195
Indigo Magic Scales 197
Indigo Magic Scale Guidelines 197
Labels 198
Label Guidelines 198
File Finder 199
File Finder Guidelines 200
Thumbwheels 200
Thumbwheel Guidelines 201
Dials 202
Dial Guidelines 202

10. Dialogs 205
Types and Modes of Dialogs 205
Dialog Modes 209
Guidelines for Using the Various Types and Modes of Dialogs 210

Designing Dialogs 211
Decorations, Initial State, and Layout of Dialogs 211
Standard Dialog Actions 213
Choosing Specific Actions for Your Dialogs 213
Choosing Default Actions 214
Labeling Dialog Buttons 215
Content of Specific Types of Dialogs 216
Prompt Dialogs 216
Error Dialogs 217
Warning Dialogs 217
Question Dialogs 217
Working Dialogs 218
Guidelines for Designing Dialogs 218
Invoking Dialogs 221
Invoking Dialogs When Manipulating Files 222
Other Situations for Invoking Dialogs 224
Guidelines for Invoking Dialogs 225

11. User Feedback 227
Types of Feedback 227
Provide Graphic Feedback 227
Keep Information Up to Date 228
Provide Messages to the User 229
General User Feedback Guidelines 229
Pointer Shapes and Colors 230
Standard Pointer Shapes and Colors 230
Designing New Pointer Shapes 232
Pointer Shapes and Colors Guidelines 233

A. Summary of Guidelines 235
Icon Appearance Design Guidelines 237
Icon Behavior Guidelines 238
Application Icon Accessibility Guidelines 239

Indigo Magic Look Guidelines 239
Application Window Characteristic Guidelines 240
Guidelines for Keyboard Focus Across Windows 243
Minimized Window Guidelines 243
Desks Guidelines 245
Session Management Guidelines 245
Software Installation Guideline 245
Guidelines for Designing Online Help 246
Guidelines for Creating SGIHelp Content 248
Desktop Variables Guidelines 250
File Monitoring Guideline 251
Data Exchange Guidelines 251
Application Model Guidelines 252
Primary Window Guidelines 253
Support Window Guidelines 254
Pointer Behavior Guidelines 255
Keyboard Focus and Navigation Guidelines 255
Selection Guidelines 256
Drag and Drop Guidelines 257
Menu Traversal and Activation Guidelines 257
Pull-Down Menu Guidelines 258
Popup Menu Guidelines 261
Pushbutton Guidelines 262
Option Button Guidelines 263
Checkbox Guidelines 264
Radio Button Guidelines 265
List Guidelines 266
Text Field Guidelines 268
Scrollbar Guidelines 269
Indigo Magic Scale Guidelines 270
Label Guidelines 271
File Finder Guidelines 272
Thumbwheel Guidelines 272

Dial Guidelines 273
Guidelines for Using the Various Types and Modes of Dialogs 274
Guidelines for Designing Dialogs 274
Guidelines for Invoking Dialogs 277
General User Feedback Guidelines 279
Pointer Shapes and Colors Guidelines 279

Index 281

Figures

Figure 1-1 Major Elements of the Indigo Magic Desktop 5
Figure 1-2 Icon States: User Actions and Effects 9
Figure 2-1 Application Icons 14
Figure 2-2 File Icons 15
Figure 2-3 Application and File Icon Components 16
Figure 2-4 IconSmith Examples 18
Figure 2-5 IconSmith Color Palette 20
Figure 2-6 Icon States and Effects on Color 21
Figure 2-7 Potential Icon Background Areas and Colors 23
Figure 2-8 Application Icon in Running and Not Running States 24
Figure 2-9 Examples of Application Icons That Could be Improved 26
Figure 2-10 Examples of Standard File Icons 27
Figure 2-11 Examples of File Icons for Application-Specific File Formats 28
Figure 2-12 Launch Dialog Box 31
Figure 2-13 Icon Catalog 33
Figure 2-14 The Find an Icon Tool 34
Figure 3-1 Examples of Graphic Modifications in the Indigo Magic Look 40
Figure 3-2 Color Scheme Control Panel 41
Figure 3-3 Features of a Typical Main Primary Window 45
Figure 3-4 Title Bar Label Appearing in Desks Overview 48
Figure 3-5 Labels for Main Window Title Bars 49
Figure 3-6 Default Maximum and Minimum Window Size Examples 51
Figure 3-7 Setting Auto Window Placement 52
Figure 3-8 Single-Action Pointer Grab Example: Capture by Sweeping 58

Figure 3-9 Multiple-Action Pointer Grab Example 60
Figure 3-10 Minimized Window Example: Good User Association With Application 61
Figure 3-11 Minimized Window Examples 62
Figure 3-12 Minimized Window Example: Incorrect Design 63
Figure 3-13 Minimized Window Example: Design That's Too Literal 63
Figure 3-14 Minimized Window Example: Indicating Status With the Label 65
Figure 3-15 Desks Overview Window 67
Figure 3-16 Setting Session Management 69
Figure 4-1 The Indigo Magic Software Manager 74
Figure 4-2 Typical Help Menu and Related Windows 78
Figure 4-3 Context-Sensitive Help Example 79
Figure 4-4 Typical Help Menu 82
Figure 4-5 Help Button Example 85
Figure 4-6 Task-Oriented Help Example 91
Figure 4-7 Language Control Panel 95
Figure 4-8 Mouse Settings Control Panel 96
Figure 4-9 Desktop Settings Control Panel 97
Figure 4-10 Selecting Preferred Editor in MediaMail 98
Figure 5-1 Clipboard Transfer Example 103
Figure 5-2 Primary Selection Example 105
Figure 5-3 Primary Transfer Example: Before Transfer 106
Figure 5-4 Primary Transfer Example: After Transfer 106
Figure 6-1 Allowable Parent-Child Window Relationships 115
Figure 6-2 "Single Document, One Primary" Application Model 116
Figure 6-3 "Single Document, Multiple Primaries" Application Model 117
Figure 6-4 "Multiple Document, Visible Main" Application Model 119
Figure 6-5 "Multiple Document, No Visible Main" Application Model 120
Figure 6-6 Basic Primary Window 121

Figure 6-7 Primary Windows With Tool Palettes 122
Figure 6-8 Primary Window With Two Panes 123
Figure 6-9 The IRIS Showcase Align Gizmo 130
Figure 6-10 The Indigo Magic Color Chooser 131
Figure 7-1 Location Cursor Example 136
Figure 7-2 Components and Fields 139
Figure 7-3 File Finder Component 149
Figure 8-1 Menu Bar 152
Figure 8-2 Pull-Down Menu 152
Figure 8-3 Cascading Menu 153
Figure 8-4 Popup Menu 153
Figure 8-5 Option Menu Button 153
Figure 8-6 An Open Option Menu 154
Figure 8-7 Elements of a Pull-Down Menu 157
Figure 8-8 Standard Menus for Menu Bars 158
Figure 8-9 The Standard File Menu 160
Figure 8-10 The Standard Edit Menu 163
Figure 8-11 Radio buttons 172
Figure 8-12 Checkboxes 172
Figure 8-13 Popup Menu 180
Figure 9-1 Pushbuttons 184
Figure 9-2 Option Button and Option Menu 186
Figure 9-3 Checkboxes 188
Figure 9-4 Radio Buttons 189
Figure 9-5 List 191
Figure 9-6 Scrollbar 194
Figure 9-7 Indigo Magic Scale 197
Figure 9-8 The File Finder 199
Figure 9-9 Thumbwheel 201
Figure 9-10 Dials 202
Figure 10-1 Sample Prompt, Error, Warning, Question, Working, and Information Dialogs 207
Figure 10-2 The Indigo Magic File Selection Dialog 208

Figure 10-3 Warning Dialog Layout 212
Figure 10-4 Warning Dialog With Save, Discard, and Cancel Buttons 216
Figure 10-5 Error Dialog With Specific Entity 217
Figure 10-6 Working Dialog with Indigo Magic Scale 218
Figure 10-7 Warning Dialog for Overwriting a File 223
Figure 10-8 Warning Dialog for Reverting to Previous Version 224
Figure 10-9 Product Information Dialog 224

Tables

Table 1-1 Mouse Button Names and Functions 10

Table 1-2 Keyboard Substitutes 11

Table 2-1 User / Icon Interactions and Expected Behavior 30

Table 3-1 Window Decorations and Window Menu Entries by Window Category 46

Table 5-1 Data Types Supported for Inter-Application Transfer 108

Table 7-1 Selection Actions and Results 143

Table 8-1 File Menu Entries 160

Table 8-2 Standard Edit Menu Entries 164

Table 10-1 Types of Dialogs, Their Modality, and When to Use Them 206

Table 11-1 Standard Pointer Shapes and Colors 231

About This Guide

This guide is written for developers of software products used on Silicon Graphics® workstations, including software engineers, graphical user interface designers, human factors specialists, and others involved in the design process. It contains recommended guidelines to help you design products that are consistent with other applications and that integrate seamlessly into the Indigo Magic™ Desktop. The result of this consistency and integration is that your products work the way end users expect them to work; consequently, end users find your products easier to learn and use. This guide provides information on how to design user interfaces for Silicon Graphics applications, along with specific examples of what is and isn't appropriate and why. Note that the guidelines discussed in this book are just that—guidelines, not rules; they're designed to apply to the majority of applications, but there will certainly be anomalous applications for which these guidelines don't make sense.

This guide assumes you're programming with the IRIS IM™ user interface toolkit. IRIS IM is the Silicon Graphics port of the industry-standard OSF / Motif™ user interface toolkit for use on Silicon Graphics computers. The Indigo Magic guidelines encourage compliance with the OSF / Motif guidelines described in *OSF/Motif Style Guide, Release 1.2*, so you should be familiar with the OSF / Motif manual before reading this one. In addition, the Indigo Magic guidelines clarify and elaborate on many OSF / Motif style issues; they recommend many value-added extensions and improvements to the OSF / Motif style that don't conflict with the basic OSF / Motif interface. Following the recommendations in this book will help ensure that your software product provides all of the functionality and ease of use designed into the Indigo Magic Desktop.

This guide also focuses on how your application should look and feel on the Indigo Magic Desktop when you're finished creating it—that is, how users will expect to be able to interact with your application. The implementation details of how to achieve this look and feel are covered in the *OSF/Motif Programmer's Guide* and the *Indigo Magic Desktop Integration Guide.*

What This Guide Contains

This guide contains two parts, which are described in the following sections. For your convenience, this guide is available online so that you can search it using the IRIS InSight™ Viewer; also, the online version contains many links to the *Indigo Magic Desktop Integration Guide* so that you can rapidly reach the necessary implementation information. Also available is the full-color Indigo Magic Style Card, which provides a quick overview of the important concepts presented in this guide.

Part One: Integrating With the Indigo Magic Desktop

This part describes how users will expect to be able to interact with your application from the Indigo Magic Desktop; it contains the following chapters:

- Chapter 1, “Overview of the Indigo Magic Desktop,” sets the context for Part One; it gives you an overview of the desktop environment in which users will encounter your application and describes the mouse and keyboard hardware provided with Silicon Graphics systems.
- Chapter 2, “Icons,” describes how to design your application and file icons so that they’re meaningful, they properly reflect their state (such as selected or open), and they behave appropriately for user actions such as double-click and drag-and-drop. It also describes how to make your application icon accessible so that users can interact with your application through the desktop tools, such as the Icon Catalog and the Find an Icon tool.
- Chapter 3, “Windows in the Indigo Magic Environment,” defines the various categories of windows and describes the Indigo Magic look for your application’s windows. This look is an enhanced version of IRIS IM and includes pre-packaged color and font schemes. This chapter also covers the expected behaviors that your application’s windows should support—such as sizing, moving, and minimizing windows, managing the keyboard focus across windows, interacting with desks, and responding to session management.
- Chapter 4, “Indigo Magic Desktop Services,” explains how your application can take advantage of several services provided by the Indigo Magic Desktop, such as Software Manager, SGIHelp, the IRIS

InSight online documentation viewer, global desktop settings, and the File Alteration Monitor (FAM).

- Chapter 5, “Data Exchange on the Indigo Magic Desktop,” describes the data transfer models that your application should support. It also lists the data types supported for data exchange in the Indigo Magic environment.

Part Two: Interface Components

The second part of this guide describes the individual components of an application, such as windows, menus, controls, and dialogs. Part Two contains the following chapters:

- Chapter 6, “Application Windows” discusses the different types of windows, how your application should combine them, what elements are appropriate for primary and support windows, and how these elements should be arranged.
- Chapter 7, “Focus, Selection, and Drag and Drop” discusses three general mechanisms by which users interact with your application: keyboard focus (within a window), selection, and drag and drop.
- Chapter 8, “Menus” describes the kinds of menus your application can use (pull-down, popup, and option menus), how users display, traverse, activate, and close these menus, and how to design menus and menu items for your application.
- Chapter 9, “Controls” describes controls that are supported in the standard OSF/Motif environment (such as push buttons, lists, and scrollbars) and those that are unique to the Indigo Magic environment (such as thumbwheels and dials). Each description consists of a general description of the control, and guidelines for when to use the control, how to label the control, and how the control should behave.
- Chapter 10, “Dialogs” defines the standard types of dialogs and discusses when to use them. It also covers how to design application-specific dialogs.
- Chapter 11, “User Feedback” describes various types of feedback users expect your application to provide. It also tells you when to use each of the standard pointer shapes and provides guidelines for designing your own pointer shapes.

Appendix

Appendix A, “Summary of Guidelines,” provides a checklist that you can use to determine whether your product follows the Indigo Magic user interface guidelines. This checklist contains all of the individual guidelines that appear throughout the book.

What You Should Know Before Reading This Guide

This guide assumes that you understand the concepts and terminology used with computers whose user interface is based on the X Window System™ and OSF / Motif. It also assumes that you’re familiar with the *OSF/Motif Style Guide, Release 1.2*; the material presented in this guide represents enhancements to and clarification of information presented in that manual.

Suggestions for Further Reading

The programming details of how to implement the style guidelines described here are contained in the following books, all of which are available online through IRIS InSight:

- Indigo Magic Desktop Integration Guide
- Software Packager User’s Guide
- Topics in IRIX Programming
- OSF / Motif Programmer’s Guide
- OSF / Motif Reference Manual
- The X Window System, Volume 1: Xlib Programming Manual
- The X Window System, Volume 4: X Toolkit Intrinsics Programming Manual

In addition, you might want to take a look at *IRIS Essentials* (also an online manual), which explains how users interact with the Indigo Magic Desktop. Finally, here are some references for further reading in the area of user interface design:

- Brown, C. Marlin "Lin." *Human-Computer Interface Design Guidelines*. Norwood, New Jersey: Ablex Publishing Corporation, 1989. (This book presents general user interface guidelines.)
- Laurel, Brenda, ed. *The Art of Human-Computer Interface Design*. Reading, Massachusetts: Addison-Wesley Publishing Co., 1990. (This book addresses future directions in user inteface design.)
- Mayhew, Deborah J. *Principles and Guidelines in Software User Interface Design* Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1992. (This book covers the overall design process and presents general user interface guidelines.)
- Norman, Donald A. *The Design of Everyday Things*. Reading, Massachusetts: Addison-Wesley Publishing Co., 1991. (This book uses examples of commonly used products to illustrate why good design is necessary.)
- Shneiderman, Ben (1992) *Designing the User Interface: Strategies for Effective Human-Computer Interaction*, second ed. Reading, Massachusetts: Addison-Wesley Publishing Co., 1992. (This book presents general user interface guidelines.)

Conventions Used in This Guide

In this book you'll find the following conventions that act as visual cues for different types of information.

Style Guidelines

❑ This icon indicates a guideline that summarizes the preceding discussion. The guidelines appear at the end of each main section, and a complete list is provided in Appendix A, "Summary of Guidelines."

Font Conventions

These are the typographical conventions used in this book:

- **Bold text** indicates that a term is a data type, a keyword, an X or IRIS IM widget name, a function, or an X resource.
- *Italics* indicates that a term is a file name, a button name, a variable, an IRIX™ command, or a document title.
- “Quoted text” indicates menu items.
- Angle brackets indicate special keys, as in <Ctrl>.
- Regular text is used for menu and window names.

PART ONE

Integrating With the Indigo Magic Desktop

Chapter 1

Overview of the Indigo Magic Desktop

The Indigo Magic Desktop environment allows users to interact with applications by using a graphical, point-and-click interface based on icons and windows. It offers an easy, powerful alternative to typing commands in the traditional UNIX® style.

This chapter provides a brief overview of the Indigo Magic Desktop environment from the user's perspective. It describes how users expect to interact with their graphical environment and with your application. The specific implications for your application are discussed in the remainder of Part One, "Integrating With the Indigo Magic Desktop." For details on how users interact with the Indigo Magic Desktop, see the online *IRIS Essentials* manual.

This chapter covers the following topics:

- "Overview of the Desktop" briefly describes the major elements of the Indigo Magic Desktop including the various types of desktop icons.
- "How Users Interact With Desktop Icons" describes how users interact with icons and how icons appear on the desktop during these interactions.
- "Mouse and Keyboard Hardware" describes the mouse and keyboard hardware provided for use with Silicon Graphics systems.

Overview of the Desktop

The Indigo Magic Desktop appears in Figure 1-1 along with the major desktop tools and examples of icons and minimized windows. The important elements in the Indigo Magic Desktop environment are:

- *4Dwm*—the window manager application underlying the Indigo Magic Desktop. (*4Dwm* is the only sanctioned window manager for the Indigo Magic environment.) It is an X Window System client based on the Motif Window Manager (MWM). *4Dwm* provides window management, desks (defined below), and session management. It provides facilities for controlling window placement, window size, keyboard focus ownership, and minimized windows.
- Toolchest—a menu bar that provides general system commands not specific to any application. This menu bar is displayed horizontally or vertically, depending on the user's preference, and is positioned by default in the upper left corner of the desktop.
- Directory Views—windows that display the contents of a UNIX file directory (folder icon) in various formats involving icons. The Directory View has an optional *shelf* pane where users can store frequently used icons. Note the thumbwheel on the lower left side of the window—it lets users adjust the viewing size of icons in the display area of the Directory View window.
- Icons on the screen background—symbols that represent entities, such as applications, files, directories, people, printers, the Indy Cam™ camera, removable media devices (for example, CDROM, floppy, and DAT drives), and other devices. Users can place icons in any portion of the desktop to have them readily accessible.
- Icon Catalog—a window that allows users to access application icons directly. It uses a page-oriented metaphor with six default pages: Applications, Collaboration, Demos, Desktop Tools, Media Tools, and Control Panels. Users can also create their own pages.
- Find an Icon tool—a tool for either retrieving a file's icon, given the name, or obtaining the full path and filename given the icon.

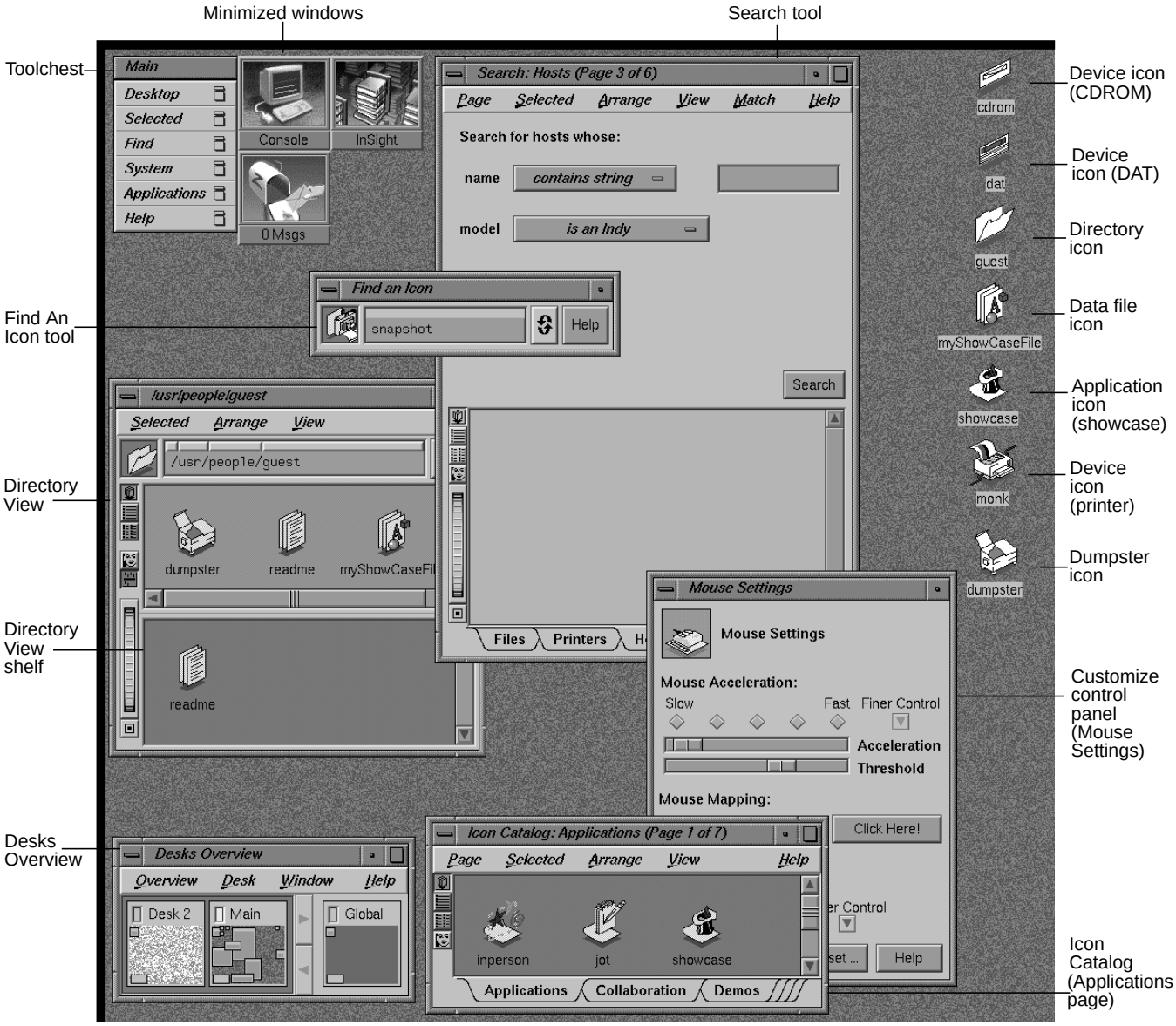


Figure 1-1 Major Elements of the Indigo Magic Desktop

- Search tool—a tool that lets users search for files, printers, hosts, drives, and people.
- Customize control panels—windows that let users specify various types of preferences, such as screen background and language.
- Desks Overview[1]—a tool for creating and managing multiple virtual screens (desks). It lets users set up and switch desks, typically for organizing their work. *4Dwm* allows these desks to display different backgrounds.

These tools affect the design of your application. Specific design implications are discussed in detail in the remainder of Part One. For more information on the tools themselves, see the *IRIS Essentials* manual.

How Users Interact With Desktop Icons

Users interact with icons on the Indigo Magic Desktop using a point-and-click graphical user interface. For example, the following techniques represent some of the ways a user can launch an application using icons:

- double-clicking the application icon, which launches the application and opens a new file
- double-clicking a file icon, which launches the application opened to that specific file
- selecting the application icon (by single-clicking or dragging a rectangle that encloses it) and then choosing “Open Icon” from the Selected

[1] Due to a software patent dispute instituted by Xerox Corporation, the Desks and Desks Overview features of this version of the IRIX operating system are now optional and may or may not be available to you after May 15, 1995. On this date, these features will be disabled unless you have entered a license code obtained from Silicon Graphics. When the Desks Overview feature is initiated, a dialog box will periodically warn you of the May 15, 1995 expiration date. Depending on how this patent dispute is resolved, prior to May 15, 1995, Silicon Graphics will attempt to inform you that the license code for these features will: (1) not be available; (2) only be available at an additional price; or (3) be available free of charge. Silicon Graphics apologizes for this inconvenience and appreciates your understanding. For updates, see comp.sys.sgi.admin, or call 1-415-390-4334. Please do not send email, or call other SGI numbers.

menu (in the Toolchest for icons on the screen background or in the specific tool window for other icons)

- dragging a compatible file icon and dropping it on top of the application icon, which launches the application and opens that specific file

The six states that are used to represent icon manipulations are indicated primarily by painting specific portions of the icon with a predefined "icon color." As the state changes, the areas painted with the *icon color* change color. For the specific implications on the design of your application's icons, see "General Icon Design: Components, Size, and Colors" in Chapter 2.

The icon states of the IRIS Showcase application icon are illustrated in Figure 1-2. (Note that the magic carpet under the magician's hat is the generic executable symbol used in the Indigo Magic environment to identify the icon as an application icon.) In the figure, the hat brim, the light shading on the hat, and the carpet are in the predefined *icon color* and thus change color as the state changes. The states are:

- *neutral*—the icon isn't involved in any operations. The icon appears in the base colors you've chosen. The portions containing the *icon color* appear in light gray.
- *locate highlight*—the pointer is resting on the icon and the icon isn't currently selected (see the following description of the *selected* state). The *icon color* portions change from the neutral state color light gray to white. The locate highlight feature lets the user know that the highlighted icon will be selected if the user single-clicks or opened if the user double-clicks. (The locate highlight feature also applies to components in windows to provide feedback as to which objects are true components and which are passive graphics, as described in "Enhanced Graphics in the Indigo Magic Look" in Chapter 3.)
- *selected*—the icon has been chosen potentially for some operation. Users select an icon by single-clicking on it with the left mouse button or dragging a rectangle around it using the left mouse button. When an icon is selected, the *icon color* portions turn yellow. Note that the *icon color* is bright yellow when the window containing the icon (or the screen background) has the keyboard focus; otherwise, the *icon color* is a dim yellow. Users can choose entries from the Selected menu (from the Toolchest if the icon is on the screen background or from a window's

menu bar if contained in a tool window) to perform various operations on the object represented by the icon.

- *open*—applies to application icons but not to data file icons. For application icons, the open state indicates that the application is running. Application icons indicate their open state by moving the magic carpet from a horizontal to a vertical position and by changing their application icon symbol. This is discussed in more detail in "Application Icon Design" in Chapter 2. Note that application icons don't indicate their open state with colors.
- *drag*—an icon is in the process of being moved on the screen. Users drag icons by pressing and holding down the left mouse button while the pointer is positioned over the icon and then moving the mouse. The icon moves around the desktop with the pointer. As an icon is dragged, the portions colored with the *icon color* display in yellow since it's in the selected state, and a ghost image of the icon with the *icon color* portions displaying in dark gray remains in the original position. The ghost image remains until the user drops or places the icon or cancels the drag operation.
- *drop-accepting*—an icon has another icon moved on top of it to perform some operation. For example, users can move a file from one directory to another by dragging the icon representing the file and dropping it onto the icon representing the new directory. When an icon has another icon positioned over it, the *icon color* portions of the destination icon turn royal blue if the destination icon can accept dropped icons or remain in their current color if the destination icon doesn't accept them.

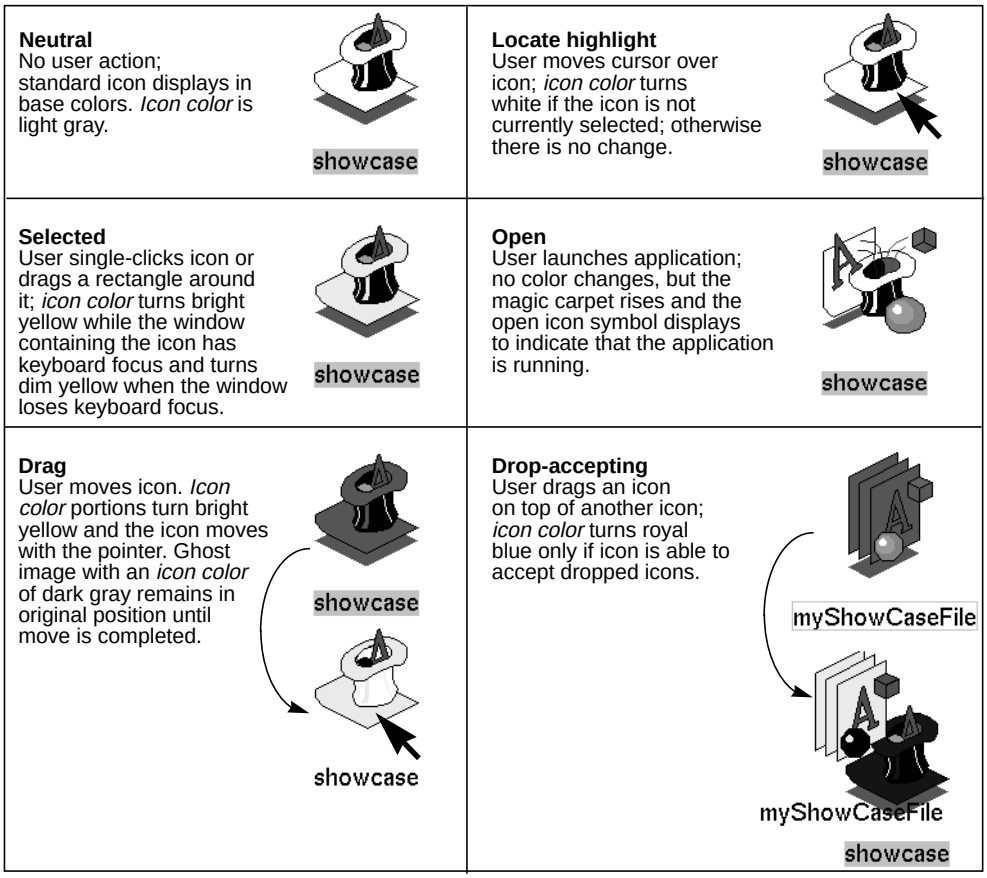


Figure 1-2 Icon States: User Actions and Effects

Mouse and Keyboard Hardware

Silicon Graphics systems come with a three-button mouse that supports the mouse actions defined in the *OSF/Motif Style Guide* (such as press, release, click, motion, multiclick, multipress, and multimotion). Table 1-1 lists these buttons and their functions. If a mouse action is mentioned in this guide without reference to a specific mouse button, assume that the button being used is the left mouse button. For example, "when the user clicks on the OK button. . ." means "when the user positions the pointer over the OK button and clicks the left mouse button. . . ." Note that users can switch the mouse to a left-handed mouse via the Mouse Settings control panel available from the Desktop->Customize menu in the Toolchest.

Table 1-1 Mouse Button Names and Functions

Button[a]	OSF/Motif Name	Silicon Graphics Name	Function
Leftmost button	BSelect	Left mouse button	Used for all primary interactions, including selection, activation, and setting the location cursor.
Middle button	BTransfer	Middle mouse button	Used for moving and copying elements. Can be used for advanced user shortcuts that are also included in a more obvious interface.
Rightmost button	BMenu	Right mouse button	Exclusively used for popping up menus.

a. This table assumes a right-handed mouse.

Silicon Graphics keyboards support the following special keys that are used to interact with Motif-compliant applications:

<Tab>

<Space>

<Back Space>

<Escape>

<Insert>

<Delete>

<Home>

<End>

<Page Up>

<Page Down>

Modifier keys: <Ctrl>, <Alt>, <Shift>

Ten function keys: <F1> through <F10>

Special printing characters: </>, <\>, <!>

Arrow keys: <down arrow>, <left arrow>, <right arrow>, <up arrow>

In addition to these keys, Silicon Graphics keyboards also support the function keys <F11> and <F12>, which aren't included in the *OSF/Motif Style Guide*. If your application uses these keys, limit them to application-specific functionality rather than the general functionality described in this guide.

Table 1-2 lists the keys that the *OSF/Motif Style Guide* defines but that don't appear on Silicon Graphics keyboards; it also lists the corresponding Motif-compliant substitutions for Silicon Graphics keyboards.

Table 1-2 Keyboard Substitutes

OSF/Motif Key	Silicon Graphics Substitute
<Return>	<Enter>
<Cancel>	<Escape>
<Help>	<F1>
<Menu>	<Shift><F10>
<Begin>	<Home>

Chapter 2

Icons

The Indigo Magic Desktop uses icons to represent entities such as applications, files, directories, people, printers, the Indy Cam camera, removable media devices (for example, CDROM, floptical, and DAT drives), and other devices. Users can manipulate these icons through a point-and-click graphical user interface to initiate certain actions, such as launching an application or sending a data file to the printer, instead of entering UNIX commands in a shell window. This chapter covers the following topics:

- "Designing the Appearance of Icons" tells you how to design an application icon to represent your application's executable file and a file icon to represent your data files.
- "Defining the Behavior of Icons With FTRs" explains which File Typing Rules (FTRs) you need to define for your application and file icons so that they will respond to user actions such as double-click and drag-and-drop.
- "Making Application Icons Accessible" discusses adding your application icon to the Icon Catalog and naming and locating your executable file in the file system so that users can easily find your application icon.

Note: In this guide, the term *icon* refers to Indigo Magic Desktop icons and not to minimized windows, as in the *OSF/Motif Style Guide*. Minimized windows are described in"Minimized Windows" in Chapter 3.

Designing the Appearance of Icons

Your application's desktop icons can appear in any of several places on the Indigo Magic Desktop, along with icons from other applications. For example, your icons can appear on the desktop background, in Directory View windows, and in the Icon Catalog. Figure 2-1 shows the Icon Catalog, which displays several application icons representing executable files, and Figure 2-2 is a snapshot of various data file icons that might appear on a user's desktop. You'll need to create an application icon. If your application saves its data in a unique format, you'll also need to create a data file icon. (If your application saves its data in a standard data format, your application's data files will automatically appear on the desktop using the appropriate standard data icon.)

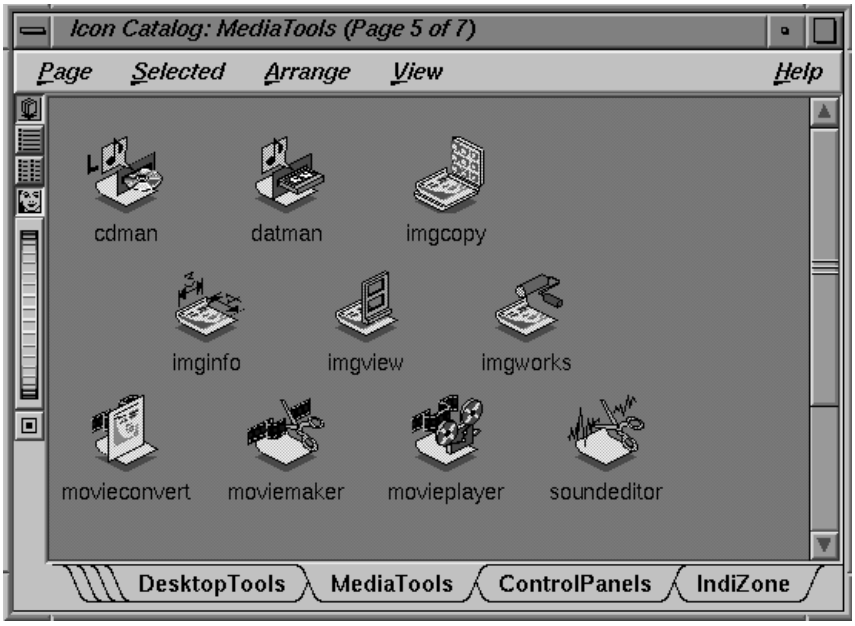


Figure 2-1 Application Icons

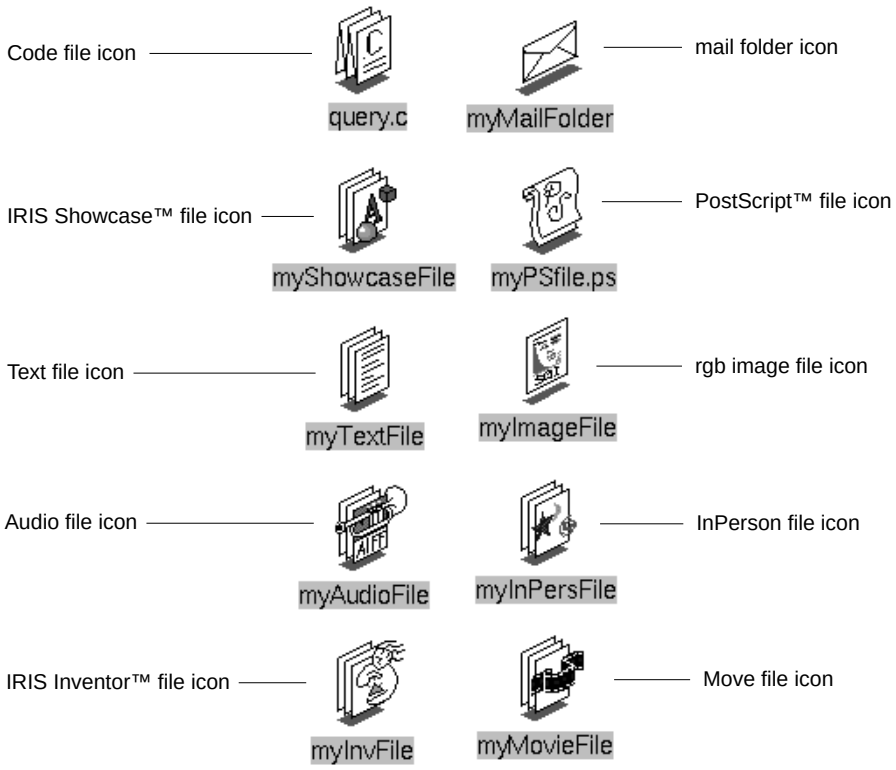


Figure 2-2 File Icons

To make your application and file icons stand out in a group of icons, you need to create meaningful symbols that are distinctive and represent your product. Your icon designs should also follow the guidelines detailed in the remainder of this section so that users recognize your icons as desktop icons and know that they can interact with these icons in expected ways.

General Icon Design: Components, Size, and Colors

You should design your application and file icons so that they

- convey your product identity
- have the right components and thus are recognizable as desktop icons

- are identifiable when zoomed to the minimum and maximum sizes allowed by the Indigo Magic Desktop
- use an effective color scheme, which allows the icon to reflect its state

Keep in mind that your primary goal is to differentiate your icons from those of other products and that your design should be fairly simple because desktop icons can be displayed at a relatively small size.

Icon Components

As shown in Figure 2-3, application and file icons consist of several components: a symbol identifying the application or type of data file, an outline around the edge of the symbol, a drop shadow that helps give the icons a 3D appearance, and a label identifying the executable or data file. Application icons also include a *magic carpet*, the generic executable symbol, which differentiates them from file icons and shows whether or not the application is running (by moving from the horizontal, at-rest position into a vertical, up-and-running position). File icons for document-based applications incorporate the generic data file symbol (three rectangles representing sheets of paper) into the file icon design.

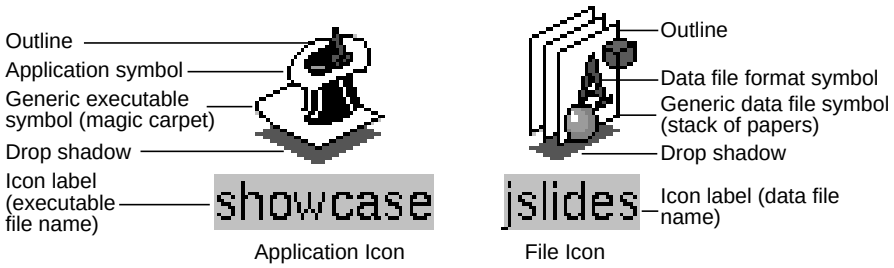
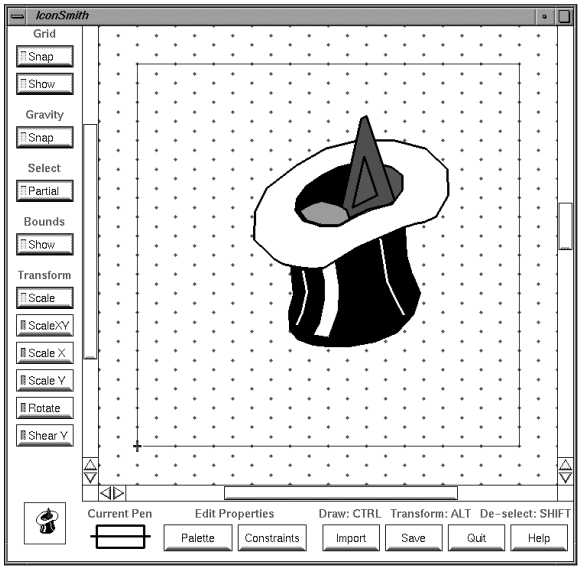


Figure 2-3 Application and File Icon Components

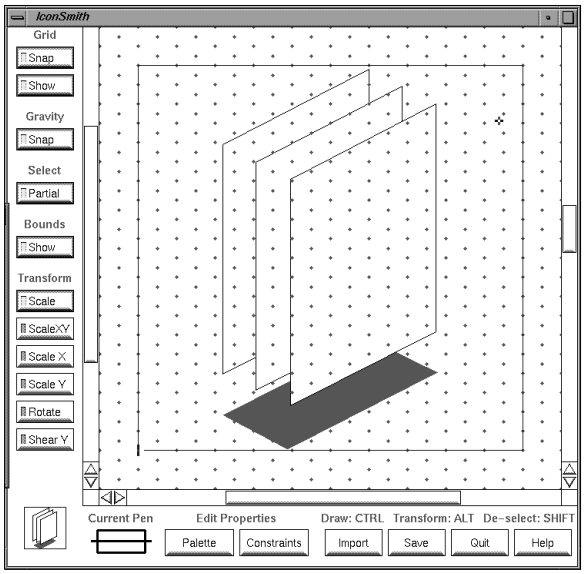
You need to create the symbols for your application icon and for any unique file types your application creates. You create symbols with the IconSmith™ drawing tool, which is described in Chapter 12, “Using IconSmith,” in the *Indigo Magic Desktop Integration Guide*. IconSmith provides a drawing grid with tools for creating 3D graphics. It also supplies predefined templates for the magic carpet in both the running and not running states and the data file format symbol, complete with drop shadows. You create your unique icon symbols in the drawing area and can readily add the predefined templates

to your design. Figure 2-4 shows four examples of IconSmith. The drawing area in the example in the upper left corner contains the IRIS Showcase application symbol for its not-running state. The other three examples contain the three predefined templates. The icon labels are supplied automatically when the icons are displayed on the desktop. This label is the name of the executable for application icons and the name of the data file for file icons.

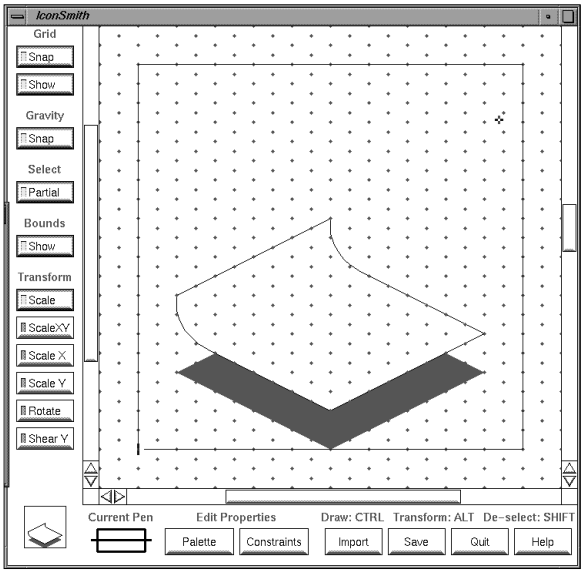
Showcase application symbol (not running)



Data file symbol



Magic carpet symbol (not running)



Magic carpet symbol (running)

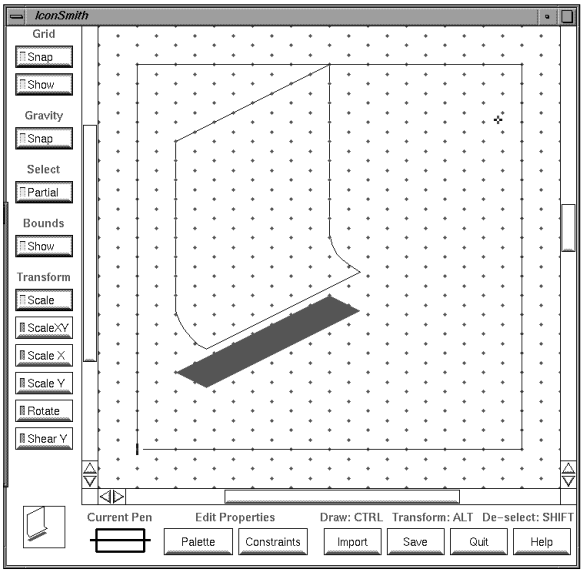


Figure 2-4 IconSmith Examples

Icon Size

You need to draw your icon within the 100×100-pixel boundary box defined by IconSmith. Keep in mind that your symbol designs should allow any generic icon components, such as the magic carpet and stack of papers, to be at least partially visible. This is described in more detail in "Application Icon Design" and "File Icon Design" later in this chapter.

By default, icons are reduced to fit within a 50x50 pixel area when they're displayed on the desktop. Because users can adjust the viewing scale for icons making the icons either larger or smaller than this default, your icon drawings should look good across the range of available sizes. You can use the IconSmith preview box to see what your icon looks like when scaled to various sizes.

Icon Colors

IconSmith provides the color palette shown in Figure 2-5 for specifying the colors in your icon. The following two buttons are of particular importance for applying color to icons:

- *Specific*—lets you apply a color from the color selection area to the selected part of the graphic. The *specific color* remains constant as an icon goes through its states.
- *Icon*—lets you apply the predefined *icon color* to the selected part of the graphic. Portions painted with the *icon color* change color as the icon changes its state.

When choosing colors for your icons, consider that the *icon color* changes to indicate state and that icons appear against a variety of differently colored backgrounds. To accommodate these issues, you need to enable your icon to show its state and make sure your icon stands out from the most likely backgrounds, as explained in the following paragraphs. (See Chapter 1, "Overview of the Indigo Magic Desktop," for explanations of icon states from the user's point of view.)

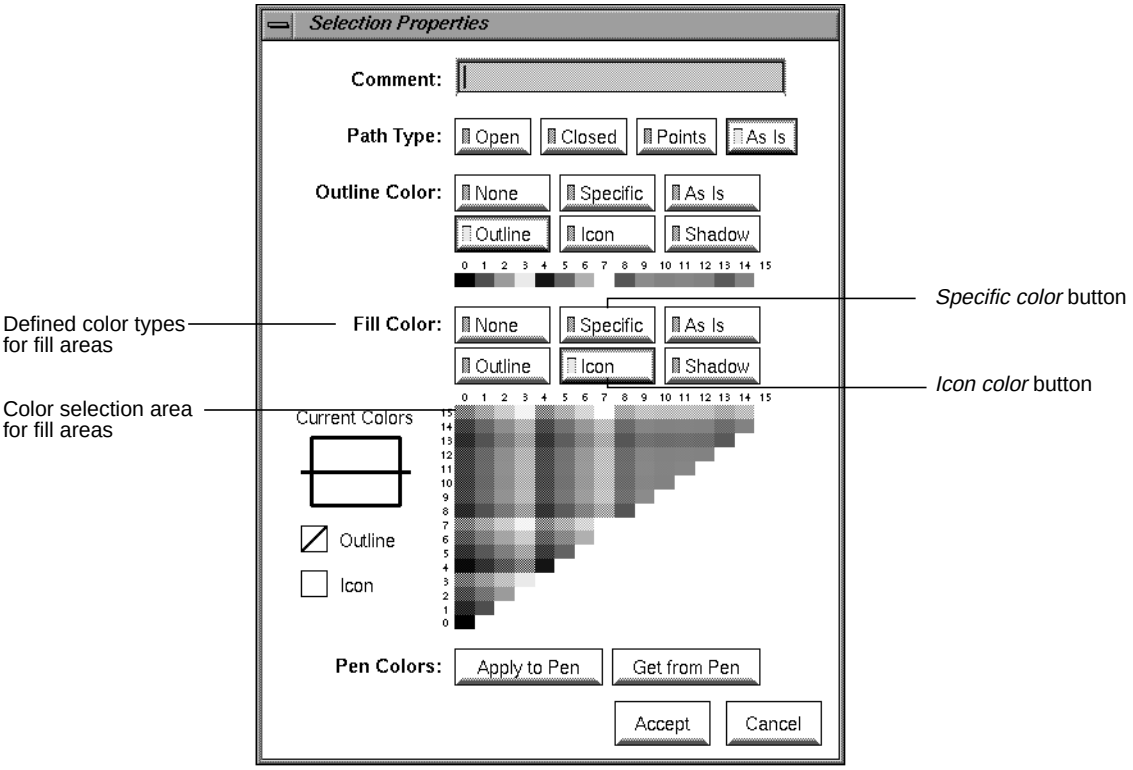


Figure 2-5 IconSmith Color Palette

To enable your icon to reflect what state it's in, you need to paint portions of it with the predefined *icon color* supplied by IconSmith. Those portions of your icon are changed automatically by the Indigo Magic Desktop to indicate the icon's state. In the example shown in Figure 2-6, the hat brim, the light shading on the hat, and the carpet of the IRIS Showcase application icon use the predefined *icon color* (which is a light gray in the neutral state). In addition, to make the state color changes easier for users to detect in your icons, avoid (or use sparingly) intense, strongly saturated colors and the specific colors used by the Indigo Magic Desktop to indicate state—bright yellow, dim yellow, royal blue, pure white, and light gray.

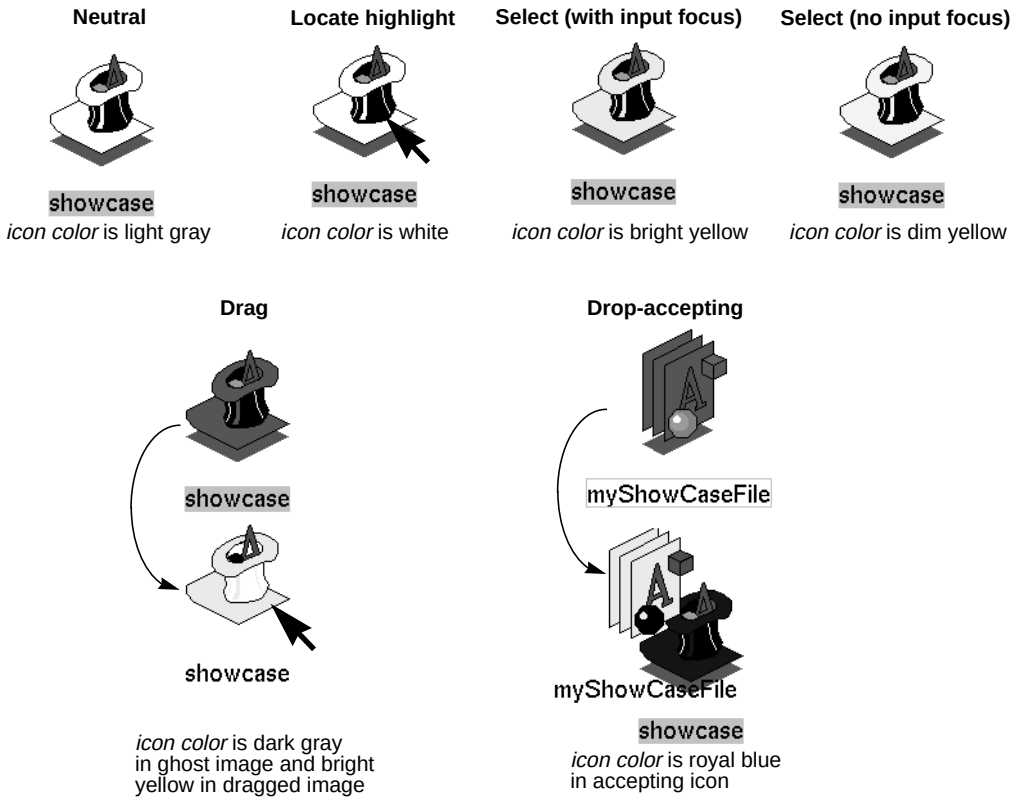


Figure 2-6 Icon States and Effects on Color

There are several things you can do to increase your icon's chances of standing out on user-customized desktops. (Remember that users can customize the desktop background to be any color and pattern, as described in Chapter 1, "Overview of the Indigo Magic Desktop.")

- Use the *icon color* defined by IconSmith to color most of your icon.
- Use two or more areas of accent colors to help your icon stand out against user-customized background colors.
- Avoid using only very small color areas (2-4 pixels) because these small areas may be difficult to see against a patterned background.
- Use the *outline color* (which is black) supplied by IconSmith to define the outline of your icon.

- Since your icons can display in certain Indigo Magic tools as shown in Figure 2-7, avoid (or use sparingly) the following background colors used by these tools:
  - light gray-green—used in Directory View windows
  - cadet blue—used in read / write panes such as drop pockets, shelves in Directory View windows, and the Icon Catalog pages
  - Navajo white—used in read-only panes such as the results area of the Search tool

  Note: The color descriptions “light gray-green” and “cadet blue” are approximate since hardware and gamma values vary. Look at these colors on your system in Directory View windows and read / write panes, and avoid using these colors in your icon or use them sparingly.

Icon Orientation

Icons on the Indigo Magic Desktop should display at a three-quarter view and face the lower right of the screen to make them appear 3D. When designing your application and data file format symbols, draw them using this perspective. This is the same perspective that is used for the generic executable symbol (the magic carpet) and the generic data file symbol (stack of papers). Icons that don’t follow this convention appear to be facing the wrong way.

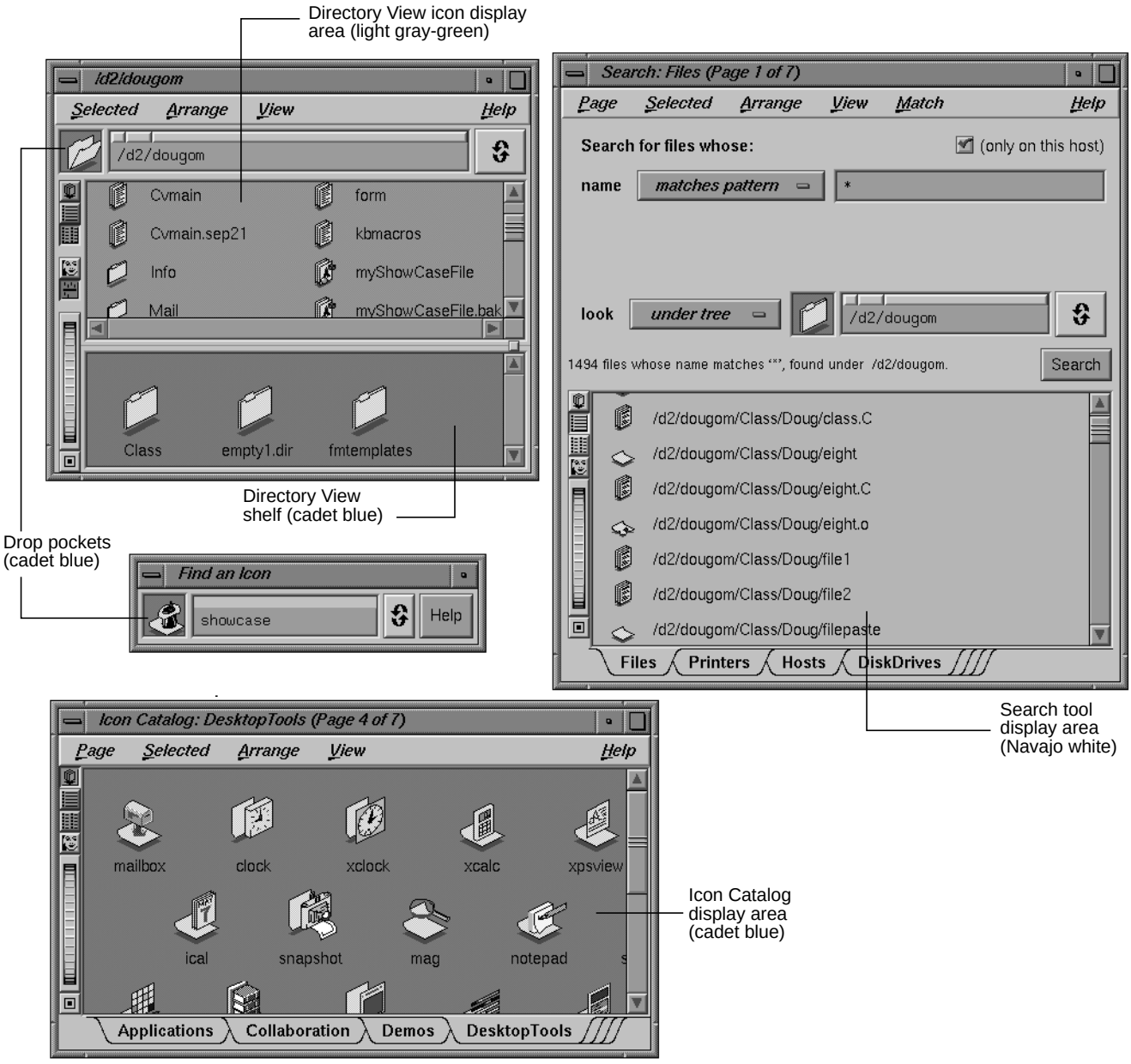


Figure 2-7 Potential Icon Background Areas and Colors

Application Icon Design

In addition to adhering to the guidelines discussed in the previous section, "General Icon Design: Components, Size, and Colors," the application icon that represents your application's executable needs to convey to users

- that it's an application icon (as opposed to a file icon)
- what its current state is (whether or not the application is running)

The magic carpet is the generic executable symbol; it differentiates application icons from file icons. As mentioned in the previous section, the magic carpet and its drop shadow are predefined components that can easily be incorporated into your icon design when using IconSmith. For details of having them appear with your application symbol, see Chapter 11, "Creating Desktop Icons: An Overview," in the *Indigo Magic Desktop Integration Guide.*

As shown in Figure 2-8, your application icon should use both of the following techniques to indicate its state—that is, whether or not the application is running:

- The 3D application symbol changes (in this example, items pop out of the hat in the running state).
- The magic carpet moves from a horizontal, at-rest position to a vertical, up-and-running one, and the drop shadow changes shape appropriately. Since the magic carpet must be recognizable in both states, make sure that your application symbol doesn't completely obscure it in either position.

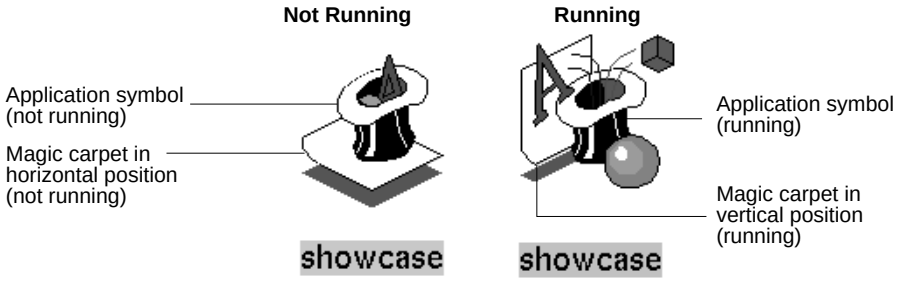


Figure 2-8 Application Icon in Running and Not Running States

To have your application icon change to reflect its running state, create two different symbols using IconSmith and associate them with your application. See Chapter 11, “Creating Desktop Icons: An Overview,” in the *Indigo Magic Desktop Integration Guide* for details on how to do this. The specific design of the two symbols is up to you, but keep in mind that you’re trying to convey an inactive state and an active, running one. Also, when viewed in succession, the two symbols should look like a progressive animation rather than two unrelated icons.

The IRIS Showcase application icon shown in Figure 2-8 is an example of a well-designed icon: it animates in a clever way to show which state the application is in, and the carpet is always visible. Figure 2-9 shows three examples of application icon designs that can be improved:

- The first example is an application icon for a point-and-click ASCII text editor. Its carpet is clearly visible and functions properly. The application symbol doesn’t change, however, to reflect that the application is running. To improve the design, the existing symbol could be used to show the running state and the pencil could be aligned at the bottom or side of a blank version of the pad when the application isn’t running.
- The application icon in the second example represents an online book viewer. The application symbol changes nicely to indicate its running state. The carpet, however, is almost totally obscured when the application isn’t running and doesn’t appear at all when the application is running.
- The third application icon represents a database application for tracking software bugs. The application symbol is visually appealing but completely obscures the carpet. In addition, there’s no change in the application symbol to reflect its running state, and the symbol is oriented toward the lower left instead of the lower right. The design could be improved by adding a second symbol for the not-running state (perhaps showing the file drawer closed with the bug’s antennae sticking out), making the application symbol smaller so that the magic carpet is visible, and redrawing the symbol so that it faces the lower right.

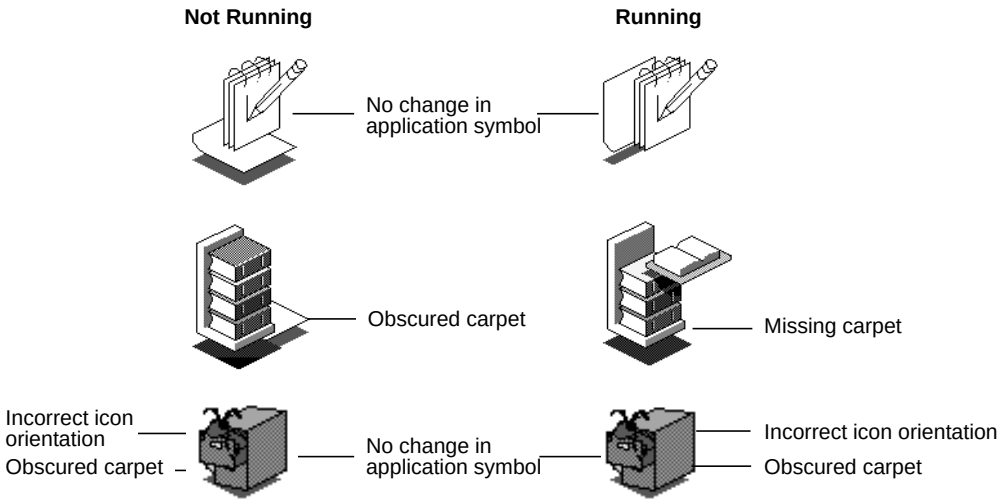


Figure 2-9 Examples of Application Icons That Could be Improved

File Icon Design

If your application creates files, those files need to be associated with *file icons*. Silicon Graphics defines a set of standard data formats and provides associated file icons, some of which are shown in Figure 2-10. (For information on these file formats, see Appendix E, “Predefined File Types,” in the *Indigo Magic Desktop Integration Guide*.) If your application saves its data in any of these standard formats, it will automatically use the appropriate file icon for those files.

If your application saves its data in a unique format, however, you need to design a distinctive file icon that relates graphically to the theme of your application icon. It should also indicate, if possible, how the data is used. Your file icon should follow the guidelines discussed in this section and in the earlier section “General Icon Design: Components, Size, and Colors.”

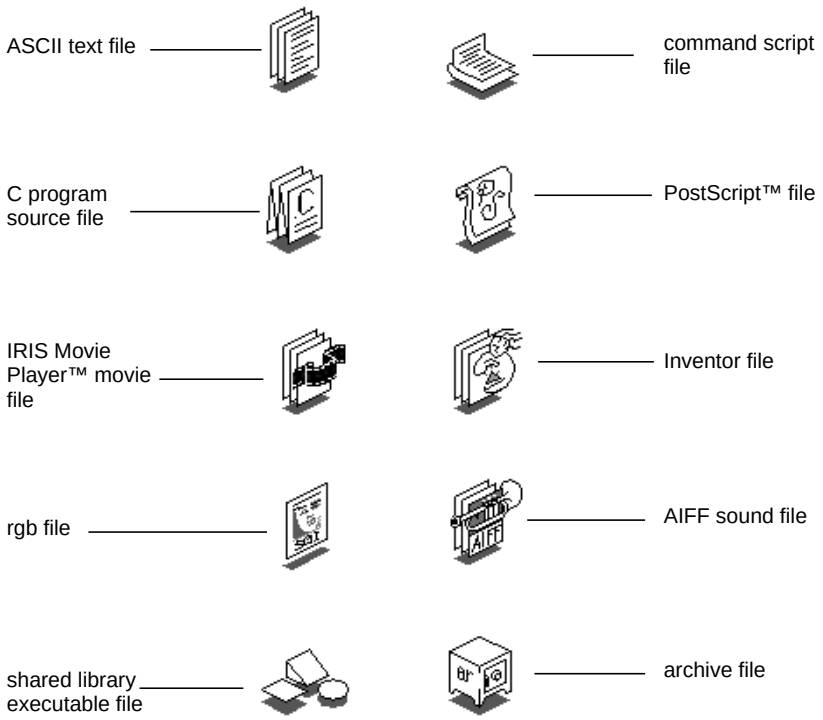


Figure 2-10 Examples of Standard File Icons

The standard file icons shown in Figure 2-10 that symbolize documents all contain the generic data file symbol (stack of papers). If your application creates data files, your file icon should include this generic symbol. Both the generic symbol and its drop shadow are predefined components available in IconSmith. If you're creating a unique icon that does not use the generic data file symbol, create an appropriately shaped drop shadow for your design and paint it with the *shadow color* predefined in IconSmith.

Figure 2-11 shows some file icons for application-specific file formats. For example, the IRIS Showcase file icon includes the generic data file symbol (a stack of papers) to indicate that it represents a document, and the data file format symbol is similar to that found in the IRIS Showcase application symbol for the running state.

Figure 2-11 Examples of File Icons for Application-Specific File Formats

Icon Appearance Design Guidelines

For any icon you create . . .

- ❑ Provide a meaningful, distinctive symbol that gives your product an identity and that allows users to readily identify your application and its corresponding custom data files, if any.
- ❑ Keep your design fairly simple because desktop icons can be displayed at very small sizes.
- ❑ Make sure that your icon can be identified across the range of viewing sizes.
- ❑ Color most of your icon using the *icon color* predefined by IconSmith so that your icon's state is easy to detect.
- ❑ Use two or more areas of accent colors to help your icon stand out against user-customized background colors.
- ❑ Avoid small areas of color (2-4 pixels) because they're difficult to see against patterned backgrounds.
- ❑ Include an outline around your custom symbol, and use the *outline color* supplied by IconSmith.
- ❑ Avoid or use sparingly intense, strongly saturated colors and the specific colors used by the Indigo Magic Desktop—bright yellow, dim yellow, royal blue, light gray-green, cadet blue, and Navajo white. These colors make it difficult to distinguish between certain icon states and to find your icon against the background colors of many desktop tools.
- ❑ Orient your icon so that it displays a three-quarter view that faces the lower right corner of the screen.

When designing an application icon . . .

- ❑ Include the magic carpet, the generic executable symbol, with your application's symbol.
- ❑ Indicate the state of the application (not running vs. running) by providing two different application symbols and by moving the magic carpet from a horizontal (not running) to vertical (running) position. Remember that your application symbols should resemble a progressive animation when viewed in succession.
- ❑ Make sure that your application symbols do not obscure the magic carpet in either its horizontal or vertical position.

If your application saves data in a custom file format . . .

- ❑ Design a unique data file format symbol that is readily associated with your application icon design and also indicates how the data is used.
- ❑ If your application is document-based, include the generic data file symbol (stack of papers) in your design.
- ❑ If your data file icon does not use the generic data file symbol, create an appropriately shaped shadow for your file icon and use the predefined *shadow color* supplied by IconSmith.

Defining the Behavior of Icons With FTRs

You define the behavior of icons on the Indigo Magic Desktop by creating a File Typing Rule (FTR) file for each icon. This behavior includes such things as what happens when the user double-clicks on an icon and what happens when the user drags and drops one type of icon onto another type of icon. This section describes how users can interact with icons, how an FTR supports these interactions (in brief), and the minimal set of standard behaviors your application and file icons should support.

Table 2-1 shows the behavior users expect for given interactions with application and file icons. The last column lists the rules for each type of interaction. For information on implementing these behaviors, see

Chapter 11, “Creating Desktop Icons: An Overview,” in the *Indigo Magic Desktop Integration Guide.*

Table 2-1 User / Icon Interactions and Expected Behavior

User Goal	User Action	Expected Behavior	Implementation Hints
Launch application	Selects application icon and chooses “Open” from the corresponding Selected menu -or- Double-clicks application icon	The magic carpet and the application symbol change from the not-running to the running state. The application launches, set to a new file.	Add a CMD OPEN rule for the application icon
Launch application with a particular file by directly opening file	Selects file icon and chooses “Open” from the corresponding Selected menu -or- Double-clicks file icon	The magic carpet and the application symbol change from the not-running to the running state. The application launches, set to the specified file.	Add a CMD OPEN rule for the file icon that specifies its application[a]
Launch application with a particular file by dragging and dropping	Drops file icon onto application icon	If file is compatible, application launches, set to specified file If file is incompatible, application posts an appropriate error message and doesn’t launch. In both cases, the magic carpet and the application symbol change from the not running to the running state.	Add a CMD DROP rule for the application icon
Launch application with command-line arguments	Double-clicks application icon while holding down the <Alt> key	The Launch dialog box opens, displaying the path to your executable (see Figure 2-12). Users can add arguments if desired. Clicking *OK* executes the text input as specified. The magic carpet and the application symbol then change from the not-running to the running state.	Add a CMD ALTOPEN rule for the application icon[a]
Print file	Selects file icon and chooses “Print” from the corresponding Selected menu -or- Drops file icon onto printer icon	Specified file prints on default printer if activated by a menu selection or prints on specified printer if activated by a drag and drop action.	Add a PRINT rule for the file icon[a]

a. Design the application so it can accept specific data (for example, a filename) as a command-line argument.

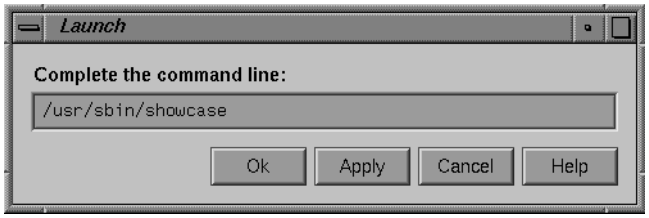


Figure 2-12 Launch Dialog Box

Icon Behavior Guidelines

When creating an FTR to define your application icon's behavior . . .

- Provide a CMD OPEN rule that launches the application. This allows the user to open your application either by selecting your application icon and then choosing "Open" from the corresponding Selected menu, or by double-clicking your application icon.
- Provide a CMD ALTOPEN rule that opens the Launch dialog box shown in Figure 2-12 with the path to your executable displayed in the text field of this window. This allows the user to open the Launch dialog box by double-clicking your application icon while holding down the <Alt> key.
- Provide a CMD DROP rule that launches your application with the file specified by the dropped icon. If your application doesn't understand the type of file represented by the icon dropped on it, your application should provide an appropriate error message to the user rather than launching. This allows the user to launch your application with a specific file by dragging the file icon and dropping it on your application icon.

When creating an FTR for your file icon . . .

- Provide a CMD OPEN rule that launches your application and automatically opens the file represented by the file icon. This allows the user to open a file created by your application either by selecting the file icon and then choosing "Open" from the corresponding Selected menu, or by double-clicking the file icon.

- ❑ Provide a CMD PRINT rule that sends the file represented by the file icon to the specified printer. This allows the user to send your application's data files to the default printer by selecting the file icon and then choosing "Print" from the corresponding Selected menu. It also allows the user to send your application's data files to any printer by dragging the file icon and dropping it on an icon that represents the specific printer.

Making Application Icons Accessible

In addition to designing the appearance and defining the behavior of your application icon, you need to make it accessible to users by

- putting it into the Icon Catalog
- naming and locating your executable file appropriately so that users can find it with the Find an Icon tool

Putting Icons Into the Icon Catalog

After users install an application, they expect to find its icon in the Icon Catalog (described in "Overview of the Desktop" in Chapter 1), so you need to add your application icon to the Catalog as part of the installation process. Also, decide on which page of the Catalog your icon should appear or, if necessary, create a new page. Once you've chosen an appropriate page for your application icon, refer to Chapter 15, "Adding Your Application's Icon to the Icon Catalog," in the *Indigo Magic Desktop Integration Guide* for details on making your application icon appear on the chosen page.

As shown in Figure 2-13, the Icon Catalog lets users access applications visually without having to search through the file hierarchy. The Catalog contains six pages by default, which are listed below along with typical applications that appear on them:

- Application page—multimedia presentation application (*showcase*), text editor (*jot*), electronic mail program *(MediaMail*)
- Collaboration page—desktop conferencing application (*inperson*)
- Demos page—*xlogo* and *buttonfly*
- Desktop Tools page—calculator program (*xcalc*), clock program (*xclock*), select fonts (*xfontsel*), mail notifier utility (*mailbox*), calendar display (*ical*), and screen magnifier (*mag*)
- Media Tools page—compact disc player (*cdman*), digital audio tape player (*datman*), image file format conversion utility (*imgcopy*), movie creation / editing application (*moviemaker*), audio editing application *(soundeditor*)
- Control Panels page—audio control panel (*apanel*), background setting control panel (*background*), desktop settings control panel (*desktop*)

In most cases your application icon should appear on the Applications page, since that's where users will expect to find it. If you're developing a suite of applications, however, you may want to create a new page for your icons and let users know (in your product's documentation) that you've done this.

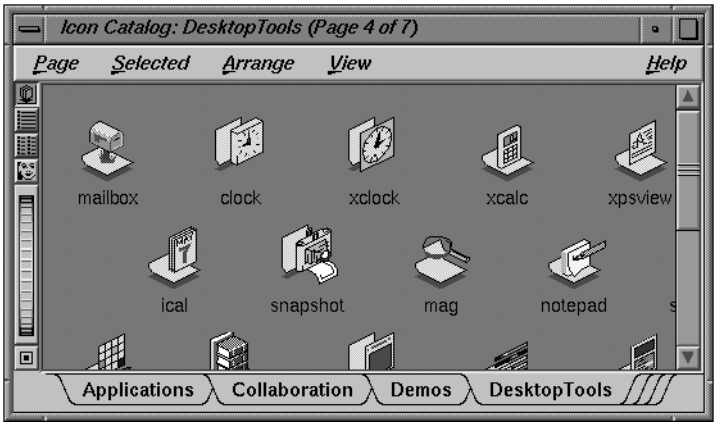


Figure 2-13 Icon Catalog

Naming and Locating Executables for the Find an Icon Tool

The Find an Icon tool, shown in Figure 2-14, allows users to find applications by typing the name of the executable. If they type the correct name and if the executable is located in a directory on the user's search path or under the user's home directory, the corresponding icon appears in the drop pocket. The user can then drag the icon to another location such as the desktop, or open it directly from the Find an Icon tool by double-clicking.

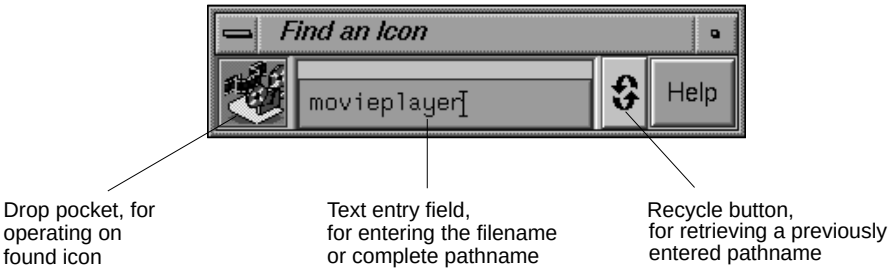


Figure 2-14 The Find an Icon Tool

Since users may be guessing at the most likely name for the application's executable, you should adhere to the following naming conventions:

- Choose a name that matches the product name or is strongly associated with the product. For example, the name of the IRIS Showcase executable is *showcase*, and the name of the online book application, IRIS InSight, is *insight*.
- Use only lowercase letters for the name. For example, the name for the SoundEditor executable is *soundeditor*.
- Don't include spaces in the name since UNIX doesn't handle spaces elegantly.
- Don't use numbers in the name to represent versions of a product. For example, the executables for IRIS InSight 2.2.1 and its earlier versions are all named *insight*.
- Don't use any special characters in the name, such as underlines or periods. Simply remove any spaces when converting your product's name to an appropriate name for your executable.

Since the Find an Icon tool searches only directories on the user's search path and under the user's home directory, your executable—or a link to it, which is preferable—should reside in one of the directories on the user's default path. The directories included in the default path are:

/usr/sbin	*/usr/bsd*	*/sbin*
/usr/bin	*/bin*	*/usr/bin/X11*

The most appropriate place to put a link to your executable—and the one that will result in users finding your application the fastest—is */usr/sbin*.

Application Icon Accessibility Guidelines

When making your application icons accessible to users . . .

- ❑ Place your application icon on the Applications page in the Icon Catalog. If you produce a suite of software applications, consider creating your own page.
- ❑ In your documentation, refer users to the appropriate page in the Icon Catalog after they've installed your application.
- ❑ When naming your executable, use the product name or choose a name that's strongly associated with the product.
- ❑ When naming your executable, use only lowercase letters. Don't use numbers, spaces, or special characters such as underlines or periods.
- ❑ Make sure that a link to your executable (preferred method) or the executable itself resides in a directory in the user's default search path. Ideally, place a link to your executable in the */usr/sbin* directory. This helps ensure that users can quickly find your application icon using the Find an Icon tool.

Chapter 3

Windows in the Indigo Magic Environment

When users run your application on the Indigo Magic Desktop, they interact with its windows through *4Dwm*, the IRIS Extended Motif Window Manager. This chapter describes the look, interactions, and behaviors that your application's windows should support. (For information on individual window components, see Chapter 6, "Application Windows," and Chapter 9, "Controls.") This chapter covers the following topics:

- "The Indigo Magic Look: Graphic Features and Schemes" discusses general characteristics of windows, including the Indigo Magic enhanced look provided by the IRIS IM toolkit and its advantages, and color and font schemes.
- "Application Window Categories and Characteristics" defines the categories of windows in the Indigo Magic Desktop environment and presents several models of applications using the various window types. It also lists the required window decorations and Window menu for each window category, prescribes how to choose labels for title bars, and discusses window size and placement issues.
- "Keyboard Focus Across Windows" establishes the *4Dwm* default keyboard focus policy across windows as implicit, and describes the behavior for applications that need to maintain control of the pointer while it's outside of the application's windows.
- "Minimized Windows" provides ideas for designing minimized window images, describes how to choose labels for minimized windows, and discusses application behavior while minimized.
- "Desks" describes the tool that provides users with multiple virtual screens or *desks*. It covers the design implications for your application windows, which can be distributed over these multiple desks.
- "Session Management" describes the *4Dwm* session manager and the implications of allowing users to log out while your application is running and return automatically to the same state upon subsequent login.

The Indigo Magic Look: Graphic Features and Schemes

When using the IRIS IM user interface toolkit, you can choose one of two different appearances for your application: either the basic OSF / Motif look, or the Indigo Magic look. The Indigo Magic look provides both an attractive, 3D look for your application and the default colors and fonts used by the Indigo Magic Desktop. There are two components to the Indigo Magic look: the graphic features and the color and font schemes.

Enhanced Graphics in the Indigo Magic Look

The Indigo Magic look contains a number of graphic modifications made to the standard IRIS IM interface. These modification improve the appearance and ease of use of applications with no impact on the component layout and minimal work on the part of the developer. Some of the differences between the Indigo Magic look and the standard IRIS IM look are illustrated in Figure 3-1. In comparison, the Indigo Magic look:

- Uses smooth shading with a rounded dimensional look to create a high-quality visual appearance. Numerous sharp bevels, such as those found in standard Motif components, detract from rather than add to the visual presentation of an application. (See Figure 3-1A.)
- Establishes the common aesthetic of burnished aluminum, giving a visually appealing look.
- Renders black outlines around stand-alone widgets to improve the readability and perception of adjoining color areas (see Figure 3-1B and C). For example, the black outline around buttons and scrollbars make them stand out from a window's background.
- Uses decals instead of stacked, 3D elements to make it easier for users to see the components (see Figure 3-1B). There is no gratuitous use of 3D such as the raised arrows and rectangular option buttons in standard Motif.
- Provides a more consolidated treatment for composite objects. For example, the Indigo Magic look visually integrates the arrow stepper buttons in scrollbars with the scrollbars themselves to give a less cluttered look. In addition, the scrollbars are visually integrated with the client pane as much as possible to make the whole assembly appear as a single, integrated unit (see Figure 3-1C).

- Enhances scrollbars by providing a grip on the slider used in scrollbars and a temporararily indented impression to indicate the original location of the slider during the scrolling process (see Figure 3-1C). The grip makes it easier for users to recognize the slider as something to be dragged rather than a button to be pressed.
- Adds additional visual feedback for selected checkboxes and radio buttons. A distinct red arrow and a blue triangle clearly indicate a selected checkbox and radio button, respectively (see Figure 3-1A).
- Adds locate highlight (the object brightens as the pointer passes over it) so that users can tell which components are live functional objects and which are passive graphics or are disabled. Locate highlight also gives users feedback as to whether or not the application is listening.
- Uses a stroked underline to indicate mnemonics in menus simply to give the look of an application some pizazz (see Figure 3-1D).

For details on how to obtain the graphic enhancements of the Indigo Magic look, see Chapter 2, “Getting the Indigo Magic Look,” in the *Indigo Magic Desktop Integration Guide*. For guidance on designing other aspects of your application windows, such as general layout principles and use of controls, see Chapter 6, “Application Windows” and Chapter 9, “Controls.”

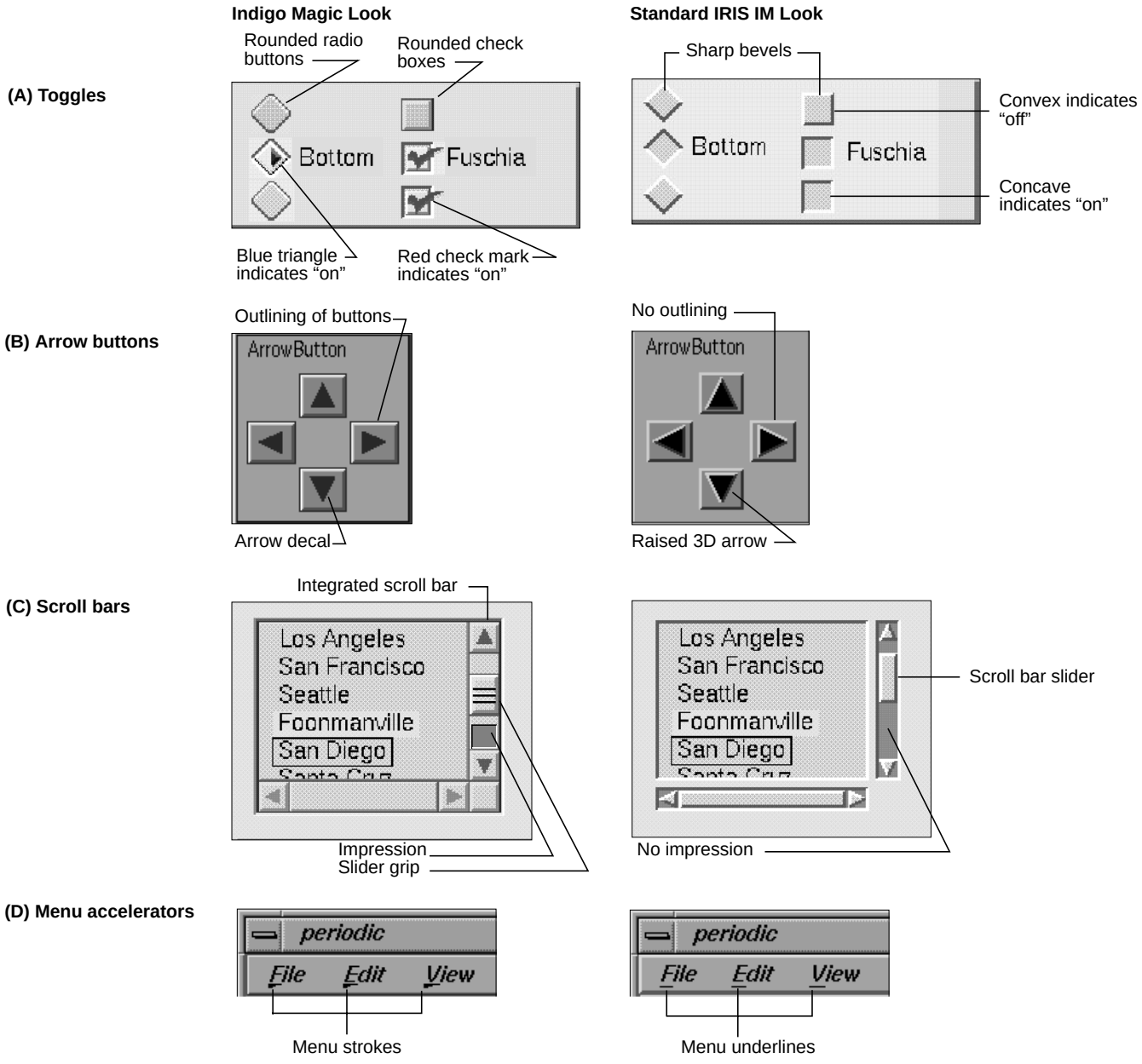


Figure 3-1 Examples of Graphic Modifications in the Indigo Magic Look

Schemes for Colors and Fonts

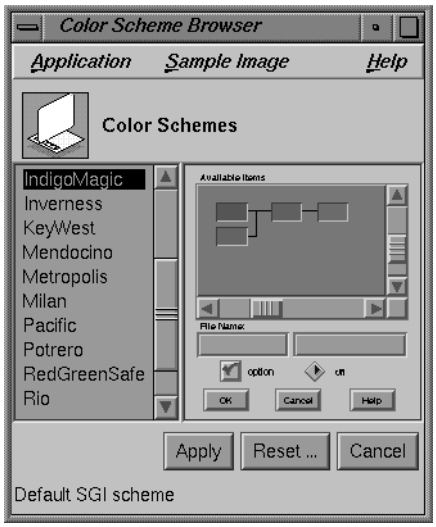


Figure 3-2
Color Scheme Control Panel

In addition to the graphic modifications described in the previous section, “Enhanced Graphics in the Indigo Magic Look,” the Indigo Magic look includes *schemes.* A *scheme* is a pre-packaged collection of colors and fonts that users can apply to application windows.

Schemes deliver several benefits to users. When all applications on a workstation use schemes, users can conveniently customize their environment. The schemes are designed with an eye to effective use of color, taking into account both usability and aesthetic considerations. Multiple schemes are provided to address such problems as red / green color-blindness, monochrome X-terminals, and user preference for light or dark text. A user changes the scheme for the desktop by selecting the new scheme from the control panel shown in Figure 3-2. (This control panel is accessed from the Desktop->Customize cascading menu in the Toolchest.)

Using schemes also benefits you as a developer by leveraging the work that has already been done on appropriate color and font choices. Using schemes eliminates worrying about different display resolutions, gamma values, user preferences, or most style guide color and font issues.

The model for using schemes as a developer is to specify color and font choices as abstract names from the predefined scheme color and font palettes instead of hard-coding specific color and font values. Then, when the user specifies a scheme, the palette entries are mapped to specific RGB color and font values.

By default, the Indigo Magic Desktop comes up in the Indigo Magic scheme. This scheme is organized around a neutral gray palette with the typographically neutral Helvetica font. Using neutral colors for standard user interface elements preserves the use of color for the application’s content areas.

Schemes are meant to apply to any region of an application window built with the standard toolkit components. Schemes should not be used for application client areas which are content specific—for example, a rendering window, a movie player, or a molecular modeler. The colors in such client areas will be application-specific and not subject to change when a user selects a new scheme.

For information on supplying schemes in your application, see Chapter 3, "Using Schemes," in the *Indigo Magic Desktop Integration Guide*.

Indigo Magic Look Guidelines

When designing the look for your application . . .

- ❑ Use the Indigo Magic look rather than the standard IRIS IM look.
- ❑ Use the pre-packaged color and font schemes supplied by Silicon Graphics rather than designing your own colors and fonts.

Application Window Categories and Characteristics

This section defines the four categories of windows in the Indigo Magic Desktop environment; presents four models of applications using these window types; and lists the decorations, Window menus, and behavior required for each window type.

Application Window Categories

The *OSF/Motif Style Guide* refers to two categories of windows: *primary windows* and *secondary windows*. The Indigo Magic environment subdivides each of these categories to yield two additional categories useful in many applications. There are two types of primary windows:

- A *main primary window* serves as the application's main controlling window. It's used to view or manipulate data, get access to other windows within the application, and kill the process when users quit. There's only one main primary window per application (and sometimes it isn't visible to users).
- A *co-primary window* is used for major data manipulation or viewing of data outside of the main window.

There are two types of secondary windows:

- A *support window* is a *persistent* special-purpose window. It typically contains a control panel or tool palette that operates directly on data in a primary window. It is used repeatedly.

- A *dialog* is a *transient* window, typically used for short, quick, user input, such as an action confirmation, or system output, as in a warning message. It may be user-requested or application-generated.

Application Models

Although there can be many combinations in an application of the four window types discussed in the previous section (main, co-primary, support, and dialog windows), most applications can be classified as fitting one of four basic models. The distinguishing factors between the models are:

- whether they can have one or multiple documents (files) open at a time
- their use of primary windows

Note: The term *document* means a grouping of data and shouldn't be thought of as simply a text-oriented file. It covers such data types as film clips, audio segments, and Inventor scenes.

These models are illustrated and discussed in more detail in "Application Models" in Chapter 6. For information about how to implement the various models, see "Implementing an Application Model" in Chapter 5 in the *Indigo Magic Desktop Integration Guide.*

"Single Document, One Primary" Application Model

"Single document, one primary" is the most basic model—it accomplishes all of its tasks within the main window and uses as many support windows and dialogs as needed. Users can work on only one document at a time. Thus, when a user has one document open and opens a second document, the second document replaces the first. IRIS Showcase operates in this manner.

"Single Document, Multiple Primaries" Application Model

The "single document, multiple primaries" model uses both main and co-primary windows to accomplish major tasks. In this model, the co-primary windows perform different functions. MediaMail is an example of this model. Its main window lets users select electronic mail messages from a folder and perform actions on them such as viewing, printing,

deleting, and sorting. Its Compose and Message windows are typical of co-primary windows with different functions designed to support the functionality of the main window. In this model, each primary window has its own menu bar tailored specifically to the functions in that window.

"Multiple Document, Visible Main" Application Model

The "multiple document, visible main" model has a main window that is mostly used to launch co-primary windows. These co-primary windows are identical to each other and perform the same functions on different files or documents. Each co-primary window has its own menu bar. IRIS InSight is an example of this model. Its main window lets users launch co-primary viewing windows, browse available files, and conduct global searches through these files. The co-primary windows are used for viewing online books.

"Multiple Document, No Visible Main" Application Model

The "multiple document, no visible main" model is identical to the "multiple document, visible main" model except that the main window is invisible to the user and new co-primary windows are launched from co-primary windows that are already open. Users open one document and leave it open while opening others. When the last open document is closed, the process is killed.

Window Decorations and the Window Menu

Users primarily interact with windows on the Indigo Magic Desktop through the window decorations and Window menu, which *4Dwm*, the IRIS window manager, places on each window. The decorations and Window menu entries vary according to the category of the window (see "Application Window Categories" earlier in this chapter) and whether any of the components in the window are resizable. Figure 3-3 shows the decorations for a typical main window. For complete details of the behavior of each of the window decorations, see Section 7.3, "Window Decorations," in the *OSF/Motif Style Guide.*

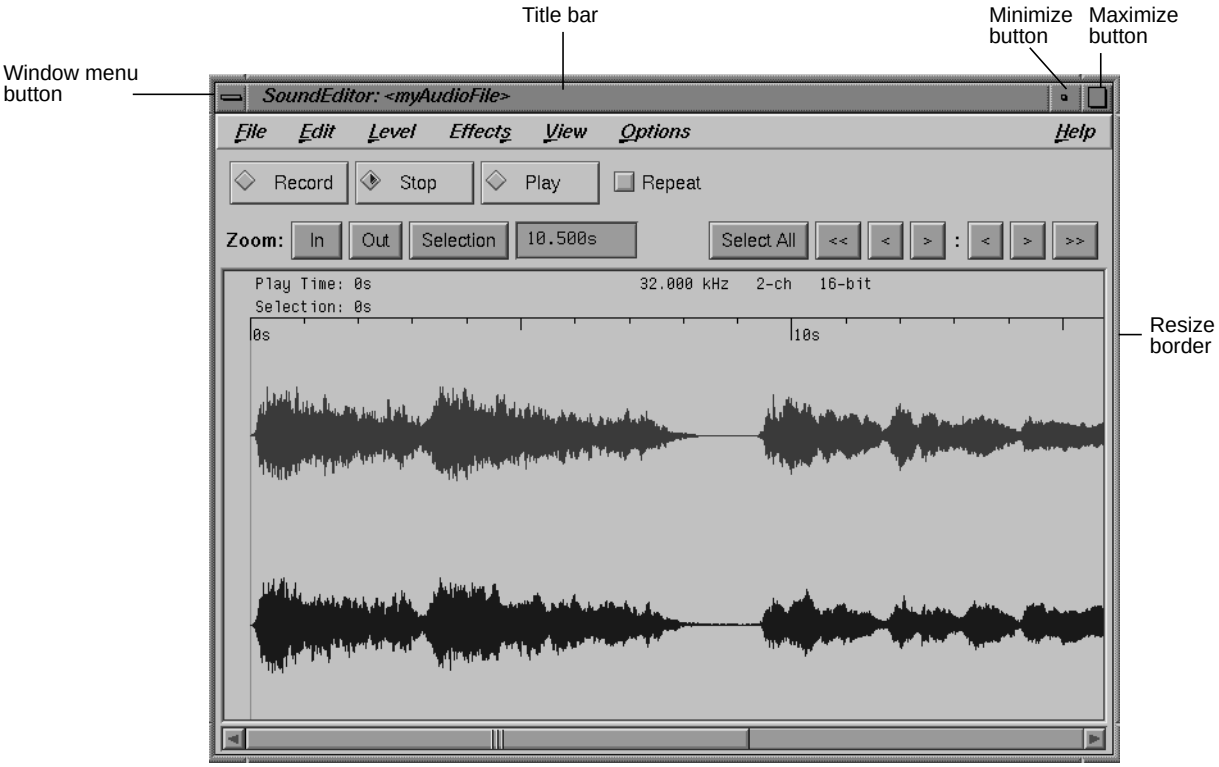


Figure 3-3 Features of a Typical Main Primary Window

The recommended decorations and Window menu entries for each category of window are shown in Table 3-1. To meet these requirements, you may have to modify both the default window decorations and Window menu entries for at least some of your application windows. For information on modifying the default window decorations and Window menu entries, see Chapter 5, “Window, Session, and Desk Management,” in the *Indigo Magic Desktop Integration Guide*. Table 3-1 also lists the keyboard accelerators and mnemonics provided by *4Dwm* for each Window menu item. These keyboard accelerators are reserved and thus should not be assigned to other functions in your application.

The behavior of the window decorations and Window menu entries is consistent with the definitions in Section 7.3 of the *OSF/Motif Style Guide*, with two notable differences:

- *4Dwm* Window menus include the entry “Raise.” “Raise” allows the user to move the window to the top of the window hierarchy, making it completely visible (in contrast to “Lower”).
- “Exit” lets users quit the application completely from a primary window. Your application must do all the appropriate cleanup work for an exit from the Window menu, such as prompting the user whether to save changes to a file. The behavior of “Exit” in the Window menu is the same as that of “Exit” in the File menu. See “File Menu” in Chapter 8 for information on the File menu. (Note that the “Close” entry on a co-primary window closes that window and any associated support windows and dialogs. It doesn’t quit the application.)

The Window menu entries are based on the functionality available for that type of window. For example, users can’t exit the application from support or dialog windows, so these window types don’t include an “Exit” entry. If a window can’t be resized, it doesn’t need the “Size” entry or the “Maximize” entry in its Window menu. (To eliminate the ability of a window to be resized, set the maximum and minimum window sizes equal to the default window size. See “Window Size” later in this chapter.) Dialogs can’t be minimized independently of their parent windows and thus don’t have a “Minimize” entry.

Table 3-1 Window Decorations and Window Menu Entries by Window Category

Window Decorations and Window Menu Entries	Main Windows	Co-Primary Windows	Support Windows	Dialogs
Window menu button	Required	Required	Required	Required
“Restore Alt+F5”[a]	Required	Required	Required[a]	Required[a]
“Move Alt+F7”	Required	Required	Required	Required

Table 3-1 (continued) Window Decorations and Window Menu Entries by

Window Decorations and Window Menu Entries	Main Windows	Co-Primary Windows	Support Windows	Dialogs
"Size Alt+F8" / Resize handles	Optional; use if user may need to expand work area or other components. If resizable, set minimum and maximum size limits.	Optional; use if user may need to expand work area or other components. If resizable, set minimum and maximum size limits.	Optional; use if user may need to expand any components, such as text input fields or scrolling lists. If resizable, set minimum and maximum size limits.	Optional; use if user may need to expand any components, such as text input fields or scrolling lists. If resizable, set minimum and maximum size limits.
"Minimize Alt+F9" / Minimize button	Required	Required	Don't use	Don't use
"Maximize Alt+F10" / Maximize button	Use only if there's a "Size" entry.	Use only if there's a "Size" entry.	Use only if there's a "Size" entry.	Use only if there's a "Size" entry.
"Raise Alt+F2"	Required	Required	Required	Required
"Lower Alt+F3"	Required	Required	Required	Required
"Close Alt+F4"	Don't use; not relevant for main windows.	Required	Required	Required
"Exit Alt+F12"	Required; closes all windows for this application and quits.	Optional; use if users can quit application from this window.	Don't use	Don't use

a. This entry always appears in the Window menu, and it's automatically disabled if there's no "Maximize" entry; this default behavior can't be changed.

Window Title Bar

By default, all windows on the Indigo Magic Desktop have a title bar that contains a label for the window. The default label used in the title bar (the application name) rarely provides enough information for users to be able to distinguish one window from another. You should label your title bars according to the rules shown below to help your users distinguish among windows belonging to the same application as well as instances of the same application. (For information on how to set the label in the title bar, see "Interacting With the Window and Session Manager" in Chapter 5 of the *Indigo Magic Desktop Integration Guide*.)

In general, use the title bar label to identify the window; don't use it to display general status (such as current page number or viewing mode) or application-critical information. Using the title bar to display information can cause problems. For example:

- The title bar may be covered by another window or off the screen.
- Users aren't accustomed to looking for status information in the title bar, so they're likely to overlook it if your application displays it there.
- It's expensive for the application to update the title bar continuously.

For more information on where to place status information or application-critical information in your application window's title bars, see "Status Areas in Primary Windows" in Chapter 6.

The label you put in the title bar is also used in the Desks Overview window (see "Desks" later in this chapter). By default, as a user moves the pointer over the thumbnail window sketches in the Desks Overview, the thumbnail window's title bar label displays as shown in Figure 3-4. (Note that users can specify that the minimized window label be shown in the thumbnail sketches instead of the title bar label.) This further emphasizes the need for users to be able to distinguish windows using only the title bar information.

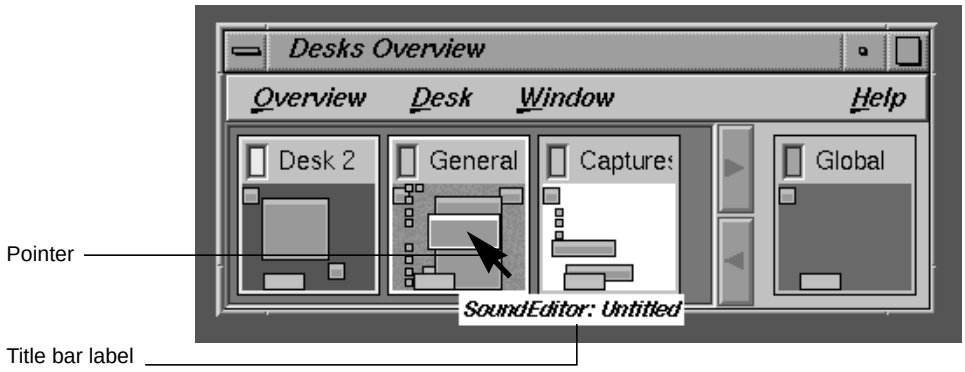


Figure 3-4 Title Bar Label Appearing in Desks Overview

Rules for Labeling the Title Bar in Main Primary Windows

The rules for labeling title bars in main windows are illustrated in Figure 3-5. First determine if your application accesses document files. If not, use just

the application name. If your application is document-based, use the application name followed by a colon (:) and the filename (or "Untitled" if it's a new file) as the label in the title bar. Note that if you include a filename in the label, you need to update the label whenever the file changes. Unless you have a real need and enough room, don't include the full pathname in the title bar. If your application is displaying remotely, use the host name followed by a colon as a prefix to the title bar label determined above. Be sure to leave spaces between strings and colons in the label. Don't use the version number in the title bar; that information should be available from the "Product Information" entry in the Help menu (see "Types of Online Help" in Chapter 4 for more information).

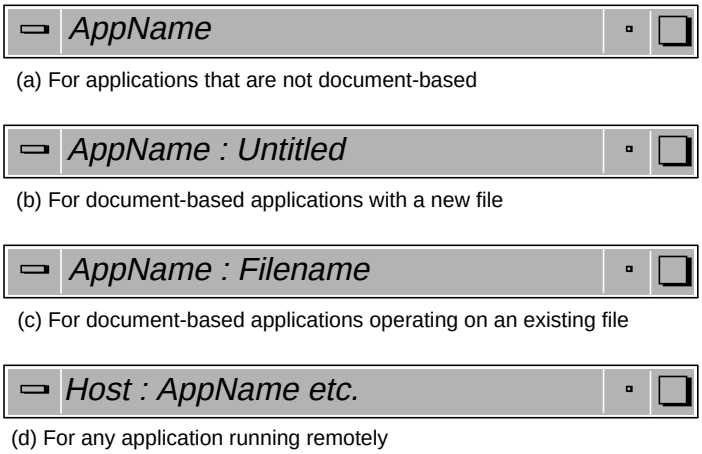


Figure 3-5 Labels for Main Window Title Bars

Rules for Labeling the Title Bar in Windows Other Than Main

For those co-primary windows that are used to supplement the main window's functionality as in the "single document, multiple primaries" application model, use the application name and function in the format *AppName : Function*. Make sure that the function closely matches the entry in the menu or the label on the button that invokes it. If the co-primary window follows a multiple document model such as the "multiple document, visible main" application model or the "multiple document, no visible main" application model, use the format *AppName : Filename* (or *AppName :* Untitled if it's a new file).

Support windows use the application name and the function in the format *AppName : Function*. Make sure that the function closely matches the entry in the menu or the label on the button that invokes it.

For dialog windows, use the application name, followed by the type of dialog in the format *AppName : DialogType*, where *DialogType* can be "Prompt," "Error," "Warning," "Question," "Information," "Working," or "File Selection." (For information on dialogs, see Chapter 10, "Dialogs.")

Window Size

The *4Dwm* window manager provides users with complete control over the size of application windows unless the application sets limits. Without a minimum window limit, users can shrink your windows to the point where they're unusable. With no maximum limit, users can expand a window to cover the full screen, potentially wasting valuable screen real estate. These extreme cases, along with a window at its default size, are illustrated in Figure 3-6. Set appropriate maximum and minimum window sizes for all of your application windows. See "Interacting With the Window and Session Manager" in Chapter 5 of the *Indigo Magic Desktop Integration Guide.*

In general, windows should be resizable only if they contain areas or components that a user might wish to resize, for example, a primary window with a resizable work area or a support window with a scrolling list or text input field. If a window does not contain resizable areas or components, then it shouldn't be resizable and you should set both the maximum and minimum size equal to the default size. Remember also to remove the Size and Maximize entries from the Window menu as described in "Window Decorations and the Window Menu" earlier in this chapter. For more information on specific window components, see Chapter 9, "Controls."

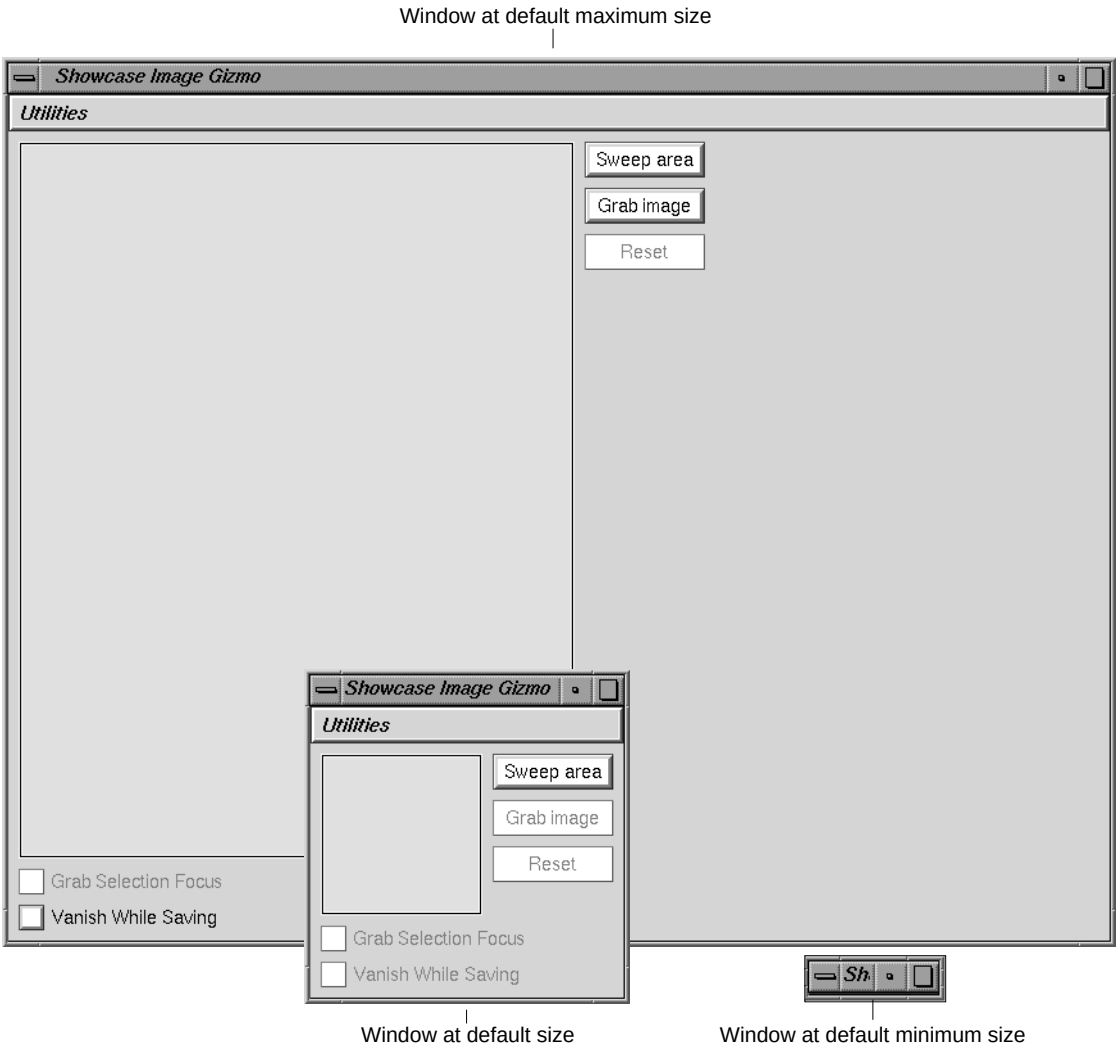


Figure 3-6 Default Maximum and Minimum Window Size Examples

Window Placement

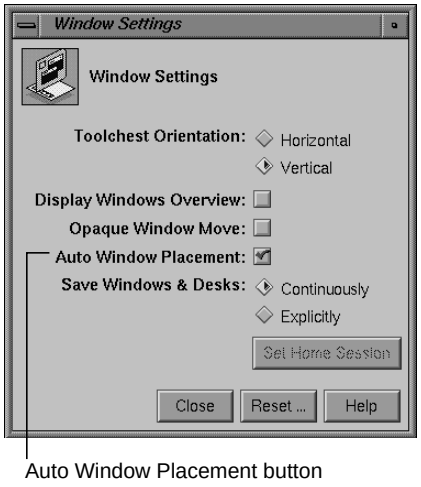


Figure 3-7
Setting Auto Window Placement

Users expect the placement of all primary windows to respond to the value of the Auto Window Placement option in the Window Settings control panel, as shown in Figure 3-7. (Users access this control panel from the Desktop->Customize cascading menu in the Toolchest.) Support windows and dialogs are always placed automatically.

When auto window placement is on (the default), *4Dwm* automatically places an application's primary windows. If a primary window does not supply any position information, *4Dwm* by default places it in the upper left corner of the screen. When auto window placement is off, users expect to be able to interactively place all primary windows. In this case, a window displays initially as a red outline attached to the pointer at its upper left corner, allowing the user to place the window manually. The user places the window by moving the outline to the desired location on the screen and clicking the left mouse button.

To take advantage of the Auto Window Placement setting, you must supply *4Dwm* with a preferred window position for each primary window rather than a required window position. With a preferred window position, when Auto Window Placement is on, *4Dwm* places the window at its preferred position. When Auto Window Placement is off, *4Dwm* ignores the preferred position, allowing the user to place the window interactively. If the window has a required position, however, *4Dwm* always tries to place the window at this preferred position even when users want to place windows themselves.

Furthermore, users expect complete control when moving windows and should be able to move any of your application's windows anywhere on the desktop. Some applications try to "help" the user by repositioning the window programmatically; this strategy is never successful and instead ends up frustrating the user.

For details on how to set a preferred window placement, see "Interacting With the Window and Session Manager" in Chapter 5 of the *Indigo Magic Desktop Integration Guide.*

Application Window Characteristic Guidelines

In general, when deciding on the characteristics for your application windows . . .

- ❑ Determine which category (main, co-primary, support, or dialog) each application window belongs to and assign characteristics appropriately.

When setting up your window decorations . . .

- ❑ Include a Window menu button for all windows.
- ❑ Include resize handles only if the window contains resizable components such as work areas, scrolling lists, and text input fields.
- ❑ Include a *Minimize* button for all primary windows. Do not include this button on support windows or dialogs.
- ❑ Include a *Maximize* button only if the window contains resizable components.

(To see the above window decoration requirements arranged according to window type, see Table 3-1.)

When designing the Window menus for your application windows . . .

- ❑ Include "Restore Alt+F5" for all primary windows. Include it for support windows and dialogs only if the menu contains a "Maximize" entry.
- ❑ Include "Move Alt+F7" for all windows.
- ❑ Include "Size Alt+F8" and resize handles for windows that contain resizable components such as works areas, scrolling lists, and text input fields.
- ❑ Include "Minimize Alt+F9" and the *Minimize* button for all primary windows. Do not include the Minimize entry for support windows or dialogs.
- ❑ Include "Maximize Alt+F10" for windows that are resizable, that is, they have a "Size Alt+F8" entry.
- ❑ Include "Raise Alt+F2" for all windows.
- ❑ Include "Lower Alt+F3" for all windows.

- Include "Close Alt+F4" for all windows except the main primary window.
- Include "Exit Alt+F12" for the main primary window. Include "Exit Alt+F12" for those co-primary windows from which users can quit the application. "Exit" always has the same behavior, that is, it quits the application, no matter how it's activated. Don't include "Exit" for support windows or dialogs.

(To see the above Window menu requirements arranged according to window type, see Table 3-1.)

- Always use the default behaviors for the Window menu entries except for "Exit." Don't add functionality to these commands. When users choose "Exit," your application must perform any necessary clean up, such as prompting the user to save unsaved changes before quitting.
- Don't add application-specific entries to this menu. Users don't expect application-specific entries in the Window menu.
- Don't add a title to the Window menu.
- Don't use the keyboard accelerators <Alt-F2>, <Alt-F3>, <Alt-F4>, <Alt-F5>, <Alt-F7>, <Alt-F8>, <Alt-F9>, <Alt-F10>, or <Alt-F12> for other functions in your application. They are reserved for the *4Dwm* Window menu entries.

When specifying the label in the title bar . . .

- For all categories of windows, limit the length of each title bar label such that the entire label displays when the window is viewed at its default size.
- Don't include application-critical information or general status information in the title bar such as the current page number or whether a file is in view-only mode.
- For main windows, first determine if your application uses document files. If it is not document-based, use the application name only. If it is document-based, use the application name followed by a colon and the filename (or Untitled if new file) in the format *AppName : filename* and update the label whenever the filename changes. Don't use the full pathname unless that information is required for users to distinguish one window from another. If your application is displaying remotely,

add the host name followed by a colon at the beginning of the title bar label in the format *Host : AppName ...* .

- For co-primary windows used in multiple document models, use the format *AppName : Filename* (or *AppName :* Untitled if a new file). For co-primary windows used in the "single document, multiple primaries" model, use the format *AppName : Function*. Make sure that the function matches the menu entry or the label on the button that invokes it.
- For support windows, use the application name and function in the format: *AppName : Function*. Make sure that the function closely matches the menu entry or the label on the button that invokes it.
- For dialog windows, use the application name, followed by the type of dialog in the format: *AppName : DialogType,* where *DialogType* is "Prompt," "Error," "Warning," "Question," "Information," "Working," or "File Selection."
- Leave spaces between strings and colons in a label.

When determining the default, minimum, and maximum sizes for your windows . . .

- Specify a default size for each window.
- If the window is resizable, specify a minimum size at which all controls and work areas will be visible and large enough to be usable. If the window is not resizable, set the minimum size equal to the default size.
- If the window is resizable, specify a maximum size such that your application window doesn't expand to fill screen space unnecessarily. If the window is not resizable, set the maximum size equal to the default size.

When considering window placement . . .

- Set a preferred window position for all primary windows. Don't set a required window position for primary windows.
- Try to anticipate other application windows that may be displayed with your application and set your preferred default position appropriately.

Keyboard Focus Across Windows

As defined in the *OSF/Motif Style Guide*, there are two types of keyboard focus (also referred to as input focus):

- *implicit*, in which the keyboard focus tracks the pointer
- *explicit*, which requires the user to explicitly select (by clicking with the left mouse button) which window or component receives the keyboard focus

The Indigo Magic Desktop uses implicit focus when moving the keyboard focus across windows. Your application should work well under implicit focus and shouldn't require users to change the default keyboard focus policy to explicit. (Note that within windows, applications should use explicit focus to move the keyboard focus between components in the window. Guidelines for using explicit focus within windows are discussed in "Keyboard Focus and Navigation" in Chapter 7.)

Certain applications need to *grab the keyboard focus*, that is, use the pointer while it is outside of the application window—for example, applications performing screen captures. This is called *pointer grab* mode.

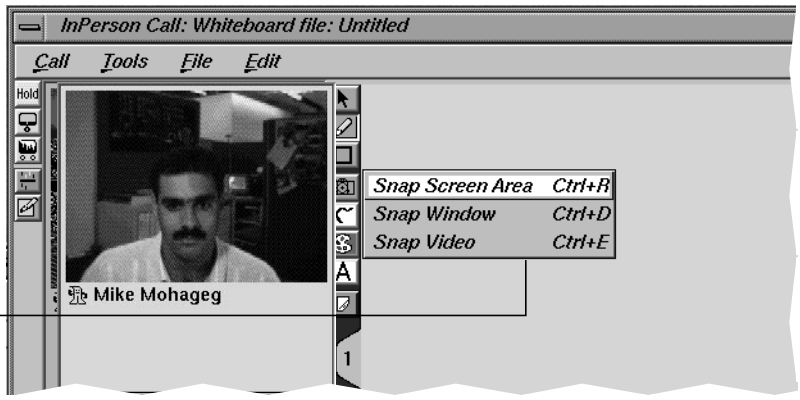
There are two recommended interaction models for pointer grab mode: *single-action*, which permits the user only one action to capture the data before returning to implicit focus; and *multiple-action*, which lets the user perform multiple actions while in pointer grab mode. In the multiple-action model, the user turns on a toggle to maintain keyboard focus while specifying the data to capture. In both the single- and multiple-action models, the application should change the pointer shape to indicate that the pointer belongs to a specific window and no longer adheres to implicit focus. (For a list of standard pointers, see "Pointer Shapes and Colors" in Chapter 11.)

Single-Action Pointer Grab Model

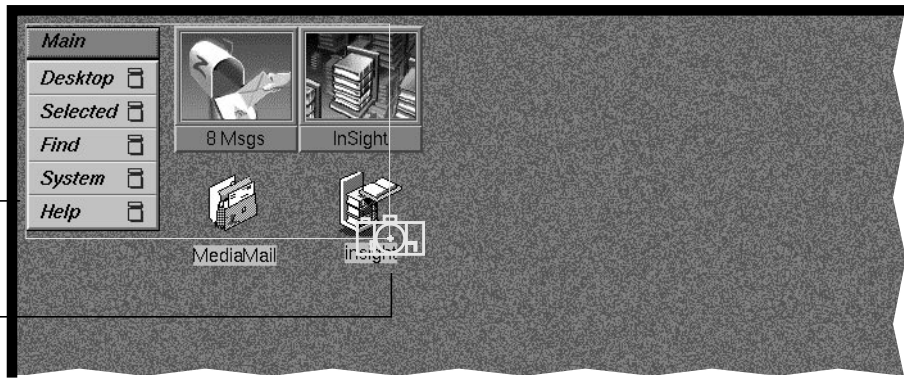
The InPerson desktop conferencing application is a good example of the single-action pointer grab model. The sequence in Figure 3-8 illustrates the single-action pointer grab where the action is to sweep out an area of the screen to be captured as an image. The user chooses the "Snap Screen Area" entry from the screen capture menu in the whiteboard's tool bar. The pointer

changes to a sighting pointer (a camera with a cross hair) and the user can then drag a rectangle around the area of the screen that the user wants to capture as an image. When the user completes the single action of dragging, InPerson releases the pointer and it is no longer in pointer grab mode. Note that in pointer grab mode, the active window is the window that has grabbed the keyboard focus, regardless of where the pointer is positioned on the screen.

Step 1 - User chooses “Snap Screen Area” from screen capture menu in InPerson window.



Step 2 - Pointer transforms to sighting pointer and user sweeps capture area.



Step 3 - User finishes drag action, which releases the pointer from pointer grab mode.

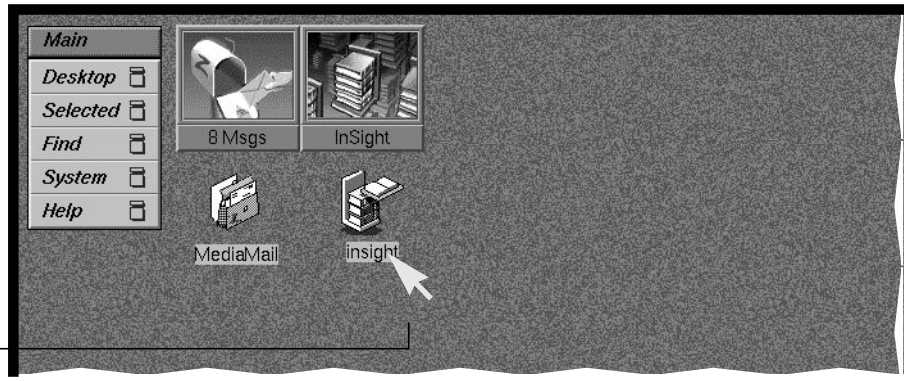


Figure 3-8 Single-Action Pointer Grab Example: Capture by Sweeping

Multiple-Action Pointer Grab Model

The IRIS Showcase Image Gizmo, used for doing screen captures, is an example of the multiple-action pointer grab model. The IRIS Showcase Image Gizmo lets users capture an area of the screen as an image, then place that image in a IRIS Showcase document. The major difference between the Image Gizmo screen capture and the InPerson screen capture described in the previous section (“Single-Action Pointer Grab Model”) is that the Image Gizmo allows a user to perform multiple actions when defining the screen capture region and not just a single action like InPerson. This allows users to grab the pointer, sweep out an area to capture, and make any adjustments to the capture area before releasing the pointer and returning to implicit focus mode.

The IRIS Showcase Image Gizmo provides an example of entering pointer grab mode. Here’s the process:

1. The user clicks the *Sweep Area* button in the Image Gizmo. This changes the pointer to a camera with a cross hair, which is used as the sighting pointer. At this point, the sighting pointer is limited to the Image Gizmo window, that is, the user hasn’t initiated pointer grab mode yet.
2. The user enters pointer grab mode by clicking the *Grab Selection Focus* button. Once in pointer grab mode, the sighting pointer is no longer limited to the Image Gizmo window. Now when the user moves the pointer out of the Image Gizmo window, the Image Gizmo retains the keyboard focus and the sighting pointer continues to display.

   Note: A better design would be to eliminate the step of requiring the user to click the *Grab Selection Focus* button and to instead have the Image Gizmo grab the keyboard focus when the user clicks the *Sweep Area* button.
3. The user drags a rectangle around the area of interest on the screen. At this point, the user is still in pointer grab mode and can redefine the area by dragging the boundaries of the current rectangle or sweeping out a completely new area.
4. The user clicks the *Grab Image* button, which releases the pointer, completes the screen capture, and returns the user to implicit focus mode.

   Figure 3-9 shows this example during pointer grab mode.

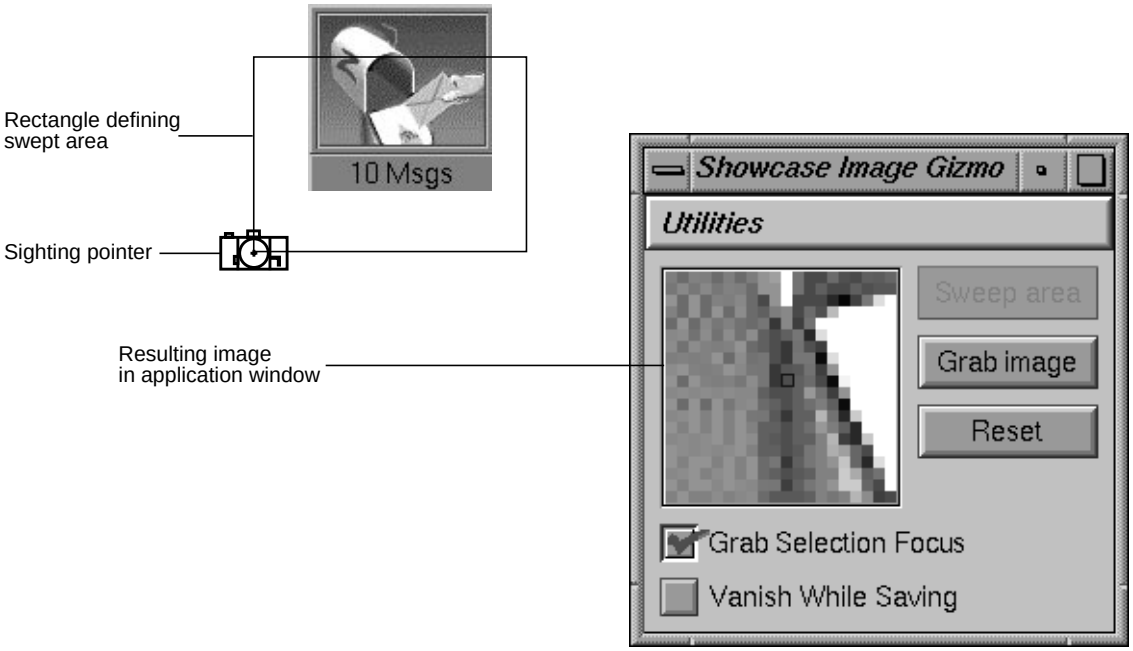


Figure 3-9 Multiple-Action Pointer Grab Example

Guidelines for Keyboard Focus Across Windows

When designing your application windows . . .

- ❑ Make sure that your application works well under implicit focus across windows.
- ❑ Don't have your application move the pointer to another location on the screen. Always allow the user to control the position of the pointer on the screen.

When incorporating a "pointer grab" function into your application . . .

- ❑ If the user is always going to specify the data to capture with a single action such as a single mouse click or a single mouse drag, use the single-action pointer grab model; otherwise use the multiple-action pointer grab model.

- Display a standard or modified sighting pointer whenever your application window grabs keyboard focus. This indicates that the keyboard focus belongs to your application's window and that the pointer isn't currently following implicit focus across windows.

Minimized Windows

Minimizing windows frees up screen real estate for other uses. On the Indigo Magic Desktop, users minimize windows by clicking the *Minimize* button in the window's title bar or choosing the "Minimize" entry from the Window menu. When a window is minimized, it's replaced by an 85x67-pixel representation with an identifying label of twelve characters or fewer. The *4Dwm* window manager determines the placement of all minimized windows.

Note that primary windows can be minimized independently of each other. Note also that dialogs and support windows become unmapped when their associated primary windows are minimized.

Choosing an Image for Your Minimized Window

It's important for users to be able to identify application windows readily when minimized. You need to define a specific image for your main window and any co-primary windows in your application. A good example in which users can easily associate the minimized window with the application appears in Figure 3-10. In the example, the IRIS InSight viewer window uses an open book as its minimized window image with the name of the book as the label.

Figure 3-10 Minimized Window Example: Good User Association With Application

If your application fits either of the single document application models discussed in "Application Models" earlier in this chapter, it should have separate images for all primary windows. If your application fits one of the multiple document models, then it should have one image for the main window and a second image for the co-primary windows.

Here are some approaches for choosing a minimized window image (see examples in Figure 3-11):

- Marketing theme—If your application has a symbol used in packaging or marketing your product, you can use some or all of it to create an image. The IRIS Showcase minimized window is an example of this approach.
- Window snapshots—If your application's main primary window layout is distinctive, you can use a snapshot of a recognizable portion of it, as in the Icon Catalog and Directory View window examples.
- Symbolic theme—You can use a symbol that reflects the nature of your application. For example, the text editor *Jot* uses an image of a writing hand. The IRIS Insight online book viewer uses an image of a stack of books to represent the main library window.
- Evocative image—You can use an image that's evocative of the function your application performs. For example, the minimized window image for the mouse control panel is an image of a mouse. The minimized window image for the background control panel is an image that uses a combination of the various background patterns available for users via this tool.

Although it is desirable to keep some family resemblance between the minimized window and other elements of an application, it's a bad idea to use a snapshot of a desktop icon as an image for a minimized window. The problem is that users could mistake the minimized window for the real desktop icon. Figure 3-12 demonstrates this problem. The minimized window at the left (faked for this example) uses a snapshot of the application icon in its open state as its image. Users could confuse the minimized window with the application icon itself. The actual minimized window appears at the right of the figure, demonstrating good design. It reuses the magician's hat theme, showing the family resemblance, but uses a different rendition of the hat to avoid confusion.

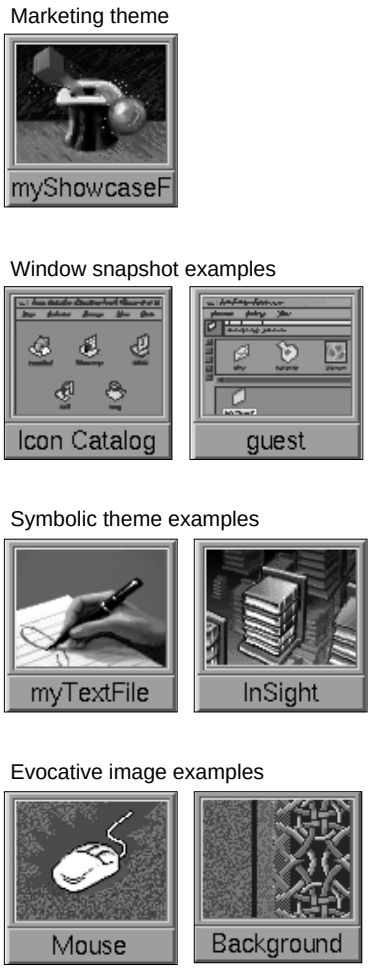


Figure 3-11
Minimized Window Examples

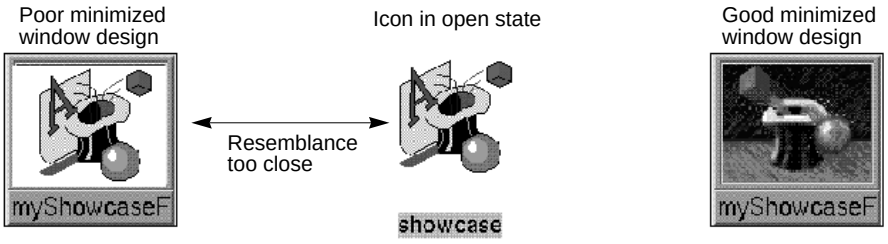


Figure 3-12 Minimized Window Example: Incorrect Design

Figure 3-13
Minimized Window Example: Design That's Too Literal

Whichever theme you choose, make sure that the significance of your image will be grasped in foreign countries and will not offend international users. Images that are too literal will not be understood by an international audience. For example, the minimized window in Figure 3-13 is for a debugging application and uses an image representing dead bugs. This may not be readily apparent to some non-English speaking users.

For information on creating and implementing minimized window images, see Chapter 6, "Customizing Your Application's Minimized Windows," in the *Indigo Magic Desktop Integration Guide*

Labeling a Minimized Window

By default, the *4Dwm* window manager reuses the title bar label for the minimized window label. (The guidelines for specifying title bar labels are discussed in "Window Title Bar" earlier in this chapter.) This doesn't usually work due to the space limit (approximately twelve characters) on the minimized window label. Thus, you will need to specify a label for each of your minimized windows.

Those applications that aren't document-based should use the application name as the minimized window label for the main window. Any minimized co-primary windows for these applications should use the label *Function*, where *Function* is the same function as in the co-primary window's title bar.

Applications that are document-based and follow the single-document models (see "Application Models" earlier in this chapter) should use *Filename* (or "Untitled" for new files) for the minimized main window label. Any co-primary windows for these applications should use *Function* for the

minimized window label, where *Function* is the same function as in the co-primary window's title bar.

Applications that are document-based and follow the multiple-document models (see "Application Models" earlier in this chapter), should use the application name as the label for the main window (if this main window is visible). The co-primary windows in these models represent the multiple documents and should have the minimized window label *Filename* (or "Untitled" for new files).

The minimized window label is also used in the Desks Overview window (see "Desks" later in this chapter). The user can customize the Desks Overview so that moving the pointer over the thumbnail window sketches in the Desks Overview displays the minimized window labels for those windows. This further emphasizes the need for users to be able to distinguish windows using only the minimized window label.

For more information on specifying a label for a minimized window image, see Chapter 6, "Customizing Your Application's Minimized Windows," in the *Indigo Magic Desktop Integration Guide*.

Processing While Minimized

Users generally expect an application to continue processing while its windows are minimized; when re-opened, the window's contents should have changed appropriately. Of course, it doesn't make sense for all types of functions to continue processing while the window is minimized. For example, you needn't keep moving a clock application's hands while it is minimized. It's up to you to determine which functions are appropriate for continued processing and which are inappropriate. Be sure to stop those functions that don't need to process as they can be a drain on CPU resources.

Using the Minimized Window to Show Status

If it is typical for users to minimize your application's windows while processing continues, you may wish to use your minimized application window to indicate status. Figure 3-14 shows how the minimized window

label can be used to indicate status in an electronic mail application by showing the number of messages in a mail folder.

Figure 3-14 Minimized Window Example: Indicating Status With the Label

It is also possible to change the minimized window image to show status, however this is quite difficult. For more information on changing minimized window labels and images to show status, see Chapter 6, "Customizing Your Application's Minimized Windows," in the *Indigo Magic Desktop Integration Guide*.

Minimized Window Guidelines

When designing images for your minimized primary windows . . .

- ❑ Use a color image rather than a two-color bitmap.
- ❑ Design your images to look best at the default size of 85x67 pixels.
- ❑ If your application is based on a single document model, create separate images for each of the primary windows. If your application is based on a multiple document model, create one image for the main window and a second image to use for all co-primary windows.
- ❑ Choose images that clearly identify the window that is minimized. If you have multiple images, make sure that the separate images work well together.
- ❑ Make sure that the images you use for minimized windows will be understood by an international audience.
- ❑ Don't use a snapshot of the desktop icon for the image. This could be confused with the real icon.

When choosing labels for your minimized primary windows . . .

- Limit the label to approximately twelve characters. If you need a few more characters than this, check that your label will fit with the default size and font for minimized windows.
- If your application is not document-based, use the application name as the minimized window label for the minimized main window. Use the label *Function* for minimized co-primary windows where *Function* is the same function as in the co-primary window's title bar.
- If your application is document-based and follows one of the single-document models, use *Filename* (or "Untitled" for new files) for the minimized main window label. Use *Function* for minimized co-primary window labels where *Function* is the same function as in the co-primary window's title bar.
- If your application is document-based and follows one of the multiple-document models, use the application name as the label for the main window (if it is visible). The co-primary windows in these models represent the multiple documents and should have the minimized window label *Filename* (or "Untitled" for new files).

When determining the behavior for a window that the user has chosen to minimize . . .

- Decide which operations should and should not continue to be processed while the window is minimized.
- Indicate status with the minimized window label if your application is typically minimized during long processes.
- Use the default screen locations supplied by *4Dwm* for the minimized window. Don't specify your own screen location.

Desks

The Indigo Magic Desktop provides users with a handy tool called Desks Overview[1] for organizing their work (see Figure 3-15). The Desks Overview application allows users to create multiple virtual screens (desks). The user can place any primary window (main or co-primary) on any desk. The window appears in the thumbnail sketch in the Desks Overview window. Support windows and dialogs don't appear in these thumbnail sketches.

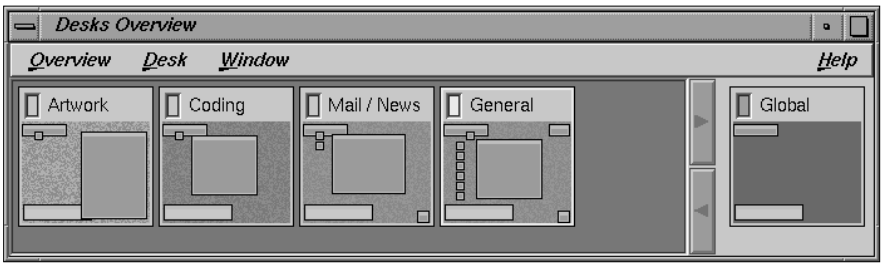


Figure 3-15 Desks Overview Window

There are several things you need to know about desks and the corresponding effects on the design of your application.

- *4Dwm* treats application windows on desks other than the current one as if they are minimized. The windows are no longer mapped to the screen display and the application is informed that it's unmapped. This emphasizes how important it is for you to decide which operations continue processing when the application is in an unmapped (minimized) state. (See "Processing While Minimized" earlier in this chapter.)

[1] Due to a software patent dispute instituted by Xerox Corporation, the Desks and Desks Overview features of this version of the IRIX operating system are now optional and may or may not be available to you after May 15, 1995. On this date, these features will be disabled unless you have entered a license code obtained from Silicon Graphics. When the Desks Overview feature is initiated, a dialog box will periodically warn you of the May 15, 1995 expiration date. Depending on how this patent dispute is resolved, prior to May 15, 1995, Silicon Graphics will attempt to inform you that the license code for these features will: (1) not be available; (2) only be available at an additional price; or (3) be available free of charge. Silicon Graphics apologizes for this inconvenience and appreciates your understanding. For updates, see comp.sys.sgi.admin, or call 1-415-390-4334. Please do not send email, or call other SGI numbers.

- As users move the pointer over the miniaturized window representations in the thumbnail sketches, the title bar labels display by default. If the user prefers, the minimized window labels can be displayed instead. This further emphasizes the need to pick title bar labels and minimized window labels so that a user can distinguish windows using only this information. (For information on defining these labels, see “Window Title Bar” and “Labeling a Minimized Window” earlier in this chapter.)
- Support windows and dialogs will appear on all desks if their associated parent window is not mapped to the screen. Since support windows and dialogs should only appear on the desk where their parent appears, make sure that all parent windows are visible and mapped to the screen.
- Your application should not create a screen background. Screen backgrounds are managed by *4Dwm* by default or by the user through the Background control panel. Users typically employ different screen backgrounds on different desks as an aid in orienting themselves.

Desks Guidelines

When designing your application . . .

- ❑ Make sure that all windows with associated support or dialogs are visible and mapped to the screen so that the support windows and dialogs appear only on the desk where their parent window displays.
- ❑ Don’t design your application to manage the screen background.

Session Management

Session management allows users to log out of their accounts and have any running applications automatically restart when they log back in, thus eliminating the need for users to restart applications manually when they log back in. In *4Dwm*, users have the option of turning session management on (the default) or off by using the Window Settings control panel (which is available from the Desktop->Customize cascading menu in the Toolchest), as shown in Figure 3-16.

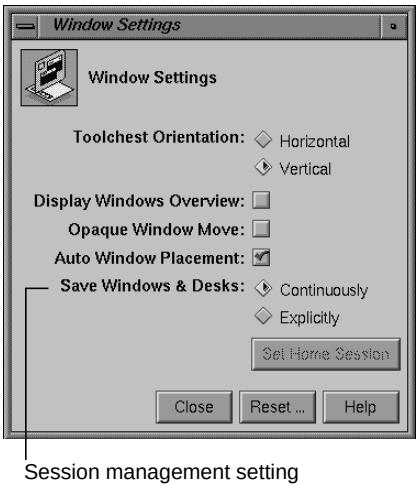


Figure 3-16
Setting Session Management

For your application to be restarted via the *4Dwm* session manager, your application needs to create a command line that will launch the application and restore its current state. Your application needs to update this command line as the application state changes. For details of specifying this command line and keeping it up to date, see "Interacting With the Window and Session Manager" in Chapter 5 of the *Indigo Magic Desktop Integration Guide*. The following scenario outlines the process.

When a user launches an application, that application registers itself with the *4Dwm* session manager by creating a command line that will launch the application and restore its current state. For example, if your application is used to edit a specific file, the command line should contain the information necessary to launch your application and open this specific file. There is only one command line per application.

As the state changes over time, your application needs to update this command line. So, to continue the example, suppose the user opens a different file to edit under your application. In such a case, your application needs to create a new command line that will launch the application and open this new file.

If the user opens co-primary windows or support windows (see "Application Window Categories and Characteristics" earlier in this chapter for window definitions), these windows should also redisplay when the user logs out and back in again. (Dialogs typically don't redisplay.) One method for obtaining this behavior is to allow your application to take command line arguments to redisplay these windows so that you can include these arguments on the stored command line when appropriate.

When the user logs out, *4Dwm* saves the command lines for all applications that are currently running on the user's desktop. When the user logs back in, *4Dwm* attempts to execute the commands that it saved and restore the user's desktop to what it was when the user logged out.

Applications can also request to have *4Dwm* inform them when a user chooses "Log Out." When applications receive this notification, they should not post any dialogs such as "Save unsaved changes before quitting?" Instead, if *4Dwm* notifies your application that the user is logging out and there are unsaved changes for the current file, your application should use one of the following strategies:

- Save these changes into another file and name it something logical such as *original_file_name*.save. When the application is restarted at login, post a dialog that tells the user that this file with unsaved changes exists and query the user whether to open the original file or the file with the unsaved changes. This is the preferred strategy.
- Ignore the user's unsaved changes, and simply restart the application using the most recent saved version of the file. This strategy is okay, but it is not preferred.

Don't automatically save the user's changes by default. This may cause the user to lose as much data as throwing away all unsaved changes. Let the user control when changes are saved.

For details on how to request log-out notification from *4Dwm*, see "Interacting With the Window and Session Manager" in Chapter 5 of the *Indigo Magic Desktop Integration Guide*.

Session Management Guidelines

When designing your application . . .

- ❑ Have your application create a command line that will launch the application and restore its current state. This current state should minimally include reopening any files that are currently open under the application and opening any primary or support windows that are currently open.
- ❑ Update this command line as the state of the application changes.
- ❑ If your application allows users to create and edit data files, have *4Dwm* notify your application when the user chooses "Log Out."

If your application is running when the user chooses “Log Out” and there are unsaved changes for a specific file . . .

- Save these changes into another file and name it something logical such as *original_file_name*.save. When the application is restarted at login, post a dialog that tells the user that this file with unsaved changes exists and query the user to determine whether to open the original file or the file with the unsaved changes.
- If you cannot implement the preferred strategy described above, ignore the user’s unsaved changes. Do not automatically save the user’s changes by default.

Chapter 4

Indigo Magic Desktop Services

Indigo Magic provides desktop services you can take advantage of in your applications. These services save you time by providing common functionality that you don't have to develop on your own. Using the services can also help ensure that your application is consistent with other applications in the Indigo Magic environment. This chapter covers the following topics:

- "Software Installation" describes how users install, remove, and upgrade applications using Software Manager, an Indigo Magic Desktop utility.
- "Online Help" describes SGIHelp, the standard online help system on all Silicon Graphics platforms.
- "Online Documentation" discusses IRIS InSight, an online documentation delivery system available on all Silicon Graphics platforms.
- "Desktop Variables" describes certain customization settings that users can choose and their implications for your application.
- "File Alteration Monitor (FAM)" describes the FAM service, which informs your application about ongoing changes to the file system, eliminating the need for your application to do its own polling.

Software Installation

Users install software on Silicon Graphics workstations using the Software Manager, a graphical tool for installing, removing, and tracking software (see Figure 4-1). For your application to work with the Software Manager, you must package your files into software images that it will understand. This packaging process is simplified by the Software Packager tool, which is described in the *Software Packager User's Guide*.

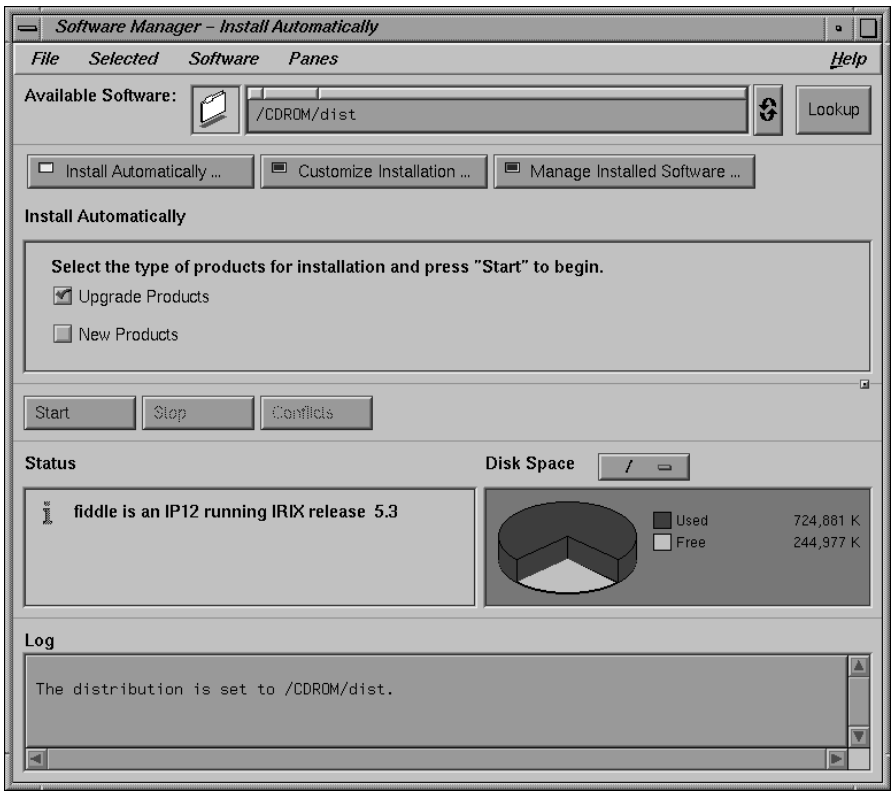


Figure 4-1 The Indigo Magic Software Manager

Although you can create installation and removal scripts independently of the Software Manager, it's strongly recommended that you package your application to use the Software Manager format. The advantages of the Software Manager are:

- It gives users a single, graphical tool to install all of their applications, upgrades, and maintenance releases. Users won't have to learn how to use additional tools, read lengthy installation instructions, or enter commands in a UNIX shell.
- It allows users to remove their applications cleanly. Users can remove your application and all supporting directories and files using Software Manager. They don't have to rely on specialized removal instructions or guess which directories and files they should remove.
- It helps users upgrade applications cleanly. When the user installs an upgrade, the Software Manager automatically removes previous versions of that application.
- It provides installation status information. Users can query the Software Manager database to quickly obtain information such as whether the application is installed, when it was installed, how much disk space it uses, and whether any upgrades or maintenance releases have been installed. The Software Manager doesn't track this information for applications that were installed using specialized scripts.

Software Installation Guideline

❑ Make sure that users can install and remove your application through the Software Manager, an Indigo Magic Desktop utility.

Online Help

Your application should provide online help. In the Indigo Magic environment, users are accustomed to specific kinds of online help information, including context-sensitive help (letting users click on window areas and components to get specific help) and task-oriented help (presenting help information for specific tasks that can be performed using the application).

Providing Help Using SGIHelp

SGIHelp is the online help system available on all Silicon Graphics platforms. It provides an easy method for delivering help information to users. Although you can supply your own help system, it's strongly recommended that you use SGIHelp. The advantages of supplying SGIHelp with your application are:

- Users like it—it's easy to use and convenient. SGIHelp provides context-sensitive help and task-oriented help. It also allows users to get help information by browsing and searching an index of available help topics and by following cross-references (links) to related topics.
- Silicon Graphics users are already familiar with it. They may get frustrated if they have to learn a different help system for your application, particularly when they're looking for help.
- It eliminates the need for creating and maintaining your own help system. Many of the help capabilities that your application should support (which are discussed in the next section, "Types of Online Help") are provided automatically when you use SGIHelp.

SGIHelp is a full-featured system that provides users with: fast, direct access to any help topic, the ability to navigate forward and backward through cross-referenced help topics, an index of help topics, search capabilities, and convenient printing of help information. SGIHelp is also a multimedia tool—you can include inline images, 3D objects, and audio clips, and you can launch applications from it. For details on how to include SGIHelp in your application, see Chapter 9, "Providing Online Help With SGIHelp," in the *Indigo Magic Desktop Integration Guide*.

Types of Online Help

The keys to supplying useful online help are to anticipate users' questions and to provide them with easy access to the answers to those questions. Each window in your application should include a Help menu if the window has a menu bar or a *Help* button if the window doesn't have a menu bar.

Figure 4-2 demonstrates how users access online help from a Help menu. A typical Help menu showing the various types of information appears in the upper left of the figure. The SGIHelp windows available from the "Overview," "Index," and "Keys & Shortcuts" menu entries are shown in clockwise order around the Help menu.

Online help falls into six general categories:

- context-sensitive information
- overview information
- task-oriented information
- index of help topics
- keyboard shortcut information
- product information

These categories are defined and discussed in the following paragraphs.

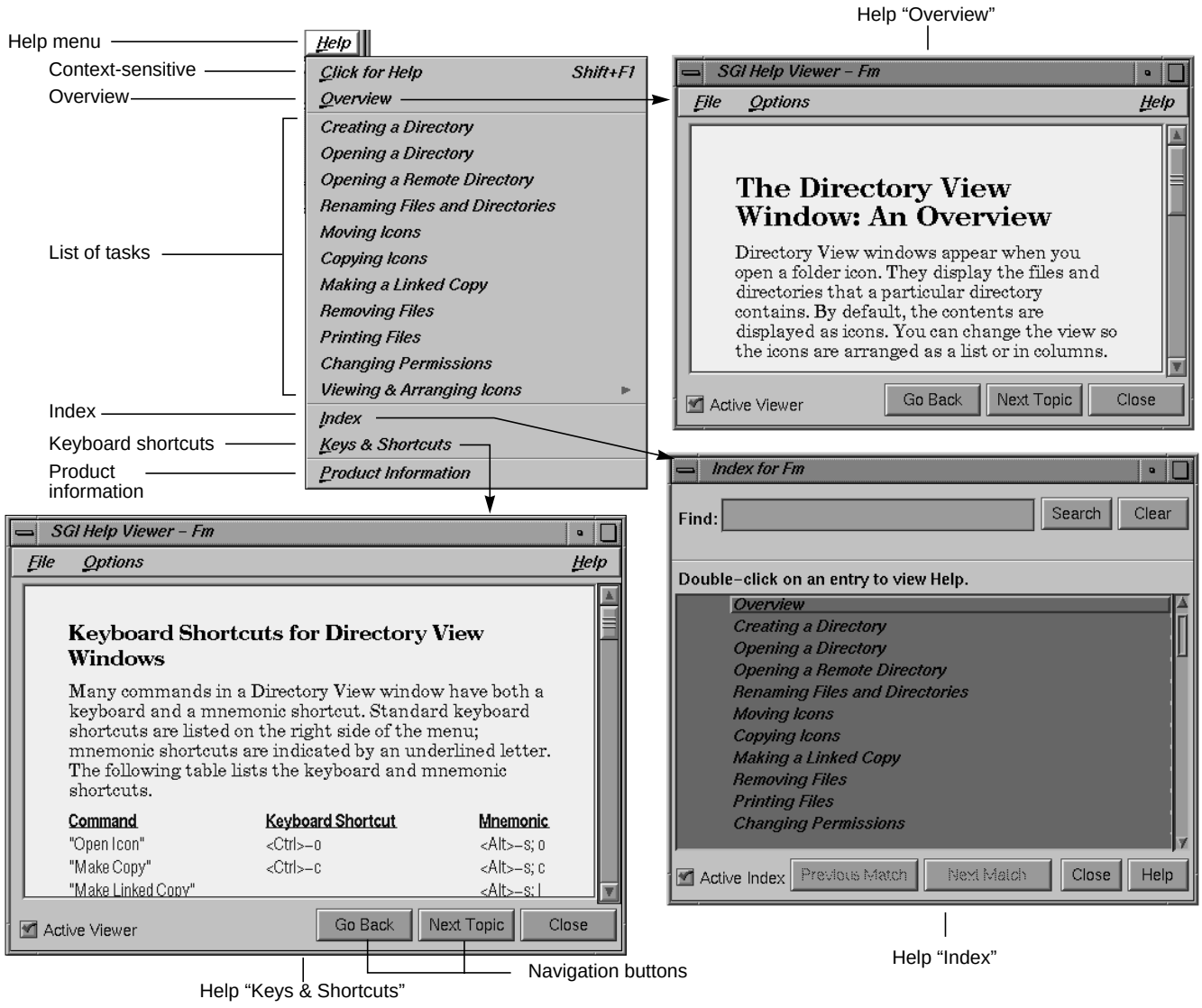


Figure 4-2 Typical Help Menu and Related Windows

Context-Sensitive Information

Context-sensitive information answers the question “What is the purpose of this area or component in this window?” It should be available for all primary and support windows. Context-sensitive help mode should be enabled when the user either chooses the “Click for Help” entry in a Help menu (if the window has a menu bar) or presses <Shift>-<F1> (whether or not the window has a menu bar).

Once the user has enabled context-sensitive help mode for a particular window, the pointer should change to a question mark and the user should be able to click in any area of the window to get specific help information (see Figure 4-3). At a minimum, your application should provide separate context-sensitive help for each control area, work area, status area, and menu in the window. This help should describe the purpose of the corresponding area and should include cross-references to task-oriented help topics which describe tasks which use this area.

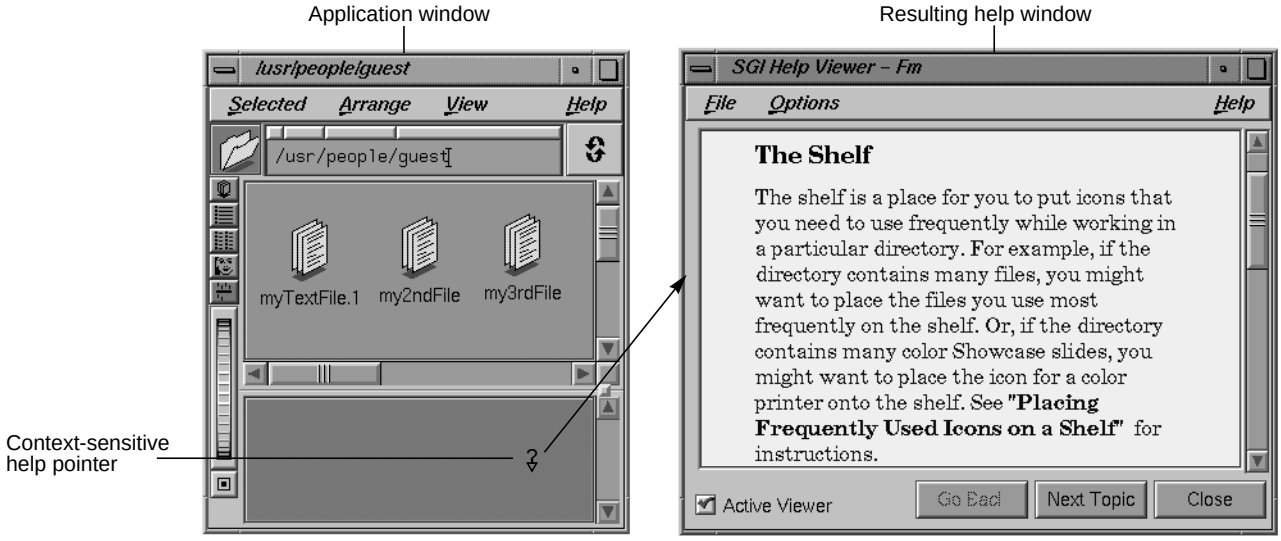


Figure 4-3 Context-Sensitive Help Example

Overview Information

Overview information answers the question "What does this application or window do?" All main windows should provide overview help information regardless of whether help is provided from a menu or a button. Co-primary and support windows with menu bars should also provide this information. If the window has a menu bar, the overview information should be accessible from an "Overview" entry in the Help menu. For main windows that don't have a menu bar, this overview information should be contained in the help that's displayed when the user clicks on the Help button in the window.

For main windows, the overview information should briefly describe the functionality of the entire application. For co-primary and support windows, the overview should describe the functionality of the current window. It can also provide cross-references to task-oriented information. An example Overview help topic is shown in Figure 4-2.

Task-Oriented Information

Task-oriented information answers the question "How do I accomplish a specific task?" It also serves to give users a quick overview of your application; users often scan a new application's menus—especially the Help menu—to get an idea of the application's functionality.

All windows should provide task-oriented help. Windows with a menu bar should provide Help menu entries for each of the most important tasks that users can accomplish in that particular window. When a user chooses any of these entries, the resulting help topic should present task-oriented information that describes step-by-step instructions for accomplishing the given task. A typical list of tasks is shown in the Help menu for Figure 4-2. For windows without a menu bar, the task-oriented information should be displayed when the user clicks on the Help button in that window. This help information should include step-by-step instructions for accomplishing all of the tasks in that specific window.

Index of Help Topics

The *index of help topics* answers the question "What help topics are available for this application?" This index should be available from all windows with

a menu bar and should be displayed when the user chooses the “Index” entry from the Help menu. It should list all available help topics for the application, including those that are generated using the context-sensitive help mode and those that are available directly from the Help menu. Users should be able to browse the index and select individual help topics as an alternative to the other methods of accessing help. See Figure 4-2 for an example index.

Keyboard Shortcut Information

Keyboard shortcut information answers the question “How can I use the keyboard as a shortcut for performing specific actions?” This information should include all mnemonics, keyboard accelerators, and function keys available for the entire application (not just those for a specific window). An example keyboard shortcuts help topic is shown in Figure 4-2.

All main windows should provide information on keyboard shortcuts. Co-primary and support windows with menu bars should also provide access to this information. If the window has a menu bar, the keyboard shortcut information should be accessible from a “Keys & Shortcuts” entry in the Help menu. For main windows that don’t have a menu bar, this shortcut information should be contained in the help that’s displayed when the user clicks on the Help button in the window.

Product Information

Product information answers generic questions about the application. At a minimum, it should identify the application and version number, but it can also include general product information such as copyright and trademarking, licensing, and customer support access. All main windows should provide access to this information. Co-primary and support windows with menu bars should also provide access to this information.

If the window has a menu bar, the product information should be accessible from a “Product Information” entry in the Help menu. For main windows that don’t have a menu bar, this product information should be contained in the help that is displayed when the user clicks on the Help button in the window.

Note that product information should always be provided using an Information dialog rather than through the SGIHelp system. Some users don't install online help to save disk space, so using a dialog for this information allows more users to access the application's version number and customer support contact information. See "Other Situations for Invoking Dialogs" in Chapter 10 for information about how to design an appropriate product information dialog.

Providing Help through a Help Menu

All windows that have a menu bar should provide a Help menu. (See "Standard Menus" in Chapter 8 for more information on standard menus.) Help menus have the general layout, mnemonics, and keyboard accelerators shown in Figure 4-4. The entries are divided into four groups. The list of tasks appears between the "Overview" and "Index" entries and is specific to the application.

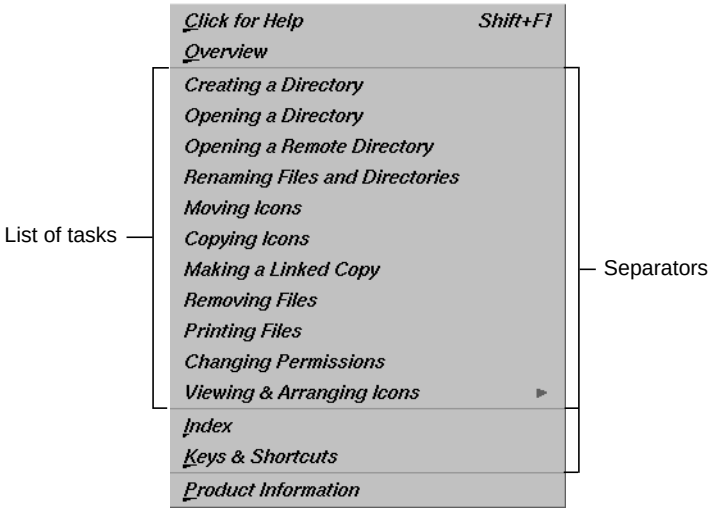


Figure 4-4 Typical Help Menu

Help menus should contain these elements, in the order presented:

- "Click for Help" provides context-sensitive information for the current window. It uses <Shift+F1> as the keyboard accelerator. (Note that this

differs from the *OSF/Motif Style Guide* which recommends using the label “Context-Sensitive Help” for this entry.)

- “Overview” provides an overview of the entire application when it appears in the Help menu for the main window. Otherwise, it provides an overview of the functionality of the current window and should be labeled “Overview for <*window name*>”.
- The list of tasks provides menu entries for those tasks that can be performed in the current window. This task-oriented information should include step-by-step instructions for how to accomplish the specific task. If you have more than ten or twelve or task entries, consider using a cascading menu such as the “Arranging Icons” entry in Figure 4-4. Cascading menus are described in “Using Cascading Menus” in Chapter 8. These entries shouldn’t have mnemonics or keyboard accelerators.
- “Index” displays a list of all help topics available for the application and allows users to choose topics from this list for viewing.
- “Keys & Shortcuts” displays all mnemonics, keyboard accelerators, and function keys available for the entire application and not just those for a specific window. (Note that this differs from the *OSF/Motif Style Guide* which recommends using the label “Keyboard” for this entry.)
- “Product Information” identifies the application and version number. Additionally, this entry can provide general product information such as copyright and trademarking, licensing, and customer support access.

Providing Help Through a Help Button

If an application window doesn’t have a menu bar, it should provide a *Help* button for accessing online help. When users click this button on a main window, they should get information that includes an overview of the functionality of the application (overview), step-by-step instructions for how to perform all of the tasks in this main window (task-oriented), a list of all keyboard shortcuts for the application, and the version number, copyright, licensing, and customer support access information for the application (product information).

When users activate the *Help* button in a co-primary or support window, they should get information that describes the function of the specific window (overview), and step-by-step instructions for how to perform all of the tasks in this window (task-oriented). For dialogs, activating the *Help* button should provide help that focuses on the main purpose of the dialog, describes how to use the dialog, and might also provide pointers to additional information, especially in the case of error dialogs.

Both primary and support windows with *Help* buttons should also provide context-sensitive help when the user presses <Shift>-<F1>. Dialogs typically don't support context-sensitive help mode.

A typical window with a Help button appears in Figure 4-5. For specific information on laying out pushbuttons in windows, see "Control Areas in Primary Windows" in Chapter 6 and "Decorations, Initial State, and Layout of Dialogs" in Chapter 10.

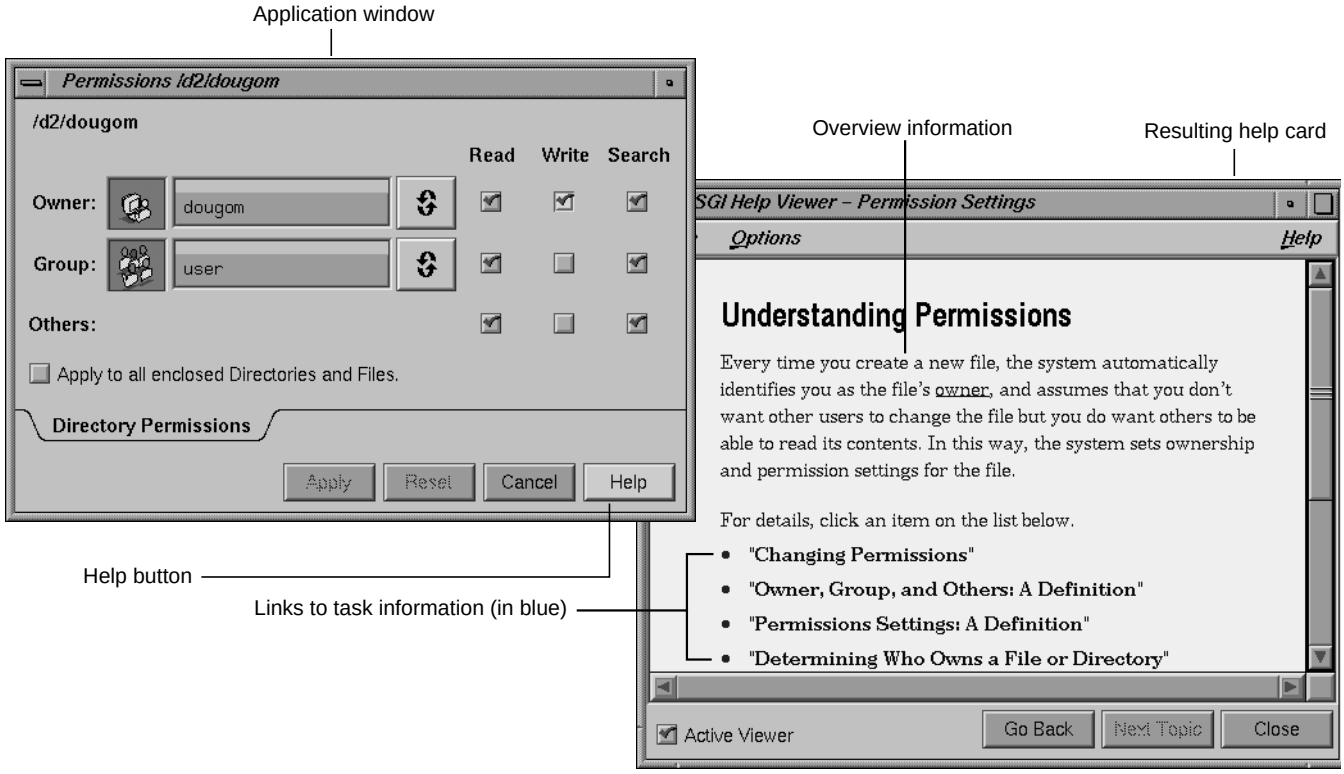


Figure 4-5 Help Button Example

Guidelines for Designing Online Help

When designing access to online help for your application . . .

- ❑ Provide access in each window of your application from either a Help menu if the window has a menu bar or a *Help* button if the window doesn't have a menu bar.
- ❑ Use SGIHelp. This provides users with a familiar viewer and familiar navigation techniques when reading the online help for your application.

When defining the types of online help for your application . . .

- ❑ Provide context-sensitive help, overview information, task-oriented help, a list of keyboard shortcuts, product information, and an index of help topics.
- ❑ Provide context-sensitive help for all primary and support windows.
- ❑ Enable context-sensitive help mode when the user either chooses the "Click for Help" entry in a Help menu (if the window has a menu bar) or presses <Shift>-<F1> (whether or not the window has a menu bar). Change the pointer to a question mark when context-sensitive help mode is enabled.
- ❑ At a minimum, provide separate context-sensitive help for each control area, work area, status area, and menu in the window. This help should describe the purpose of the corresponding area and should include cross-references to task-oriented help topics which describe tasks which use this area.
- ❑ Provide overview information for all main windows whether help is provided from a menu or a button. This overview should briefly describe the functionality of the entire application.
- ❑ For co-primary and support windows that include a menu bar, provide overview information that describes the functionality of that specific window.
- ❑ Provide task-oriented information for all windows. This information should include step-by-step instructions for how to accomplish all of the tasks available in the current window.
- ❑ For windows with a menu bar, provide access to an index of help topics. This index should list all available help topics for the application including those that are generated using the context-sensitive help mode and those that are available directly from the Help menu. In addition, users should be able to browse the index and select topics for reading.
- ❑ Provide keyboard shortcut information for all main windows (whether help is provided from a menu or a button) and for co-primary and support windows that include a menu bar. This information should include the mnemonics, accelerators, and function keys available for the entire application and not just for the current window.

- ❑ Provide product information for all main windows (whether help is provided from a menu or a button) and for co-primary and support windows that include a menu bar. This information should minimally include the product name and version number. It might also include other general product information such as copyright and trademarking, licensing, and customer support access.
- ❑ Display product information using an Information dialog so that users who don't install an application's online help can still access version number and customer support information.

When providing a Help menu in an application window . . .

- ❑ Include a "Click for Help" entry to enable context-sensitive help mode with the keyboard accelerator <Shift>-<F1>.
- ❑ Include an "Overview" entry for main windows. For co-primary and support windows, include an entry labeled "Overview for <*window name*>".
- ❑ Include entries that represent a list of tasks that users can accomplish in the current window. If this list of tasks is more than ten or twelve entries, use cascading menus. These entries shouldn't have mnemonics or keyboard accelerators.
- ❑ Include an "Index" entry that allows the user to access the help index.
- ❑ Include a "Keys & Shortcuts" entry to display all keyboard shortcuts for the application.
- ❑ Include a "Product Information" entry.

When providing a *Help* button in an application window . . .

- ❑ Provide a *Help* button for all windows that don't have a menu bar.
- ❑ For main windows, provide overview, task-oriented, keyboard shortcuts, and product information when the user clicks this button.
- ❑ For co-primary and support windows, provide overview and task-oriented information when the user clicks this button.
- ❑ For dialogs, provide help that focuses on the main purpose of the dialog and describes how to use the dialog.

- For primary and support windows that include a *Help* button, also provide access to context-sensitive help when the user presses <Shift>-<F1>. Dialogs typically don't support context-sensitive help mode.

Writing Online Help Content for SGIHelp

This section discusses the actual writing of the online help content. It might be of interest to both application designers and online help documentation writers.

Learning About SGIHelp

The basic unit of help information in SGIHelp is the *help card* (see Figure 4-2, Figure 4-3, and Figure 4-5). Help cards typically cover one topic although they may be divided into subtopics. They can have cross-references (links) to other help cards for the application. These links appear as blue text in the help card viewer window.

Before writing online help content for SGIHelp, read through the help information for some Indigo Magic applications to get a feel for content, functionality, and presentation. Some good examples of help in applications with a Help menu are the Directory View windows and the Icon Catalog application. Take a look at the User Manager (accessed from the System menu in the Toolchest) as an example of a utility that provides help through a Help button. Also try out the *Help* button in dialogs in the desktop applications.

As you experiment with SGIHelp, follow some (blue) hypertext links to other help topics. Trace these cross-reference links both forward and backward. Look at the presentation of inline figures. Print some of the help information. Bring up the help index and search this index for specific keywords and phrases. Be sure to try out the "Click for Help" facility to get a feel for how specific the information is when you click on an area of the window.

Creating Help Cards

Online help is designed to be presented in short chunks of information that answer specific questions. (See “Types of Online Help” earlier in this chapter for a list of the types of questions users ask and the types of information that should be available in your online help to answer these questions.) Each help topic consists of one card of information that answers a specific question. The SGIHelp viewer is designed to take up no more than one quarter of the screen. If a help card has more information than will fit in the viewer window at one time, the user can use the scroll bar to scroll through the information. Each help card should span no more than three viewer windows of information so that users can keep scrolling to a minimum. Sometimes a help card may depend on another help card. It’s better to provide a link to the supplementary card than to repeat the information.

All help cards have titles. Each title should tell the reader what’s contained in the specific help card, and it shouldn’t be written so that it assumes users have read other help topics. The titles should be as descriptive as possible since they also appear in an index of help topics. Users can display the list of help topics in the index by choosing “Index” from the Help menu. Note that the topics listed in the index should match the titles of the help cards as closely as possible.

As in any form of documentation, use familiar terminology and natural, friendly, real-world language. Remember that when users access online help, they’re often searching for a quick answer to a specific question. This isn’t the time to teach them new technical terms.

Writing Context-Sensitive Help

Before the writing process begins for context-sensitive help, list all appropriate window components that users might need help on. Describe components in terms of the user tasks they support. That is, don’t just say what the component is—tell users when they would find this component useful. Be careful not to mix specific step-by-step instructions with the context-sensitive information; instead, you might include links to the appropriate task-oriented information. Figure 4-3 shows the help card for context-sensitive help on the shelf area in a Directory View window.

Writing Overview Information

Overview information should be a one or two paragraph overview of either the application's functionality (main windows) or the functionality of a specific application window (co-primary and support windows). It should answer the question, "What does this application (or application window) do?" Figure 4-2 shows the overview card for Directory View windows. Notice how it provides an overview of the entire application and limits the information to the application's functionality.

Writing Task Information

Start by compiling a list of major tasks that users will want to perform using the application. For each task, supply the step-by-step instructions necessary to accomplish that task. If the instructions span more than three or four viewer windows of information, try to divide this topic into several smaller help topics. In addition to the step-by-step information, provide a brief summary paragraph at the beginning of the help card. Some readers take the time to read only the first few sentences of a help card.

Figure 4-6 is an example of a task-oriented help card taken from a Directory View window. Note that it displays the summary paragraph and the initial steps in the in the first viewer window of information. Remember that when displayed in a Help menu, the list of tasks gives users a quick overview of what the application can do, before they've even had to look at a help card. Make sure that you have descriptive titles for your task-oriented help cards and review them as a group. See Figure 4-2.

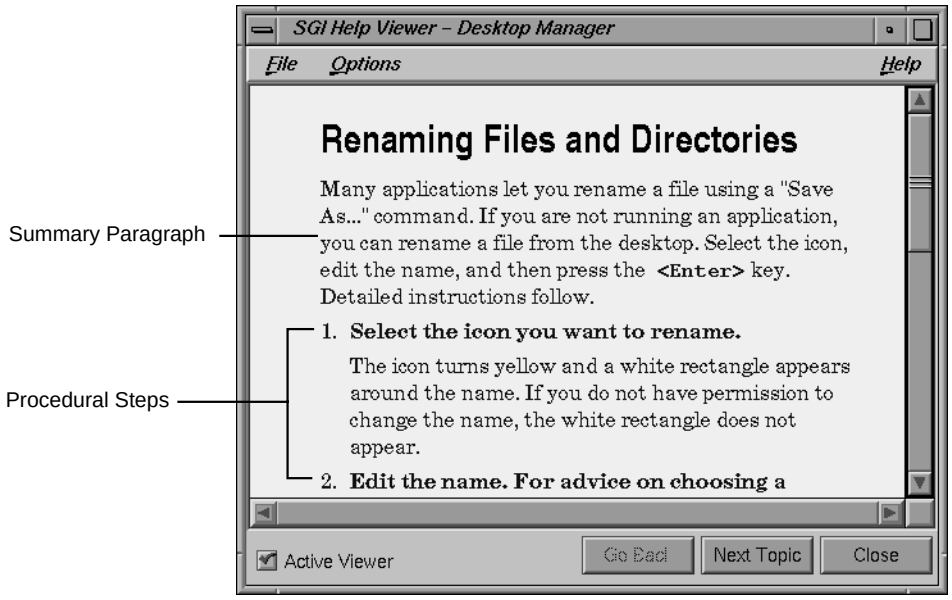


Figure 4-6 Task-Oriented Help Example

Writing Index Information

Users access the help index to browse or to search for specific topics, so it's important to have descriptive titles for your help cards. The help index presents topics roughly in the order in which they appear in the Help menu: overview, list of tasks, context-sensitive topics, and keyboard shortcuts. If the application has more than one window with a Help menu, the help topics are grouped by windows. See Figure 4-2.

Writing Keyboard Shortcut Information

Every Help menu should have a "Keys & Shortcuts" entry containing all the mnemonics, keyboard accelerators, and function keys for the entire application. Typically, this help card contains an introductory description and a three-column table containing columns for the menu entry label, keyboard accelerator, and mnemonic. You should also include information on function keys and other shortcuts that can be used in your application. See Figure 4-2.

Writing for Windows With Help Buttons

If your main application window has a *Help* button rather than a Help menu, the initial help card should contain an overview of your application, task-oriented information, a list of all keyboard shortcuts, and product information. To make it easier for users to navigate through this amount of information, after the overview information at the beginning of the help card, place links which take users directly to the other chunks of help information contained in the card. This strategy greatly reduces the amount of scrolling and searching necessary. See Figure 4-5 for an example help card with links to additional information.

On other windows with a *Help* button, present only the help information for the specific window. If this information is longer than three or four viewer windows of information, use the above strategy of presenting brief overview or summary information followed by a list of links to the individual chunks for the task-oriented information.

Guidelines for Creating SGIHelp Content

When writing any online help for your product . . .

- ❑ Create separate help “cards” for each help topic.
- ❑ Limit each help card to no more than three viewer windows full of information.
- ❑ Write a descriptive heading for each help card.
- ❑ If a particular help topic needs supplemental information, provide links to that information rather than repeating it in the current card.
- ❑ Use language your users will understand.
- ❑ Use figures when appropriate. SGIHelp allows users to view graphics inline with the help text.

When writing the “Click for Help” context-sensitive information for your application . . .

- ❑ Begin by listing the individual controls and areas of your application windows that you need to describe.

- ❑ At a minimum, provide separate help cards for each group of controls and areas in that window.
- ❑ Provide descriptions in terms of the user tasks the components support.
- ❑ Don't include procedural, task-oriented information with the context-sensitive information—include links to the appropriate task-oriented topics instead.

When writing the overview help cards for your application . . .

- ❑ Restrict the content to information about what the product does, not how to use it.
- ❑ Limit the text to one or two viewer windows of information.
- ❑ Use the heading "Overview" for the main window's overview help card and "Overview of <*window name*>" for co-primary and support windows with overview help cards.

When writing the task-oriented information for your application . . .

- ❑ Begin by listing the tasks that users will want to accomplish with your application.
- ❑ For each task, list the step-by-step instructions users will need to accomplish that task. If these instructions span more than three or four viewer windows, try to divide this topic into several smaller help topics.
- ❑ Provide a brief summary paragraph at the beginning of the help card, followed by the step-by-step information.

When writing the keyboard shortcuts information for your application . . .

- ❑ Include all shortcuts for your application in a single card—mnemonics, keyboard accelerators, and function keys.

When creating the index for your help topics . . .

- ❑ Match the titles in the index as closely as possible to the titles of the help cards.
- ❑ Place the topics in the index in the following order—overview, list of tasks, context-sensitive topics, and keyboard shortcuts.

When writing help information that will be available from a *Help* button rather than from a Help menu . . .

- For the main application window, the help card should contain an overview of your application, task-oriented information, a list of all keyboard shortcuts, and product information.
- For *Help* buttons not on the main application window of your application, present only the help information for the specific window.
- If the amount of information on this one help card spans more than three or four viewer windows of information, after the overview or summary information at the beginning of the help card, place links which take users directly to the other chunks of help information contained in that card.

After writing your online help . . .

- Have reviewers examine your help content online rather than reviewing a printed copy. Help topics will “read” differently depending on which paths readers (reviewers) traveled to get there.
- Have reviewers check the titles of the help topics to make sure they are descriptive and appropriate.
- Have reviewers test out all links to make sure they are appropriate.

Online Documentation

Silicon Graphics now ships almost all manuals in online form using IRIS InSight. IRIS InSight is an online documentation delivery system based on SGML (Standard Generalized Markup Language), a portable, platform-independent document description language.

InSight offers users a convenient means of viewing manuals. It can save money for organizations that ship a high volume of manuals, that have frequent releases, and whose users can get by without hardcopy versions of the manual. If your organization fits this description and uses a process based on a standardized approach to document management, consider using IRIS InSight. To contact Silicon Graphics for more information, send email to techpubs@sgi.com.

Desktop Variables

The Indigo Magic Desktop allows users to set a variety of variables that can affect the way an application behaves on the desktop. These variables are set using the graphical control panels selected from the Desktop->Customize cascading menu in the Toolchest. Users expect all applications to adhere to the values set in these graphical control panels. Your application should support this expectation by using the values that users have set for these variables.

Your application needs to be concerned only with those variables listed in this section. For details on having your application respond to the settings of these variables, see Chapter 3, “Using Schemes,” and Chapter 5, “Window, Session, and Desk Management,” both in the *Indigo Magic Desktop Integration Guide*. The other variables that users can set using the graphical control panels either don’t affect your application or automatically apply to your application. These variables include things such as whether the mouse is set for right-handed or left-handed users and the default permissions for new files.

Scheme Setting

The Scheme variable allows users to change the color and font scheme used by applications that are currently running on the desktop. This variable is discussed in the section “Schemes for Colors and Fonts” in Chapter 3.

Figure 4-7
Language Control Panel

Auto Window Placement Setting

The Auto Window Placement setting allows users to specify whether newly opened windows should be placed automatically by the *4Dwm* window manager or should be placed interactively by the user. This variable is discussed in the section “Window Placement” in Chapter 3.

Language Setting

When users launch an application, they expect the application to appear in the language set in the Language Control panel (see Figure 4-7). The default

setting is U.S. English. For guidelines on designing an internationalized application, see the online manual *Topics in IRIX Programming*.

Mouse Double-Click Speed Setting

Users expect applications to respond to the mouse double-click speed set in the Mouse Settings control panel (see Figure 4-8).

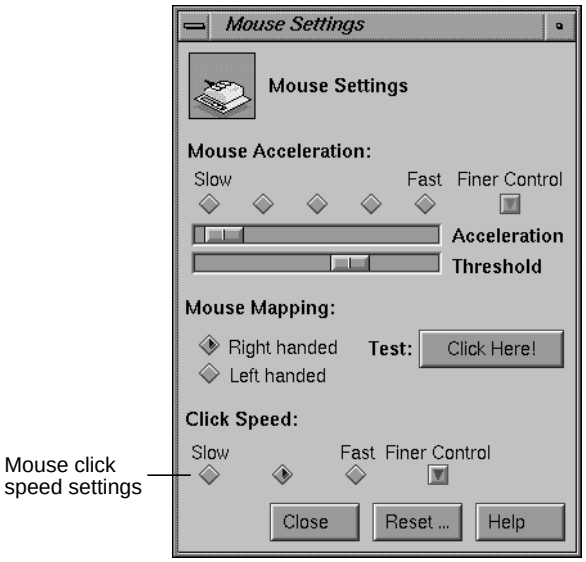


Figure 4-8 Mouse Settings Control Panel

Editor Preference Setting

When users open ASCII text files on the Indigo Magic Desktop, the default editor is *jot*, a point-and-click graphical editor. Users can change the default editor using the Desktop control panel (see Figure 4-9). Applications that let users modify ASCII text files should default to the text editor specified in the Desktop control panel.

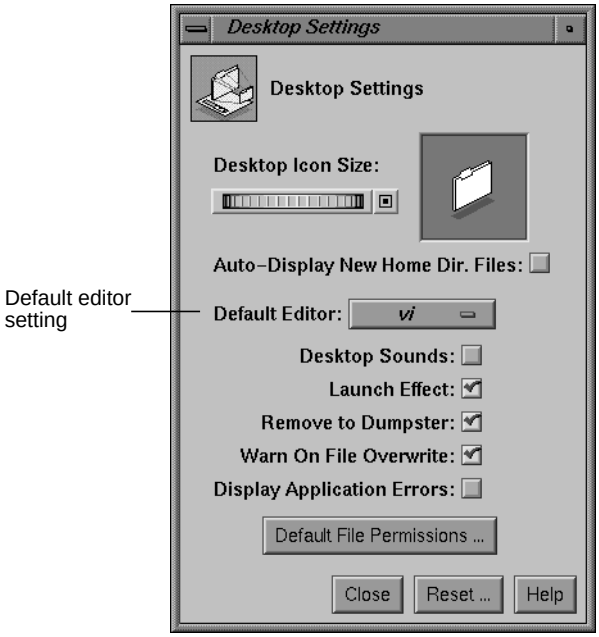


Figure 4-9 Desktop Settings Control Panel

For example, the Compose window in the MediaMail application, shown in Figure 4-10, gives users a simple editor for composing electronic mail messages. In addition, users can also elect to use their preferred editor by selecting the “Editor” entry in the Edit menu for this window. This launches the editor set in the Desktop control panel, which in this example is *jot*.

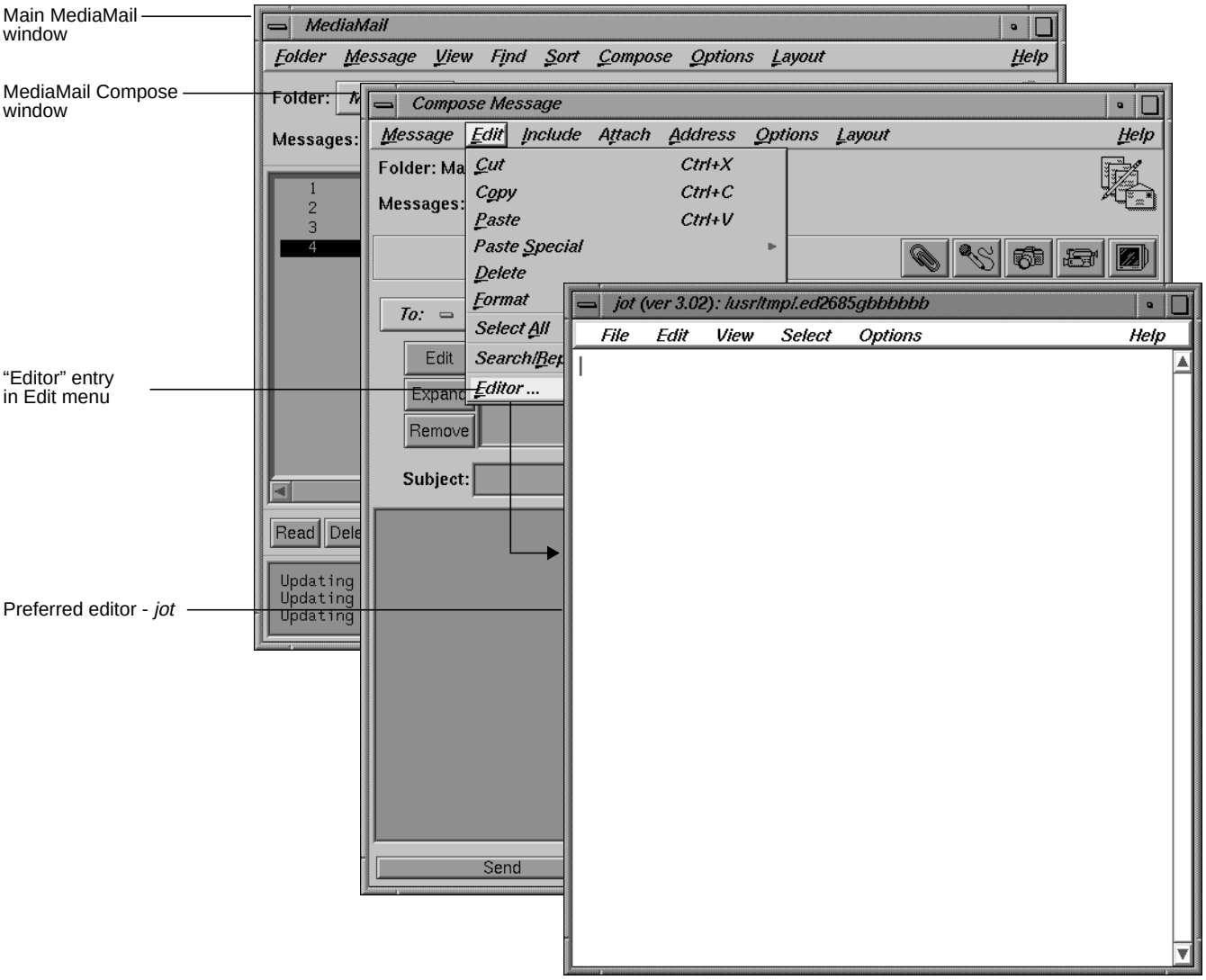


Figure 4-10 Selecting Preferred Editor in MediaMail

Desktop Variables Guidelines

In general . . .

- ❑ Always honor the user's desktop customization settings. Never override or ignore them.

When considering color and font schemes for your application. . .

- ❑ Use the pre-packaged color and font schemes supplied by Silicon Graphics rather than designing your own.

When considering window placement . . .

- ❑ Set a preferred window position for all primary windows. Don't set a required window position for primary windows.
- ❑ Try to anticipate other application and tool windows that may be displayed with your application and set your preferred default position appropriately.

To allow users to control the language for your application . . .

- ❑ Check the value of the default language each time your application is launched. Don't reset this value while the application is running.

To allow users to control the mouse double-click speed for your application . . .

- ❑ Check the value of the double-click speed each time your application is launched. Don't reset this value while the application is running.

If users will be editing and/or browsing ASCII text files in your application . . .

- ❑ Make their preferred editor (specified in the Desktop control panel) available for use on text files.
- ❑ Check the value for the preferred editor each time your application is launched, but don't reset this value while your application is running.
- ❑ If users can only browse the ASCII text files, launch the editor in read-only mode.

File Alteration Monitor (FAM)

The File Alteration Monitor (FAM) is a service that monitors changes to files and directories in the file system and notifies interested applications of these changes. If your application needs to stay in sync with the state of any part of the file system, you should use FAM. This process is considerably more efficient than having an application poll the file system itself to detect changes. (See Chapter 8, "Monitoring Changes to Files and Directories," in the *Indigo Magic Desktop Integration Guide* for details on using FAM in your application.)

Here are two examples of Indigo Magic applications that use FAM:

- The File Manager relies on FAM to track any changes to directories and/or files that are visible on the user's desktop or in any open Directory View windows. When FAM notifies the File Manager of any changes, the File Manager updates the views. This guarantees that the user always sees an accurate view of the file system, both on the desktop and in the Directory View windows.
- MediaMail uses FAM to monitor the arrival of new mail by tracking changes to the */usr/mail* directory. If a user is running MediaMail and new mail arrives, the user is immediately notified.

File Monitoring Guideline

❑ If your application needs to stay in sync with the state of any part of the file system, use FAM. Don't have your application directly poll the file system to detect changes.

Chapter 5

Data Exchange on the Indigo Magic Desktop

The Indigo Magic Desktop enables users to transfer data between applications. There are two types of transfers from the user's point of view. *Copy* takes data specified in the source application and creates a duplicate in the destination application. *Move* removes the data specified in the source application and places it in the destination application. The source and destination applications can be the same application.

This chapter covers the following topics:

- "Supporting the Clipboard Transfer Model" explains what users expect when performing operations using the "Cut," "Copy," and "Paste" entries in the Edit menu.
- "Supporting the Primary Transfer Model" describes the expected behavior when users select data and copy it using the left and middle mouse buttons.
- "Data Types Supported for Inter-Application Transfer" lists those data types that users can transfer between applications.

Supporting the Clipboard Transfer Model

In the clipboard transfer model, users move or copy a selection using the "Cut," "Copy," and "Paste" entries from the Edit menu. If your application contains data that users will want to transfer to other applications or across separate instantiations of your application, your application should support the clipboard transfer model described in this section. Note that this model is a subset of the clipboard transfer model described in Section 4.3 of the *OSF/Motif Style Guide*. For information on implementing clipboard transfer, see "Implementing the Clipboard Transfer Model" in Chapter 7 of the *Indigo Magic Desktop Integration Guide*. For more information on the layout of the Edit menu, the behaviors of each entry, and keyboard accelerators, see "Edit Menu" in Chapter 8.

In a typical clipboard transfer, the user selects data in an application window and initiates a move or copy transfer operation by choosing "Cut" or "Copy" from the window's Edit menu. The source application then takes ownership of the clipboard atom replacing the prior owner (if there was one). The user completes the transfer by choosing "Paste" from the Edit menu in the destination application. The destination application then moves or copies the data associated with the clipboard to this destination. When the transfer is complete, the newly pasted data is not selected or highlighted. For more information on selection techniques, see "Selection" in Chapter 7.

Note that clipboard data is nonpersistent. If a user quits the application that currently owns the clipboard, the data associated with the clipboard atom is lost. Only one application can own the clipboard atom at any given time.

Figure 5-1 illustrates clipboard transfer using the SoundEditor application. In the figure, there are two instantiations of the application; the window on the left is the source and the one on the right is the destination. Note that the clipboard atom in the figure is only a representation and doesn't actually appear. In this example, the user selects a region of sound in the source window and chooses "Copy" from the Edit menu in that window. The source instantiation of SoundEditor takes ownership of the clipboard atom. When the user chooses "Paste" from the Edit menu in the destination window, the data associated with the clipboard is inserted into the destination sound file at the insertion cursor (the vertical black line). Note that after the "Paste" operation, the SoundEditor application doesn't select or highlight the newly pasted data.

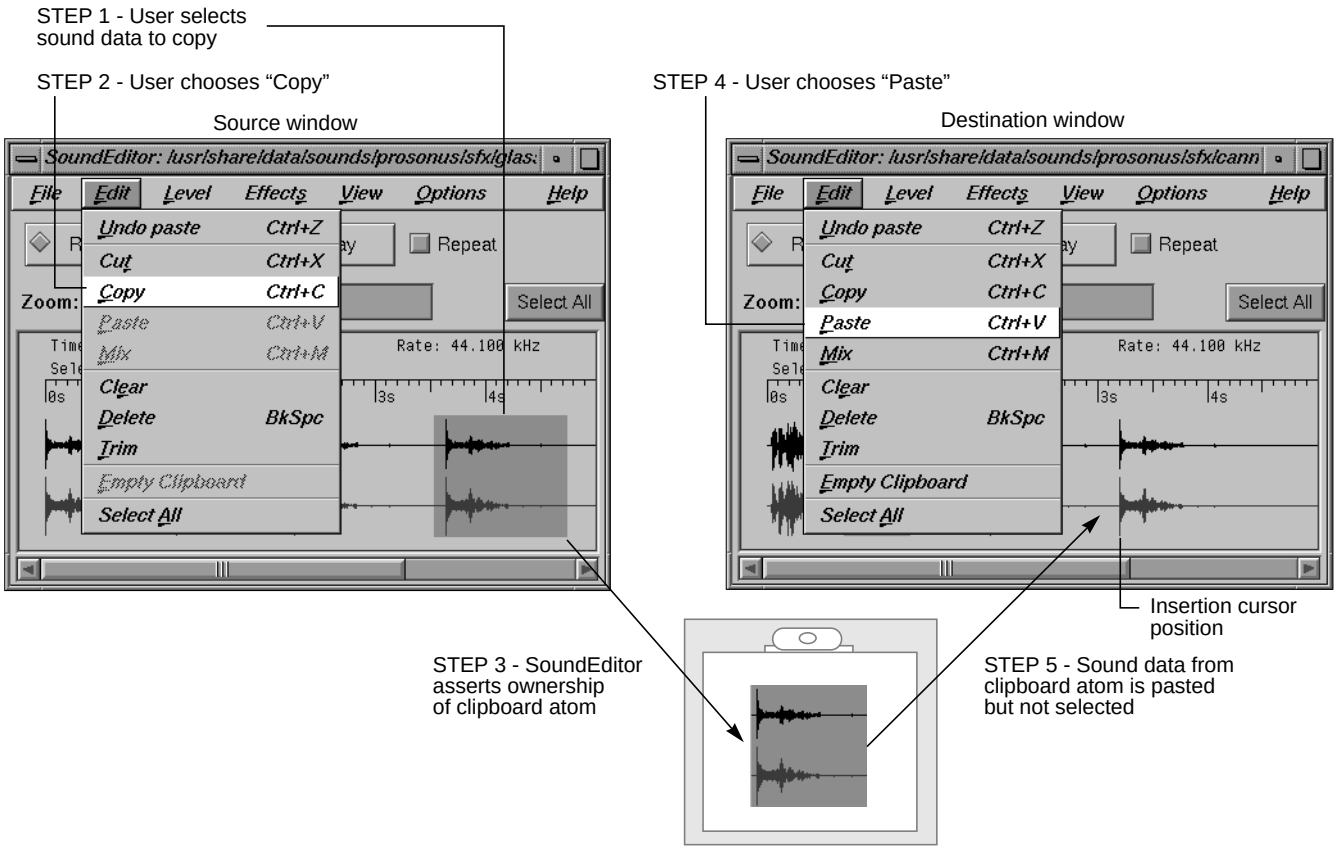


Figure 5-1 Clipboard Transfer Example

The clipboard operates independently of the primary selection described in the next section, “Supporting the Primary Transfer Model.” When a user chooses “Cut” or “Copy” from an Edit menu, that application takes ownership of the clipboard but the primary selection remains unchanged.

Supporting the Primary Transfer Model

In the primary transfer model, users select the data for transfer using the left mouse button and copy the data to the destination application using the middle mouse button. If your application contains data that users will want to transfer to other applications or across separate instantiations of your application, your application should support the primary transfer model described in this section. Note that this model is a subset of the primary transfer model described in Section 4.3 of the *OSF/Motif Style Guide*. For information on implementing primary transfer, see "Implementing the Primary Transfer Model" in Chapter 7 of the *Indigo Magic Desktop Integration Guide*.

In the *primary transfer model*, when a user begins a selection in an application, that application takes ownership of the primary selection atom, replacing the previous owner if there was one. This selection is referred to as the *primary selection*. The user can then copy the primary selection to a destination application by moving the pointer to the destination window (making it the active window) and clicking the middle mouse button. At this point, the destination application copies the primary selection data to this destination. Note that the data is pasted at the position of the pointer, not at the insertion cursor. Also note that when the copy is complete, the newly pasted data isn't selected or highlighted. For more information on selection techniques, see "Selection" in Chapter 7.

Figure 5-2 through Figure 5-3 illustrate primary selection and transfer using the SoundEditor application. Figure 5-2 illustrates making a primary selection. The user creates a selection by dragging with the left mouse button across a range of sound data. As soon as the user begins this selection, SoundEditor takes ownership of the primary selection atom. This selection is then referred to as the primary selection.

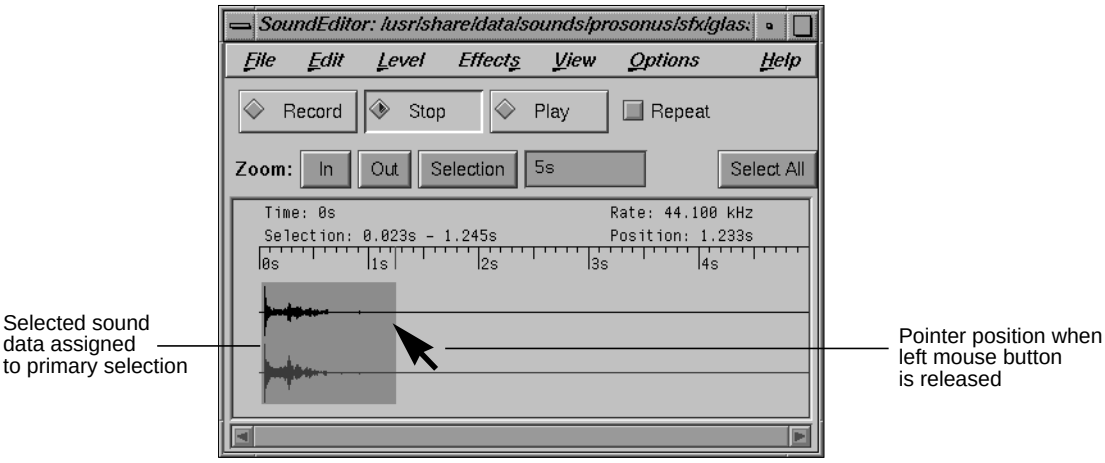


Figure 5-2 Primary Selection Example

Figure 5-3 shows the source and destination windows just prior to a primary transfer. Figure 5-4 shows the source and destination windows after the transfer. Note that when the user clicks the middle mouse button, the primary selection is inserted at the pointer location rather than at the insertion cursor. Also note that after the transfer operation, the SoundEditor application doesn't select or highlight the newly pasted data and the primary selection doesn't change.

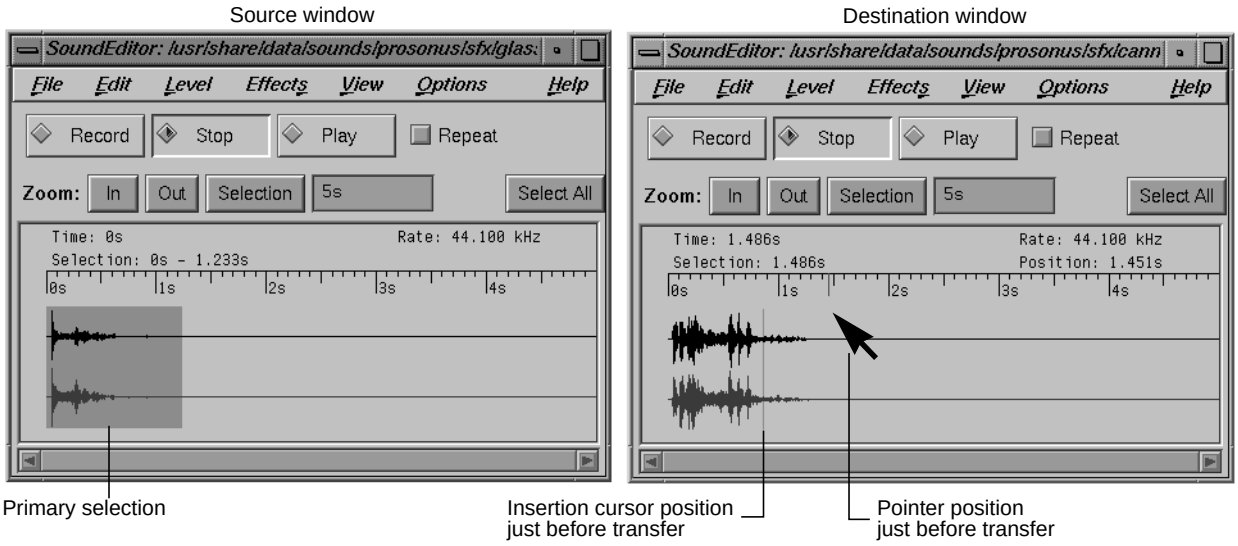


Figure 5-3 Primary Transfer Example: Before Transfer

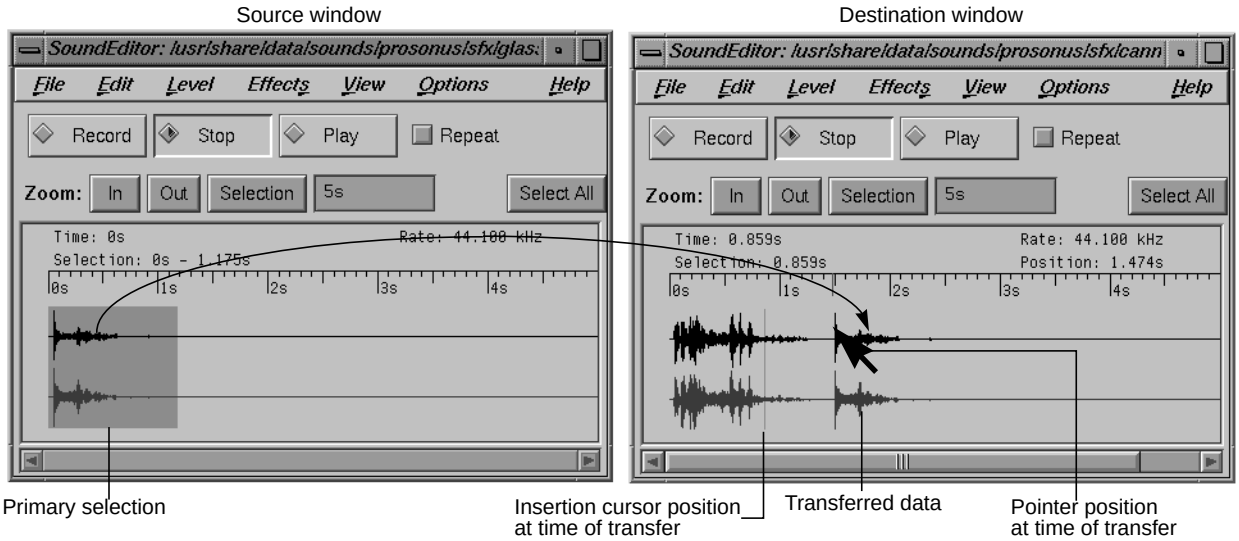


Figure 5-4 Primary Transfer Example: After Transfer

In general, when your application loses the primary selection, it should keep its current selection highlighted. When a user has selections highlighted in more than one window at a time, the most recent selection is always the primary selection. This is consistent with the *persistent always selection* model discussed in Section 4.2 in the *OSF/Motif Style Guide*. Exceptions to this guideline are those applications that use selection only for primary transfer—for example, the *winterm* Unix shell window. The only reason for users to select text in a shell window is to transfer that text using the primary transfer mechanism. In this case, when the *winterm* window loses the primary selection, the highlighting is removed. This is referred to as *nonpersistent selection* in Section 4.2 in the *OSF/Motif Style Guide*. For guidelines on selection in general, see "Selection" in Chapter 7.

If the user returns the keyboard focus to a window with a highlighted, superseded primary selection, the application should allow the user to reinstate the highlighted selection as the primary selection by pressing <Alt-Insert>. In addition to supporting this key combination, you can also add the entry "Promote" to the Edit menu for the application window. The "Promote" menu entry should be active only when your application has a selection which is not the primary selection. (For details of placing this item in the Edit menu, see "Edit Menu" in Chapter 8.)

Note that when a user begins to modify a selection, such as adding elements to it, it's considered to be a new selection. In this case, your application needs to reassert ownership of the primary selection if your application doesn't currently own it.

The primary selection operates independently of the clipboard, as described in the previous section, "Supporting the Clipboard Transfer Model." When the user makes a selection, that selection becomes the primary selection—there's no effect on the contents of the clipboard.

Data Types Supported for Inter-Application Transfer

Applications can use the atom names listed in Table 5-1 for both clipboard and primary transfer of the corresponding types of data. If you want users to be able to transfer data from your application to other applications, you need to be able to export data in one of the formats listed in Table 5-1. If your application is to receive data from other applications, see Table 5-1 for what's required. For complete details of implementing both clipboard and primary data transfer, see Chapter 7, "Interapplication Data Exchange," in the *Indigo Magic Desktop Integration Guide*.

Table 5-1 Data Types Supported for Inter-Application Transfer

Target Format Atom Name	Nature of Data	What Target Receives	Receiving Application Requirements
INVENTOR	3D, generated by Inventor	Inventor data (Scene Graph)	Ability to read Inventor data
_SGI_RGB_IMAGE_FILENAME	color image in *rgb* format	rgb file name	Ability to use *rgb* files
_SGI_RGB_IMAGE	color image in *rgb* format	stream of rgb data	Ability to use *rgb* files
_SGI_AUDIO_FILENAME	sound data	audio file name	SGI audio file library *libaudiofile*
_SGI_AUDIO	sound data	stream of audio data	SGI audio file library *libaudiofile*
_SGI_MOVIE_FILENAME	movie data	movie file name	SGI Movie library *libmovie*
_SGI_MOVIE	movie data	stream of movie data	SGI Movie library *libmovie*
STRING	text encoded as ISO Latin 1	textual data	Ability to read text encoded as ISO Latin 1
COMPOUND_TEXT	compound text	textual data formatted as compound text	Ability to read compound text

Data Exchange Guidelines

If your application contains data that users may wish to transfer to other applications or across separate instantiations of your application . . .

- Support the Clipboard Transfer Model using the “Cut,” “Copy,” and “Paste” entries in the Edit menu. In this model, the clipboard is a global entity that’s shared by all applications. Your application shouldn’t use these entries to refer to a clipboard that’s private to your application.
- When supporting the Clipboard Transfer Model, don’t select or highlight newly pasted data after a “Paste” operation.
- Support the Primary Transfer Model. Assert ownership of the primary selection when the user begins to make a selection. Insert data at the location of the pointer when the user clicks the middle mouse button (which isn’t necessarily at the insertion cursor).
- When supporting the Primary Transfer Model, don’t select or highlight newly transferred data after a transfer operation.
- Use *persistent always selection* highlighting (keep the current selection highlighted even when your application loses the primary selection), unless the only action that can be performed on the selection is to copy the data using primary data transfer. In this case, use *nonpersistent selection* highlighting—that is, remove the selection highlight when the selection is no longer the primary selection.
- When supporting the Primary Transfer Model, if the current active window has a selection that isn’t the primary selection, reinstate this selection as the primary selection if the user presses <Alt-Insert>. Additionally, you can include a “Promote” entry in the Edit menu to perform the same function.
- When supporting the Primary Transfer Model, when the user begins to modify a selection, such as adding elements to it, reassert ownership of the primary selection if your application does not currently own it.
- When supporting both Clipboard Transfer and Primary Transfer, keep the primary selection independent from the clipboard. When the user begins to make a selection in your application, assert ownership of the primary selection but do not change the ownership of the clipboard. When the user chooses “Cut” or “Copy” from an Edit menu in your application, assert ownership of the clipboard but do not change the ownership of the primary selection.

PART TWO

Interface Components

Chapter 6

Application Windows

This chapter discusses the role of different types of windows in the Indigo Magic environment. It includes information on how your application should combine the different types of windows, as well as guidelines on what elements are appropriate for primary and support windows, and how these elements should be arranged. (Dialog windows are discussed in detail in Chapter 10, “Dialogs.”)

This chapter covers the following topics:

- “Application Models” discusses different models for applications. Which model is appropriate for your application depends upon whether your application needs multiple primary windows, and whether it can have multiple documents open at the same time.
- “Main and Co-Primary Windows” discusses the design of primary windows.
- “Support Windows” discusses the design of support windows.
- “Pointer Behavior in a Window” discusses the limits of what your application should do with the mouse pointer.

Note that by default, *4Dwm*, the window manager for the Indigo Magic Desktop, provides window decorations and a Window menu for all application windows. The specific window decorations and contents of the Window menu depend on the type of window and whether any of the components in the window are resizable. For guidelines on window decorations, Window menu entries, and related issues, see “Application Window Categories and Characteristics” in Chapter 3.

Application Models

This section describes how the different types of windows used in the Indigo Magic environment can be used together in applications.

Window Types

As discussed in "Application Window Categories" in Chapter 3, windows in the Indigo Magic environment are divided into four types, which are subdivisions of Motif's two types: primary and secondary. Primary windows are divided into main primary windows and co-primary windows, and secondary windows are divided into support windows and dialogs. The definitions are repeated here for your convenience:

- A *main primary window* serves as the application's main controlling window. It's used to view or manipulate data, get access to other windows within the application, and kill the process when users quit. There's only one main primary window per application (and sometimes it isn't visible to users).
- A *co-primary window* is used for major data manipulation or viewing of data outside of the main window
- A *support window* is a *persistent* special-purpose window. It typically contains a control panel or tool palette that operates directly on data in a primary window. A support window can be used repeatedly.
- A *dialog* is a *transient* window, typically used for short, quick user input, such as an action confirmation, or system output, as in a warning message. A dialog may be user-requested or application-generated. It is usually dismissed as soon as it has served its purpose. Dialogs are discussed in detail in Chapter 10, "Dialogs."

The next section ("Standard Application Models") describes several standard models for combining these types of windows in a real application. Whichever model you choose, try to keep the window hierarchy shallow so that users can form a relatively simple conceptual model of how your application's windows are related. Figure 6-1 shows the allowable parent and child relationships within an application's window hierarchy. For example, all co-primary windows should be children of the main window, so primary windows should never be more than two levels deep; dialogs can be children of any type of window.

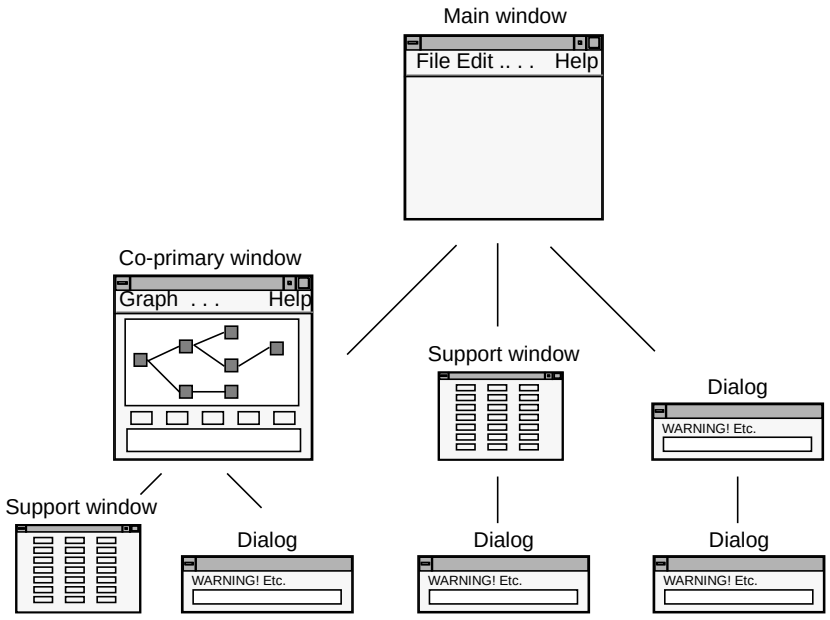


Figure 6-1 Allowable Parent-Child Window Relationships

Standard Application Models

Applications can combine the basic window types in many ways, but most applications fall into one of four basic models. These models differ with respect to whether they can have one or multiple documents (files) open at a time and their use of primary windows. (For information about how to implement the various models, see “Implementing an Application Model” in Chapter 5 in the *Indigo Magic Desktop Integration Guide*.)

Note: The term *document* means a grouping of data and shouldn’t be thought of as referring exclusively to text-oriented files. A document can include such data types as film clips, audio segments, and 3D scenes.

“Single Document, One Primary” Application Model

“Single Document, One Primary” (see Figure 6-2) is the most basic model. It accomplishes all of its tasks within the main window and uses as many support windows and dialogs as needed. Users can work on only one document at a time. Thus, when a user has one document open and opens a second document, the second document replaces the first. IRIS Showcase operates in this manner.

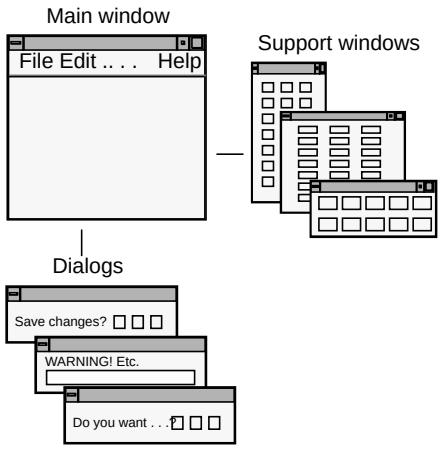


Figure 6-2 ”Single Document, One Primary” Application Model

"Single Document, Multiple Primaries" Application Model

The "Single Document, Multiple Primaries" model (see Figure 6-3) uses both main and co-primary windows to accomplish major tasks. In this model, each co-primary window performs a different function; these functions supplement the functionality of the main window. MediaMail is an example of this model. Its main window lets users select electronic mail messages from a folder and perform actions on them such as viewing, printing, deleting, and sorting. Its Compose and Message windows are typical of co-primary windows with different functions designed to support the functionality of the main window.

Also in this model, each primary window has its own menu bar tailored specifically to the functions in that window. Each co-primary window is opened from the main window. Each support and dialog window is associated with a specific primary window.

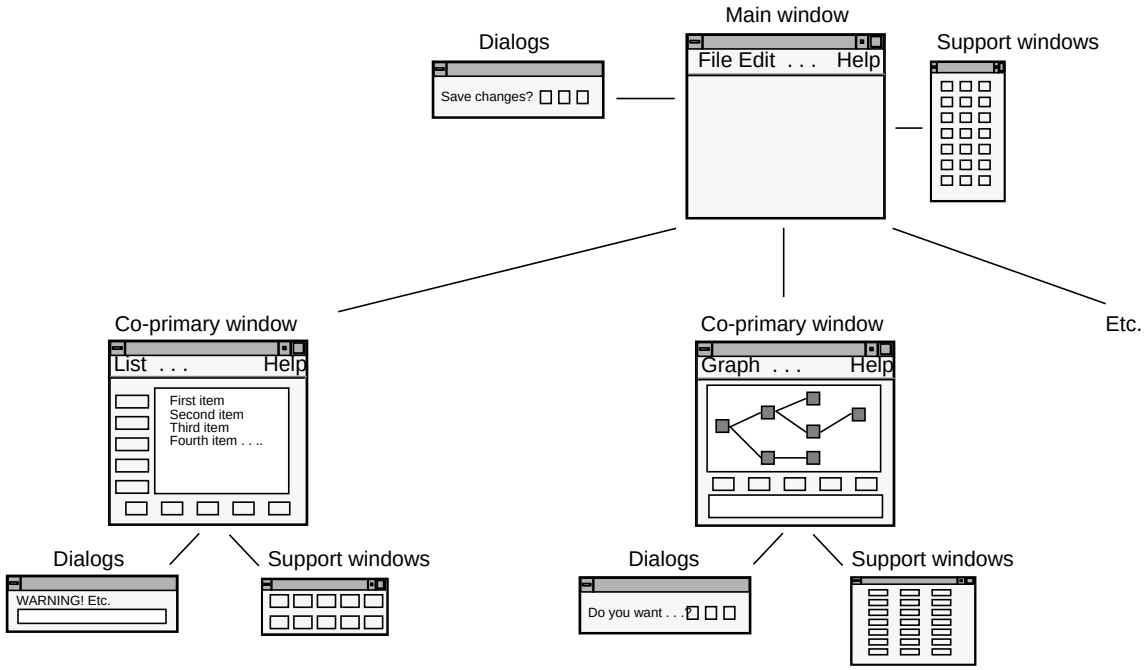


Figure 6-3 "Single Document, Multiple Primaries" Application Model

The single-document application with multiple primary windows is useful for dealing with data that the user needs to view in a number of different ways at the same time. For example, the WorkShop debugger operates on a single executable at a time. However, at a given time WorkShop might have several primary windows displaying source code, with other primary windows displaying call graphs and output from running the executable. In the case of WorkShop, the nature of the application demands multiple-primary windows.

"Multiple Document, Visible Main" Application Model

The "Multiple Document, Visible Main" model (see Figure 6-4) has a main window that's used primarily to launch co-primary windows. These co-primary windows are identical to each other and perform the same functions on different files or documents. All co-primary windows have identical entries in the menu bar and identical sets of dialogs and support windows. Each set of dialogs and support windows acts on a single document, the one represented by the co-primary window from which the dialog or support window was invoked.

Co-primary windows are opened either from the main window or from a co-primary window that's already open. IRIS InSight is an example of this model. Its main window lets users launch co-primary viewing windows, browse available files, and conduct global searches through these files. The co-primary windows are used for viewing online books.

A multi-document application should have a visible main window only if the main window offers functionality that can't be provided from the co-primary windows. For example, the IRIS InSight main window allows users to browse a list of books and to search a collection of books for specific words or phrases. A multi-document application should *not* have a visible main window if the only thing in the main window would be a menu bar. In this case, the "multiple documents, no visible main" model should be used, and the menu entries from the main window should instead be made available from the pull-down menus in each of the co-primary windows, as described in the next section.

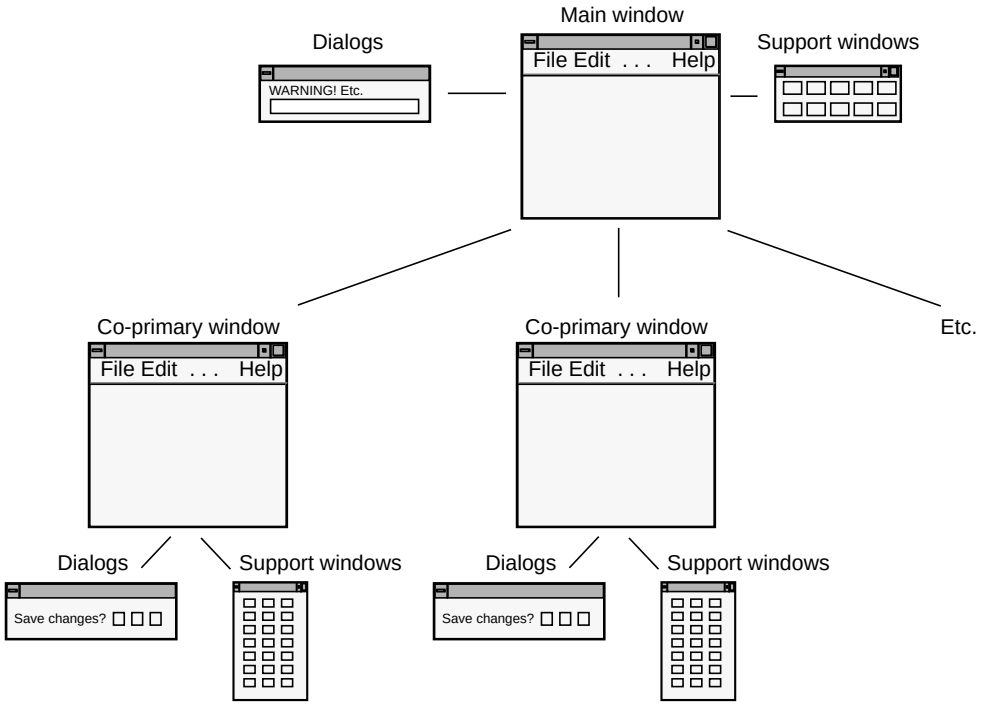


Figure 6-4 ”Multiple Document, Visible Main” Application Model

"Multiple Document, No Visible Main" Application Model

The "Multiple Document, No Visible Main" model (see Figure 6-5) is identical to the "Multiple Document, Visible Main" model described in the previous section except that the main window is invisible to the user (that is, unmapped) and new co-primary windows are launched from co-primary windows that are already open. Users open one document and leave it open while opening others. When the user has closed all of the documents, the process is killed.

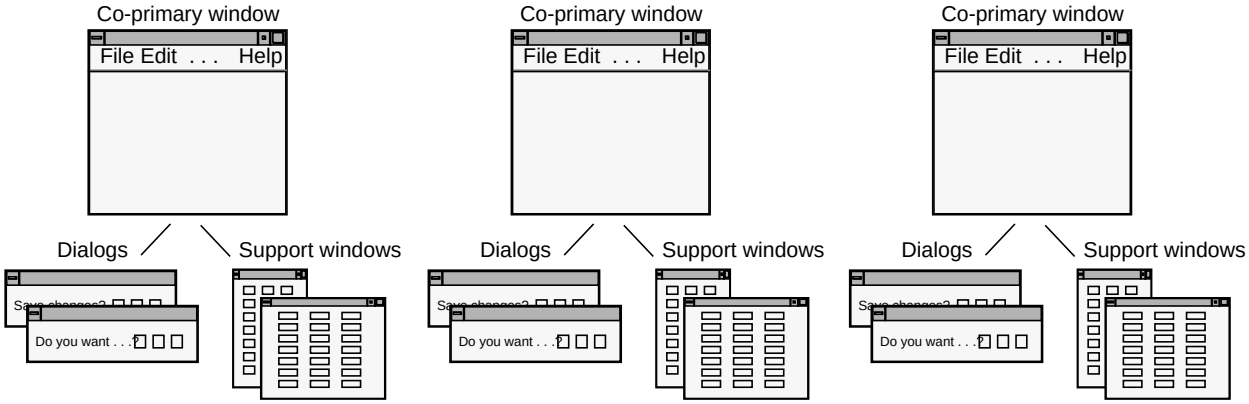


Figure 6-5 "Multiple Document, No Visible Main" Application Model

Application Model Guidelines

For all applications . . .

- ❑ Choose an appropriate application model for combining the different types of windows in your application.
- ❑ Use only the allowable parent-child window relationships and keep your application window hierarchy shallow.

Main and Co-Primary Windows

Every application has at least one primary window, which serves as the application's main controlling window. In fact, the majority of a user's interactions should occur in the main and co-primary windows (these window types are defined in "Window Types" earlier in this chapter). Several typical layouts for primary windows are shown in Figures 6-6 through 6-8. These layouts are discussed in the following sections.

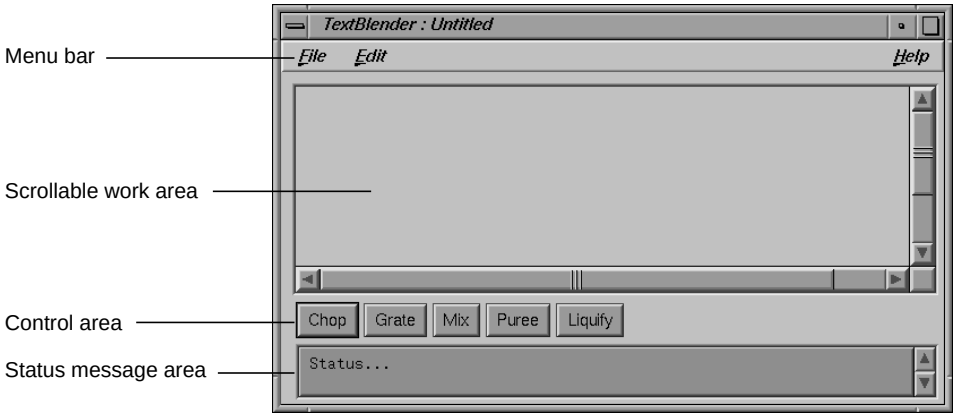


Figure 6-6 Basic Primary Window

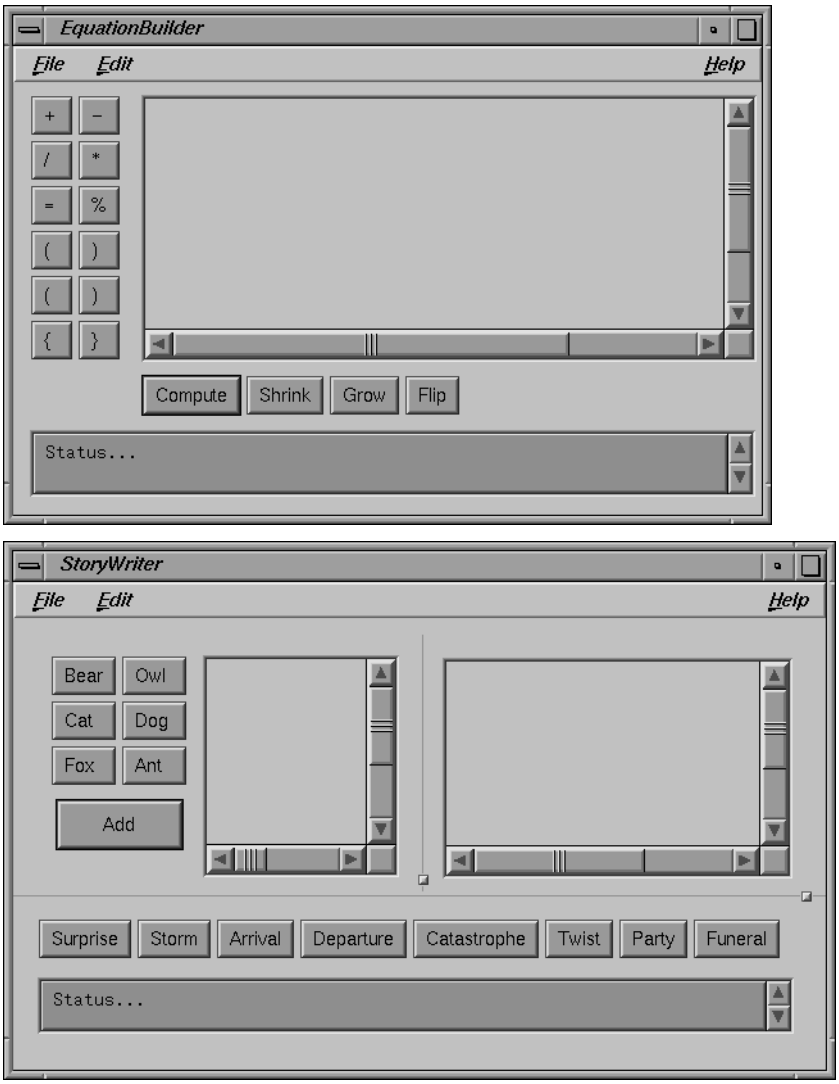


Figure 6-7 Primary Windows With Tool Palettes

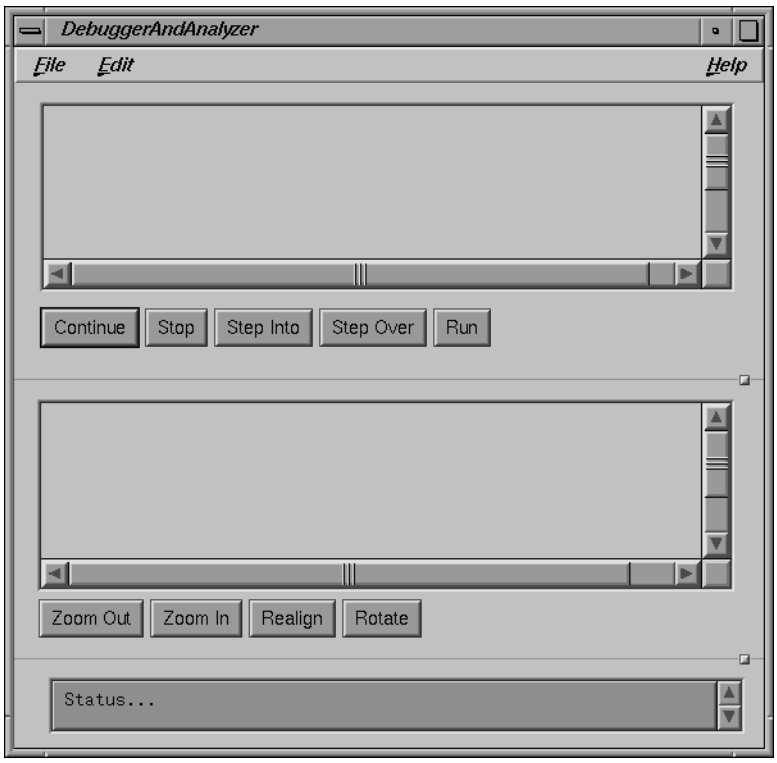


Figure 6-8 Primary Window With Two Panes

Menu Bars in Primary Windows

Primary windows typically have a menu bar, which is part of the window as shown in Figures 6-6 through 6-8. It shouldn't be a detached menu bar contained in a separate window. For details on designing the menu bar and its contents for a primary window, see "The Menu Bar and Pull-Down Menus" in Chapter 8.

If the primary window doesn't have a menu bar and all of its functionality is available using buttons, the window should still respond to the keyboard accelerators for Close (Ctrl+W) and Exit (Ctrl+Q) when appropriate. That is, the window should respond to these accelerators according to the guidelines

for when to use just Exit, when to use just Close, and when to use both Close and Exit for a window, as described in the section “File Menu” in Chapter 8.

Scrollable Work Areas in Primary Windows

The most prominent area of the window is typically a scrollable work area, as shown in Figures 6-6 through 6-8. Use scrollbars for the work area of a window when the window can be resized such that some of the available data may be hidden in the work area. Note that the scrollbars scroll only the work area and don’t scroll the menu bar, buttons in a command area, or the status message area.

Each scrollbar should span the entire width or height of the scrollable region. Don’t put controls or status information in the areas reserved for the scroll bars. Controls should be put in the control area, as described later in “Control Areas in Primary Windows.” Status information should be put in the status area, as described later in “Status Areas in Primary Windows.”

Use a vertical scrollbar on the right of the work area if the window or pane that contains the work area can be resized such that the data being displayed in the work area won’t fit in a vertical direction. Similarly, use a horizontal scrollbar directly below the work area if the window or pane can be resized such that the data being displayed in the work area won’t fit in a horizontal direction. Disable (rather than remove) the appropriate scrollbar when all of the data is being displayed in a given direction. For more information on using scrollbars, see “Scrollbars” in Chapter 9.

Control Areas in Primary Windows

Controls in primary windows, which are typically pushbuttons, are generally placed directly beneath the horizontal scrollbar and on the left side of the window (see Figures 6-6 through 6-8). (Note that this is different than the *OSF/Motif Style Guide*, which states that controls can be arranged along the top, bottom, or side of the work area.) As stated in the previous section, controls shouldn’t be placed directly below or on the right side of the work area in the scroll bar area—scrollbars should span the entire width and height of the work area. (See Chapter 9, “Controls,” for more information about how to use controls in your application.)

Control areas sometimes contain pushbuttons that are grouped into tool palettes. Figures 6-6 through 6-8 show primary windows with pushbuttons in the control areas, and Figure 6-7 shows primary windows with both tool palettes and pushbuttons that aren't part of a palette. Buttons that are part of a tool palette don't need to have corresponding menu entries. These "tools" typically allow a user to launch support and co-primary windows, or put the work area in a different mode (for example, edit mode or draw mode).

In contrast, pushbuttons used in control areas that do *not* represent tool palettes should represent the most frequently accessed application-specific menu entries which provide users a more convenient way of accessing these actions. There are two advantages to having these buttons repeat functionality from the menus:

- Having the functions in a menu allows you to assign keyboard accelerators to those common functions and allows users to choose between using point-and-click on the button or using a keyboard accelerator to access the functionality.
- Having the functions in a menu means that users can skim one place (the menu entries) to get an idea of the overall functionality of the product, and can skim another place (the control area) to see the frequently used functionality.

These non-palette buttons generally don't include actions from the standard File, Edit, or Help menus because these entries typically aren't the most frequently accessed when compared to the functionality that's specific to your application. For example, these buttons don't include the actions "Exit" or "Close" because these functions are used only once each time the window is opened, and they don't include "Help" because help is easily accessible from the Help menu. (For more information on buttons, see "Pushbuttons" in Chapter 9; also see "Standard Menus" in Chapter 8.)

The control area can also include an area to enter command line input. This command line area should be in addition to the buttons. Note that this differs from the *OSF/Motif Style Guide*, which states that the command area can contain only command line input. For an example window with an command line input area, see Section 6.2.1 in the *OSF/Motif Style Guide*.

Status Areas in Primary Windows

Primary windows can also include a single status message area at the bottom of the window if the application needs to post frequent messages to the user about the status of the application or the status of specific user actions (see Figures 6-6 through 6-8). For example, messages in this area might confirm that a file has been saved or that an option has been turned on or off. You should provide vertical scrollbars for this area so that users can view previously displayed messages.

This area shouldn't be used for warnings, errors, or other kinds of messages requiring the user to respond. Instead, dialogs should be used to display these types of messages. (See Chapter 10, "Dialogs," for guidelines on designing dialogs.) It also shouldn't be used to display help information.

Splitting Primary Windows Into Panes

Windows can be split into various panes of information (see Figure 6-8). Panes are separated from each other by separator lines. Each separator line may or may not include a sash control, which allows users to resize the panes. (See the Sash reference page in Chapter 9 of the *OSF/Motif Style Guide*.) Windows can include panes that are stacked vertically (Figure 6-8) or that are next to each other in a side-by-side horizontal layout (Figure 6-7). Note that control areas can be associated either with a specific pane or with the entire window.

Don't overuse panes—each application window typically should have no more than four separate panes and no more than three sash controls. If certain panes are optional to performing the task, provide menu entries that show or hide specific panes of information (see "View Menu" in Chapter 8).

Popup Menus in Primary Windows

Popup menus (which aren't shown in the figures in this chapter) can provide quick access to frequently used functions in primary windows. For information on when and how to use popup menus, see "Popup Menus" in Chapter 8.

Primary Window Guidelines

When designing a primary window . . .

- ❑ Use a menu bar unless all of the window's functionality is available through pushbuttons. Don't use a "floating" menu bar in a separate window.
- ❑ Support keyboard accelerators for Close (Ctrl-W) and Exit (Ctrl-Q) as appropriate, even if the window doesn't have a menu bar.

When designing a scrollable work area in a primary window . . .

- ❑ Use a vertical scrollbar on the right side of the work area when the data being displayed in the work area may not fit in a vertical direction. Use a horizontal scrollbar directly below the work area when the data may not fit in a horizontal direction. Don't use scrollbars if you're certain the data will fit.
- ❑ Disable the appropriate scrollbar when all the data is visible in a given direction. Don't remove the scrollbar.
- ❑ Make each scrollbar span the entire height or width of the work area. Don't include controls or status information in the scrollbar region.

When designing control areas in a primary window . . .

- ❑ Place controls below horizontal scrollbars or to the left of work areas.
- ❑ Provide pushbuttons for the most frequently accessed application-specific functions from the pull-down menus. Don't use pushbuttons for standard menu entries such as Open, Save, Close, Exit, Cut, Copy, Paste, and Help.
- ❑ Use pushbuttons only for functions that appear in menus, unless the pushbuttons are part of a tool palette.
- ❑ Provide an area for command-line input, if appropriate, in addition to (not in place of) pushbuttons.

To display status information . . .

- ❑ Use a status area along the bottom of a primary window if your application needs to post frequent messages about its status. Provide vertical scrollbars for this area so that users can view previously displayed messages.

- Use a status area to display messages that the user doesn't have to respond to rather than posting this noncritical information in dialogs. However, don't put critical warning or error messages in the status area (use a dialog instead).
- Don't use the status area to display help information.

When dividing a primary window into panes . . .

- Divide panes using separator lines. If users might need to resize the pane, also include a sash control.
- Try to limit the number of panes in a single window to four with no more than three sash controls.
- If certain panes are optional, allow users to hide or show these individual panes using entries in the "View" menu.

Support Windows

As defined in "Window Types" earlier in this chapter, support windows are persistent secondary windows that allow users convenient, constant access to sets of important controls that directly manipulate data in the associated primary window. The next two sections discuss the general design of support windows and a specific standard support window, the Indigo Magic color chooser.

General Support Window Design

Each support window should be associated with a specific primary window (its parent), which should be visible and mapped to the screen. (Support windows with invisible, unmapped parents don't work properly with desks, as described in "Desks" in Chapter 3.) Support windows shouldn't have other support windows or dialogs as parent windows. Note that this differs from the *OSF/Motif Style Guide*, which states that secondary windows can have other secondary windows as parents.

Support windows typically don't have menu bars like primary windows, but they should still respond to the keyboard accelerator for closing a window ("Ctrl+W"). Support windows should be launched from items in

the Tools menu of the associated primary window's menu bar (see "Tools menu" in Chapter 8) or from a tool palette in the primary window (see "Control Areas in Primary Windows"). Users can show or hide support windows as they wish, and rearrange where they're displayed with respect to the primary window. This makes support windows more versatile than control areas in a primary window. When bringing up support windows, don't overlap the work area of the associated primary window if you can avoid it. Note that *4Dwm* constrains support windows to always appear on top of the parent window in the window hierarchy.

A support window should be smaller and less complex than its associated primary window, so that you don't need to split the support window's contents into separate panes. Support windows typically include a related set of controls that are associated with the parent primary window. Each related set of functions or input fields should be given its own support window. The controls (which can include buttons, text fields, and scrolling lists) typically either operate directly on selected data or change the mode of the primary window. For example, they might allow the user to choose a texture that will be applied to the selected objects in the primary window, or they might allow the user to choose a specific drawing tool that changes what's drawn in the parent window. For example, the IRIS Showcase Align Gizmo shown in Figure 6-9 aligns the objects that are currently selected in IRIS Showcase's main window. You shouldn't add or remove controls from a support window depending on the current context—the layout and contents of a support window should be static.

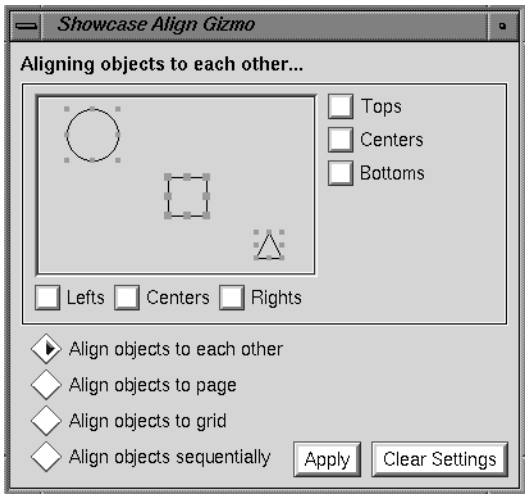


Figure 6-9 The IRIS Showcase Align Gizmo

Support windows also typically contain a response area that includes standard actions for the window: "Apply," "Cancel/Close," and "Help." See "Standard Dialog Actions" in Chapter 10 for more information about these actions. In addition, support windows may contain secondary work areas for manipulating data that will eventually be integrated into the work area of the associated primary window. Texture, pattern, icon, and geometry editors are examples of support windows that might contain secondary work areas. The Align Gizmo in Figure 6-9 contains a small display area showing a circle, a square, and a triangle, which shows the effects of the user's changes before they're applied to the main window.

Support windows should be modeless—that is, they shouldn't prevent the user from interacting with any of the application's other windows. If your application requires a secondary window that the user must dismiss before interacting with the rest of the application, you should use a modal dialog (see "Dialog Modes" in Chapter 10).

A Specific Standard Support Window: The Indigo Magic Color Chooser

The Indigo Magic color chooser is a standard support window that allows users to edit colors. You should use the color chooser in your application whenever you want to offer the user an unrestricted choice of colors. For a restricted choice of colors, you can offer the user a palette of colors to choose from, a list, an option button, or a set of radio buttons, depending on the number of choices available. Figure 6-10 shows the color chooser in its default configuration.

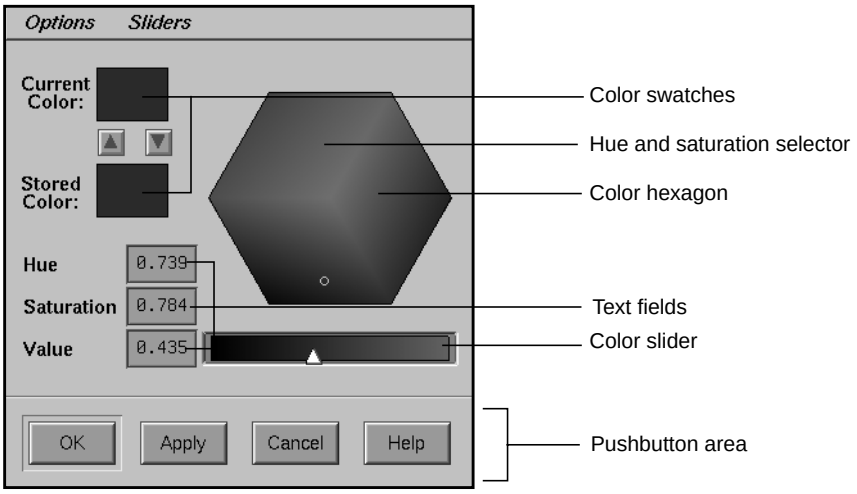


Figure 6-10 The Indigo Magic Color Chooser

You can allow users to access the Indigo Magic color chooser from your application in one of two ways: by having them click on a button that displays its (editable) color, or having them click on an object for which the color should be changed. The first method is used by the Background control panel (which is available from the Desktop->Customize menu in the Toolchest). With this panel, the user clicks on one of the color buttons to open the color chooser. If the color chooser is already open, clicking on a color button selects that color for the color chooser to edit. The colors of the buttons represent the current colors being used by the desktop background. With the second method, the user selects an object and then chooses the “Color Editor” entry from the Edit menu, as described in the section “Edit

Menu" in Chapter 8. This menu entry opens the color chooser. For details on how to include the Indigo Magic color chooser in your application, see "The New Widgets" in Chapter 4 of the *Indigo Magic Desktop Integration Guide*.

As noted in Figure 6-10, the color chooser includes the following components:

- Two color swatches: one for showing the current selected color and one for enabling the user to store a second color for reference.
- A color hexagon that provides visual selection of the hue and saturation components of a color in an HSV color space. The user changes the hue and saturation by moving the selector (which appears as a small circle) in the color hexagon.
- Color sliders for controlling various color components.
- Text fields that show the exact values for hue, saturation, and value color components and allow users to set these values numerically. (There are also text fields indicating the values of the red, green and blue color components when the red, green, and blue sliders are visible.)
- Menus for Options (which allows users to easily find the color white) and Sliders (which provides various combinations of sliders for setting hue, saturation, value, red, green, blue input values).
- Pushbuttons that allow users to apply the current values, cancel a pending change, or get help on this window.

Note: Because of drawing-speed considerations, the color hexagon and color sliders are available only if running under GL. For X-only configurations, the Color Chooser uses a Scale widget instead of the color sliders, and there is no color hexagon.

The user can apply the new color to the selected object by pressing either the *OK* or *Apply* buttons. If the user presses *OK*, the color chooser should be dismissed after the new color is applied. If the user selects a new object in the parent primary window while the color chooser is open, the color chooser should update its current color to the color of the selected object. Thus, a single color chooser window can be used to change the color of a number of different objects.

Support Window Guidelines

When designing support windows . . .

- ❑ Use them to provide access to sets of related controls.
- ❑ Allow users to access them either through entries in the Tools menu or through pushbuttons in a tool palette in the parent primary window.
- ❑ Be sure that each support window has a visible parent primary window that's mapped to the screen.

When designing the layout of a support window . . .

- ❑ Make the layout simple and static. Don't include multiple panes of information.
- ❑ Include a response area for standard actions that's similar to the one dialogs have.
- ❑ Don't include a menu bar in most cases, but do support the keyboard accelerator for Close (Ctrl-W).

When opening support windows . . .

- ❑ Avoid overlapping the work area of the parent window.
- ❑ Bring them up as modeless secondary windows.

When allowing the user to make color choices . . .

- ❑ Use the Indigo Magic color chooser whenever you want to offer the user an unrestricted choice of colors. For a restricted choice of colors, use a palette of colors to choose from, a list, an option button, or a set of radio buttons, depending on the number of choices available.

Pointer Behavior in a Window

The user should retain control over the location of the pointer at all times. Your application shouldn't change the location of the pointer. (This is sometimes referred to as "warping" the pointer.) Similarly, your application shouldn't change the gain and acceleration characteristics of mouse movement. Users set these on a global basis using the Mouse Settings control panel available from the Desktop->Customize cascading menu in the

Toolchest. If your application requires finer motion control than what's provided by the default gain settings, provide a zoom feature in the View menu that allows users to change the relative size of an area of your application. (See "View Menu" in Chapter 8 for more information about this menu.)

Although users control the location of the pointer, your application needs to control the shape of the pointer. This shape gives the user feedback about the current state of the application (for example, whether it's waiting for user input or whether it's busy processing). Pointer shapes are discussed in "Pointer Shapes and Colors" in Chapter 11.

Pointer Behavior Guidelines

When designing your application . . .

- ❑ Always allow the user to control the location of the pointer; your application shouldn't change the position of the pointer.
- ❑ Don't change the gain or acceleration characteristics of the pointer. If your application requires fine manipulation, provide a zoom feature in the View menu.

Chapter 7

Focus, Selection, and Drag and Drop

Users can interact with your application through three general mechanisms, which are discussed in the following sections:

- "Keyboard Focus and Navigation" discusses how your application should allow users to direct keyboard input to specific components. It also discusses how certain components should be controlled from the keyboard.
- "Selection" discusses various models for allowing users to select data in your application.
- "Drag and Drop" discusses how users expect to directly manipulate text and other objects in your application by dragging them with the mouse.

Keyboard Focus and Navigation

Keyboard input allows users to enter data into text fields and to control other components in your application. The keyboard focus policy determines which component in which window receives the keyboard input. Only one component in one window receives input from the keyboard at any given time; this component has the keyboard focus (also called *input focus*). For example, if a button has the keyboard focus and the user presses the Space bar on the keyboard, the button is activated. The process of moving the keyboard focus is called *navigation*. *Keyboard navigation* allows the user to navigate among components in a window using only the keyboard rather than having to manipulate the mouse (or other pointing device).

As described in "Keyboard Focus Across Windows" in Chapter 3, the Indigo Magic environment uses one policy for moving the keyboard focus among components within a window and a different policy for moving the keyboard focus between windows. When moving the keyboard focus among components within a window, your application should use an

explicit focus policy. In other words, the user clicks a mouse button or presses a key to move the keyboard focus to a new component in the active window. In contrast, *4Dwm*, the window manager for the Indigo Magic Desktop, uses implicit focus across windows: the window directly underneath the pointer receives keyboard input (that is, it's the *active window*). Note that users can't navigate among windows using the keyboard when using *4Dwm* in its default configuration.

This section discusses keyboard focus and navigation among components in the active window.

Keyboard Focus Policy and Navigation Within a Window

Only one component in the active window has the keyboard focus at any given time. Your application should use *explicit* focus (as opposed to implicit focus) within a window; in other words, the user must explicitly select the component that receives the keyboard input. Your application should support the models described in this section for navigating to specific components in a window and for using the keyboard to activate these components.

Within the active window, the component with the keyboard focus is visually identified by the location cursor. The location cursor isn't necessarily a cursor in the traditional sense of a text cursor. It gives the user visual feedback as to which component receives the keyboard input. Each standard component described in Chapter 9, "Controls," has its own method for displaying a location cursor when the component has keyboard focus. For example, the location cursor used to indicate that a specific radio button has the keyboard focus is a simple box, as shown in Figure 7-1.

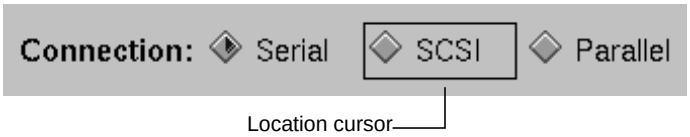


Figure 7-1 Location Cursor Example

Keyboard activation and keyboard navigation are strongly linked: if a user can activate or control a component from the keyboard, the user should also be able to navigate to that component from within the window using the

keyboard. This enables the user to perform the task without having to frequently switch between using the mouse and keyboard.

Section 2.2 in the *OSF/Motif Style Guide* states that “all application functionality must be available from the keyboard alone.” This includes navigating among windows, navigating among components in a window, and activating components. By default, users will be able to navigate to and control all components in a window except for those that aren’t traversable or don’t accept input (for example, labels and separators).

Since all Silicon Graphics systems come with a mouse, it isn’t as critical to provide access to all functionality from the keyboard alone when programming for Silicon Graphics systems. Just keep in mind that some users use alternate input devices that rely on having functions available from the keyboard. At a minimum, your application should let users do the following from the keyboard:

- navigate between editable text fields in a window
- enter data into editable text fields
- select data in a text field (see “Selection” later in this chapter and “Text Fields” in Chapter 9)
- navigate to a list component (see “Lists” in Chapter 9)
- select data in a list (see “Selection” later in this chapter and “Lists” in Chapter 9)
- navigate among all types of menus (pull-down, popup, and option menus) and their entries (see “Menu Traversal and Activation” in Chapter 8)
- activate menu entries (see “Menu Traversal and Activation” in Chapter 8)
- scroll any scrollable component (see “Scrollbars” in Chapter 9)
- activate the default button in a dialog if there is one (see “Standard Dialog Actions” in Chapter 10)
- use mnemonics for all menu titles and menu entries in the pull-down menus (see “Choosing Mnemonics” in Chapter 8)

- use keyboard accelerators for frequently used entries in the pull-down menus, such as “Cut,” “Copy,” and “Paste” in the Edit menu (see “Choosing Keyboard Accelerators” in Chapter 8)

Keyboard Navigation

This section discusses guidelines for moving the focus to a different component in the window using the keyboard. (The *OSF/Motif Style Guide* refers to this as *component navigation*.) Each window is divided into fields, where a field can be an individual control (for example, a text input field) or a group of controls (such as a group of radio buttons). By default, the fields that can accept the keyboard focus are ordered, in general, from upper left to lower right. If a window has multiple *panes*, the focus moves by default through the fields in the topmost (or leftmost) pane, then the fields in the next pane, and so on, until it wraps back to the beginning.

In some cases, you may have to modify the default order in which components are navigated from the keyboard. For example, when a window first becomes active, the component that should have the keyboard focus is the one that the user is most likely to want to interact with *using the keyboard*. This isn’t necessarily the component in the upper left-hand corner. Also, when a user returns the keyboard focus to a window that was previously the active window, the keyboard focus should return to where it was when the user moved the focus out of that window.

By default, users can cycle through the fields in order using the <Tab> key; in addition, they can use the arrow keys to move the keyboard focus among the individual components in the current field. For example, in the Add Printer window, shown in Figure 7-2, a user could use <Tab> to move keyboard focus from the first field (“New Printer Name”) to the second field (“Connection Type”). Once in the second field, the user can move keyboard focus between the radio buttons using the directional arrow keys.

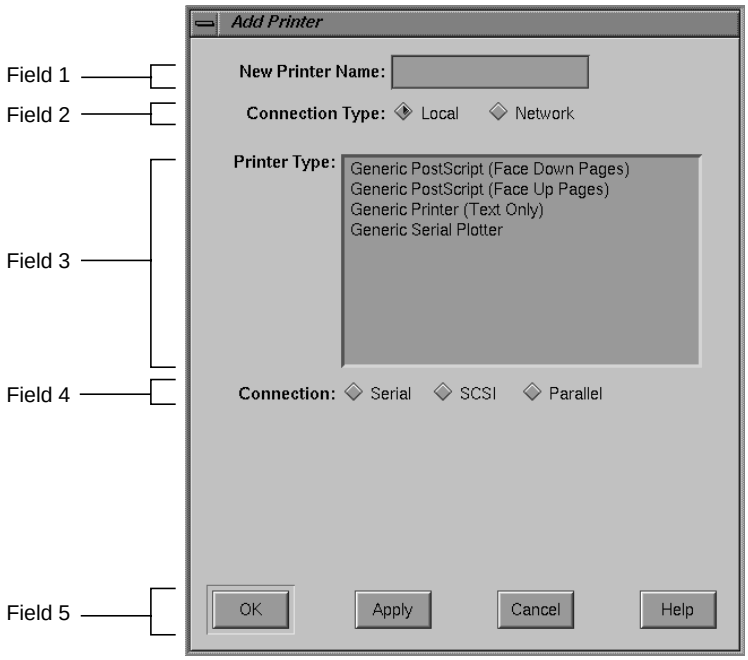


Figure 7-2 Components and Fields

By default, the following keyboard commands are used for navigating within a window. In addition, as discussed in “Menu Traversal and Activation” in Chapter 8, <Shift>-<F10> should move the location cursor to a popup menu if one is available for the current context.

<Tab> Moves the location cursor to the next field that can accept the keyboard focus, unless the current field is a multi-line editable text field. In this case it simply inserts a tab character.

<Ctrl>-<Tab> Always moves the location cursor to the next field that can accept the keyboard focus.

<Shift>-<Tab> Moves the location cursor to the previous field that can accept the keyboard focus.

<Ctrl>-<Shift>-<Tab>
Always moves the location cursor to the previous field that can accept the keyboard focus.

<down arrow>	Moves the location cursor within a field forward (or down) to the next component that can receive the keyboard focus, eventually wrapping back to the first component. If the components are in a matrix, <down arrow> moves down through a column and then proceeds to the top of the next column to the right.
<up arrow>	Moves the location cursor within a field opposite to the direction of the <down arrow> to the next component that can receive the keyboard focus. Eventually it wraps back to the last component.
<right arrow>	Moves the location cursor within a field to the next component to the right that can receive the keyboard focus, eventually wrapping back to the first component. If the components are in a matrix, <right arrow> moves across an entire row and then proceeds to the row below.
<left arrow>	Moves the location cursor within a field opposite to the direction of the <right arrow> to the next component that can receive the keyboard focus. Eventually it wraps back to the last component.
<F10>	Moves the location cursor to the leftmost menu in the menu bar if there is one. If a menu is already displayed, <F10> closes the menu and returns the location cursor to where it was previously. (See “Menu Traversal and Activation” in Chapter 8.)

Because the keys listed above are used for navigating among components, don’t use them for other purposes. However, there’s an exception to this rule: the arrow keys can be used to control a component that’s the only component in its field. For this reason, each editable text field, list, scrollbar or sash is by default placed in its own field.

Mouse Navigation

To move the keyboard focus in the current active window using the mouse, users position the pointer over a specific component and click the left mouse button. This moves the keyboard focus to the selected component, if you’ve allowed that component to accept keyboard focus. This also typically performs some action or selects some data. For example, clicking the left mouse button on a pushbutton activates the pushbutton, as well as moves

keyboard focus to the pushbutton. Clicking the left mouse button in an editable text field moves keyboard focus to the text field and places the insertion point in the text field at the pointer location. If users want to move the keyboard focus to a component using the mouse without activating that component, they can position the pointer over the component, then hold down the <Ctrl> key while clicking the left mouse button.

By default, there are certain components that don't grab the keyboard focus when activated using the mouse. These include scrollbars, sashes, any other component that's used only to change the size or location of other elements, and any components that you've designated as being unable to accept keyboard focus. If the user uses the mouse to activate any of these components, it's activated and the keyboard focus stays where it was.

Keyboard Focus and Navigation Guidelines

When designing keyboard focus and navigation for your application windows . . .

- ❑ Use explicit focus for navigating among components within a window.
- ❑ Support at least the minimum required functionality from the keyboard, such as navigating to and entering data into editable text fields, using mnemonics and keyboard accelerators to access menu entries, and scrolling any scrollable component. Keep in mind that some users use alternate input devices that rely on having functions available from the keyboard.
- ❑ When the window becomes active for the first time, give focus to the component that the user is most likely to want to interact with using the keyboard. When a user returns the keyboard focus to a window that was previously the active window, return the keyboard focus to where it was when the user moved the focus out of that window.
- ❑ Put each component that requires the use of arrow keys to control it in its own field. The following components are by default put in fields of their own: editable text fields, lists, scrollbars, and sashes.
- ❑ Don't use the default keyboard navigation keys for other purposes. These keys are <Tab>, <Ctrl>-<Tab>, <Shift>-<Tab>, <Ctrl>-<Shift>-<Tab>, the arrow keys, <F10>, <Shift>-<F10>, and <Ctrl> in combination with a left mouse button click.

Selection

IRIS IM is based on the object-action model of direct manipulation. This means that a user must first select an object or group of objects, then choose an action to perform on that data. Users typically select data by clicking the left mouse button (to select a single object) or by dragging with the left mouse button (to select a range of objects). The selection is completed when the mouse button is released. Making a selection shouldn't automatically perform any operation on that selection. When users select data in an application window, that data should be highlighted in some way so that when they pick an action, they'll know which chunk of data that action is being applied to. At any time, there's one selection that's the *primary selection*. This is the last selection explicitly started by the user and is used to copy data between applications. For details on supporting the primary transfer model in your application, see "Supporting the Primary Transfer Model" in Chapter 5.

Selection Models—What Can Be Selected and How To Select It

The data in an application window is divided into *collections*. A collection is a group of related elements that share a selection model. There are four basic selection models described in the *OSF/Motif Style Guide*:

- In the *single selection* model, only one element in the collection can be selected at any given time. For example, a color palette usually allows you to pick only one color at a time.
- The *browse selection* model is essentially the same as the single selection model, except that it allows users to browse through the available elements by dragging with the left mouse button. The list of available schemes in the Schemes control panel is an example of this model.
- In the *range selection* model, more than one element in the collection can be selected at any given time, but these elements must be next to each other. Text is usually selected in this fashion—a user can select any number of contiguous characters in a piece of text.
- In the *discontiguous selection* model, more than one element in the collection can be selected at any given time, and these elements don't

have to be next to each other. An example of this model is a list of files that allows a user to select multiple files.

Note that the *OSF/Motif Style Guide* also describes a fifth selection model, *multiple selection*. Your application shouldn't use this model because it uses mouse actions for adding and removing selected elements that are different from other mouse actions. Eliminating these inconsistent mouse actions for selection makes it much easier for users to learn how to select data.

Each collection of data in your application should support the mouse and keyboard actions for selecting and deselecting data listed in Table 7-1, depending on which of the above models it supports. By default, the IRIS IM list component supports the browse selection model, and the IRIS IM text component supports the range selection model.

Table 7-1 Selection Actions and Results

Action	Model	Result
Click on an element in the collection	All	The element is selected, and any elements in the collection that were previously selected are deselected. The location cursor is moved to the selected element.
Drag through a range of data in the collection	Browse	As the user moves the pointer over each element in the collection, that element becomes selected and all other elements in the collection are deselected. When the user releases the left mouse button, the element currently under the pointer remains the selected item, and the location cursor is moved to this element.
Drag through a range of data in the collection	Range and discontiguous	Any elements in the collection that were previously selected are deselected, and an *anchor* is set on the element or at the location where the left mouse button was pressed. While the user continues to drag the mouse, all elements between the anchor and the current location of the pointer are selected. When the user releases the mouse button, the current selection is set to all the elements between the anchor and the location of the pointer when the mouse button was released.
<Shift>-click on an element or <Shift>-drag through a range of elements in a collection	Range or discontiguous	The anchor is left in place, and the current selection is modified using one of three models for extending a range described in Section 4.1.4 of the *OSF/Motif Style Guide*. The preferred model is the *balance beam* model, which is also the default for the IRIS IM text component.
<Ctrl>-click on an element in the collection	Discontiguous	The selection state of the element is toggled, and the anchor and location cursor are moved to that element.

Table 7-1 (continued) Selection Actions and Results

Action	Model	Result
<Ctrl> drag through a range of data in the collection	Discontiguous	The selection state of the range of elements is toggled based on the *anchor toggle* model described in the *OSF/Motif Style Guide*, section 4.1.5. That is, you pick the element in the range that is closest to the anchor and set all of the elements in the range to the inverse of the selection state of this element.
Click outside of the selection (but not on any element in a collection that requires at least one element to be selected at any given time)	All	All elements are deselected.
When all of the data in a collection is selected, click anywhere inside the collection	Range and discontiguous	All elements are deselected.
<Esc> while in the process of making a selection in any collection of data.	All	The current selection action is cancelled, and all user input is ignored until the user has released all keys and buttons. The selection state is returned to its previous state.
<Ctrl>-</> when the collection has keyboard focus	Range and discontiguous	All elements in the collection are selected. The anchor is placed at the beginning of the collection. The location cursor remains unchanged.
<Ctrl><\> when the collection has keyboard focus	Range and discontiguous	All elements in the collection are deselected. The location cursor remains at its current position, and the anchor is moved to where the location cursor is.

When users select data in a component that can be scrolled, the component should support automatic scrolling—that is, if the data being selected is in a scrollable component and the user drags the pointer out of the data display region while still holding down the mouse button, the data should scroll in the direction of the pointer and should continue to be selected. Note that this behavior is automatically supported in the IRIS IM list and text components.

The mouse and keyboard actions described above represent a subset of those defined in the *OSF/Motif Style Guide*, which requires that all functionality be available from the keyboard. The *OSF/Motif Style Guide* describes specific keyboard actions to select individual elements, select a range of data, and modify the data selected. If you determine that users will want to access any of this functionality in your application using the keyboard, see Section 4.1.6 of the *OSF/Motif Style Guide* for details on supporting keyboard selection. Note that keyboard selection is automatically supported in IRIS IM list and text components.

Highlighting a Selection

When the user initiates and continues to add to a selection, your application should visually highlight the currently selected data. In addition, while the data in the collection is being adjusted, the currently selected data should always be highlighted to show users what would be selected if they were to release the mouse button immediately. Selections should remain highlighted, even when the window containing that selection is no longer the active window. The *OSF/Motif Style Guide* refers to this as *persistent always* highlighting. This is the best type of selection highlighting to use when implicit focus is used for moving the keyboard focus across windows (implicit focus is the default for 4Dwm, as explained in "Keyboard Focus Across Windows" in Chapter 3).

Use persistent always highlighting except when the only reason a user can make a selection is to transfer that data using the primary transfer model and the user cannot perform any other actions on this data. (The primary transfer model is discussed in "Supporting the Primary Transfer Model" in Chapter 5.) For this type of data, your application should use *nonpersistent* highlighting, which means that the selection is highlighted only when it's the primary selection. When this data is no longer the primary selection, the currently selected data is no longer highlighted and the current selection is set to empty.

Multiple Collections in One Application Window

This section describes some common ways that multiple collections of data might interact in a single application window. There are three basic scenarios for using multiple collections in the same window:

- The user can select data in only one collection at a time.
- The user can select data in more than one collection at a time, and any given mouse, keyboard, or menu command applies to only one of the collections.
- The user can select data in more than one collection at a time, and some mouse, keyboard, or menu commands can be applied to more than one of the collections.

If the user can select data in only one collection at a time, deselect the previous selection whenever the user makes a new selection in any of the collections. If the user can select data in more than one collection at a time, and any given mouse, keyboard, or menu command applies to only one of the collections, don't do anything special. Since each action can be applied only to one collection, it's obvious which collection to apply it to. For mouse, keyboard, or menu commands that can be applied to more than one of the collections, apply the operation to the collection that most recently had a selection made in it. (See "Keyboard Focus and Navigation" earlier in this chapter.)

Selection Guidelines

For each collection of data . . .

- Use one of the four recommended selection models—single selection, browse selection, range selection, or discontiguous selection. Don't use the multiple selection model.
- Automatically scroll the data as the user drags the pointer out of the scrollable data display region.
- Determine if your users will need to create or modify a selection using the keyboard. If so, then support the keyboard actions defined in Section 4.1.6 of the *OSF/Motif Style Guide*. (These actions are automatically supported if you use the IRIS IM list or text components.)

When highlighting a selection . . .

- Update the highlighting continuously as the user initiates and extends the selection.
- Use persistent always highlighting, unless the only reason a user can select this data is to transfer it using the primary transfer model. In this case, use nonpersistent highlighting.

When managing multiple collections of data in a single window . . .

- Deselect the previous selection whenever the user makes a new selection in any of the collections for cases where the user can select data in only one collection at a time.

- ❑ Apply the operation to the collection that most recently had a selection made in it when the user can select data in more than one collection at a time and there are mouse, keyboard, or menu commands that can be applied to more than one of the collections.

Drag and Drop

Direct manipulation, or *drag and drop*, describes an interface in which the user moves icons on the desktop or in application windows in order to perform various actions on the objects represented by the icons. Some typical uses for drag and drop include the following:

- Moving an object from one place to another by dragging the object with the mouse and dropping it on a target. For example, to move a file from one directory to another, the user drags the icon representing the file and drops it on the folder icon representing the new directory location.
- Making a reference to the object in the new location. For example, to add an online book to the personal bookshelf in IRIS Insight, the user drags an icon representing an online book from the main bookshelf to the personal bookshelf. This creates a reference to that book on the personal bookshelf in addition to the reference on the main bookshelf.
- Performing some operation on the item being dragged. For example, a user can print a file by dragging the icon that represents the file onto a printer icon.

If the user presses <Esc> during a drag and drop operation, the operation should be cancelled, and both the object and the target should be left as they were before the operation was initiated.

Two Models of Drag and Drop

There are two models for drag and drop, one recommended for use with text, and the other recommended for use with other objects.

Drag and Drop for Non-Text Objects

The most common model for drag and drop found in applications requires the user to select and drag the object using the left mouse button. This is the preferred model for implementing drag and drop of non-text objects; it reinforces the direct manipulation model of controlling objects directly using the left mouse button. The two most common scenarios for this are the following:

- The user initiates dragging the object by positioning the cursor over the object, pressing with the left mouse button and dragging the mouse. Pressing the left mouse button in this case selects the object. Dragging the mouse drags the object. (Note that if the pointer is over two different elements that can be dragged, the topmost element should be the one selected and dragged.) Releasing the mouse button drops the object on the target below the pointer location.
- The user selects one or more objects in a collection using the left mouse button. The user then positions the pointer anywhere over the selection, presses the left mouse button, and drags the mouse to drag the object(s). As with the above scenario, when the user releases the mouse button, the object is dropped on the target below the pointer location. Note that in this case, if users positions the pointer outside of the selection and begins dragging with the left mouse button, they're indicating that they want to make a new selection. See "Selection" earlier in this chapter.

For either of the above scenarios, after a drop the target should determine both the format of the data and whether the user meant to perform a move or a copy operation. In some cases, dropping the object might simply mean that the object should be moved to a new location in the same component. In the case where the drop is in the same component, after the drop the data should remain selected. For example, the user can move files around in a Directory View window using drag and drop. In other cases, dragging an object onto a target means that the object should be copied to the target so that the target can perform some operation on it. For example, when the user drags a file onto a printer icon, the file is translated into an appropriate format and sent to the printer.

To make drag and drop of file objects easy to include in your application, Indigo Magic includes a file finder component, which provides a drop pocket (see Figure 7-3). If the user drops a file icon in the drop pocket, the

text field updates to show the pathname of the file represented by the icon. If the user types in the field, the icon in the drop pocket changes to show the new choice. For guidelines on when to use this type of control in your application, see “File Finder” in Chapter 9.

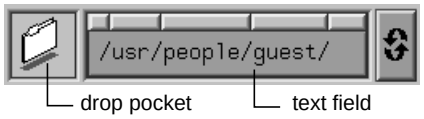


Figure 7-3 File Finder Component

Drag and Drop for Text

Drag and drop can also be implemented using the middle mouse button. Use this model of drag and drop for transferring text rather than the model described in the previous section because the left mouse button is so heavily used for selection in text.

In this case, the user selects a region of text to be dragged using the selection techniques described in “Selection,” earlier in this chapter. Then the user positions the pointer over the selected text region and drags the text with the middle mouse button. When the user releases the middle mouse button, the text is dropped on the target under the pointer. By default, all text (including labels) can be dragged using the middle mouse button. You may want to turn off drag and drop for some of the text in your application if users will never need to drag it (for example, labels).

For additional details of implementing drag and drop of text, see Sections 4.3.4 and 6.2.5 in the *OSF/Motif Style Guide*.

Pointers for Drag Operations

When selecting and dragging are integrated into the left mouse button, use the standard arrow cursor for simplicity. When drag and drop is implemented using the middle mouse button (typically for dragging text), replace the standard pointer with a *drag icon*. This reinforces to users that they’re using the middle mouse button to perform a drag and drop operation. The design of drag icons is discussed in Section 6.2.5.1 of the *OSF/Motif Style Guide*.

Drag and Drop Guidelines

When designing drag and drop for your application . . .

- ❑ Cancel a drag and drop operation if the user presses <Esc>, and leave both the object and the target as they were before the operation was initiated.
- ❑ Use the left mouse button for both selecting and dragging non-text objects. Use the standard cursor in this case.
- ❑ Use the middle mouse button for dragging text, and replace the cursor with a drag icon when the text is being dragged.

Chapter 8

Menus

Menus allow users to browse through options, settings, and commands available in your application. A well-organized set of menus shows users what your application can do and makes it easy to locate particular functions. This chapter describes the kinds of menus your application should use and how menus and menu items should be organized, in these sections:

- "Types of Menus" defines the three types of menus that your application can use: pull-down, popup, and option menus.
- "Menu Traversal and Activation" describes the default IRIS IM model for accessing menus with the mouse and the keyboard, with two additions that your application should support.
- "The Menu Bar and Pull-Down Menus" discusses how to design pull-down menus (which include cascading, or nested, menus).
- "Popup Menus" discusses how to design popup menus.

Option menus are discussed only briefly in this chapter; they're covered in detail in "Option Buttons" in Chapter 9.

Types of Menus

Indigo Magic supports three types of menus, all of which are defined in the *OSF/Motif Style Guide*: pull-down menus, popup menus, and option menus. Of the three types, pull-down menus are probably the most frequently used.

Most applications have a menu bar, which is a collection of pull-down menus. Figure 8-1 shows a typical menu bar.

Figure 8-1 Menu Bar

Each pull-down menu is represented in the menu bar by its title. A user can display a menu by pressing the left mouse button on the menu title. Figure 8-2 shows a typical pull-down menu.

Figure 8-2 Pull-Down Menu

Pull-down menus can include submenus, or *cascading menus*. A menu entry for a cascading menu is indicated by an arrowhead next to the entry, as shown in Figure 8-3. Pull-down menus are discussed in detail in "The Menu Bar and Pull-Down Menus" later in this chapter.

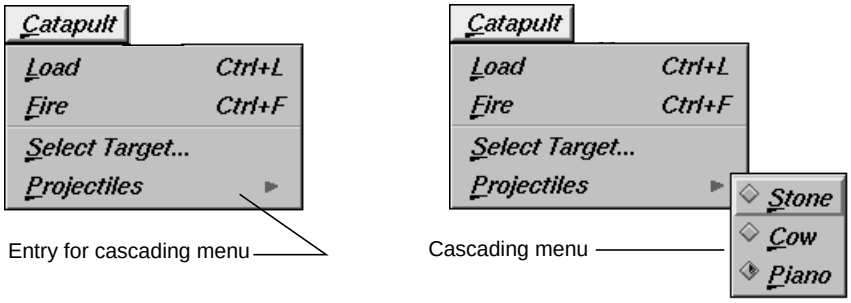


Figure 8-3 Cascading Menu

Unlike pull-down menus, popup menus are not represented by a title on the screen. A user displays a popup menu by pressing the right mouse button. The contents of the popup menu depend on where the mouse pointer is located when the button is pushed. Figure 8-4 shows a popup menu. Popup menus are discussed in detail in "Popup Menus" later in this chapter.

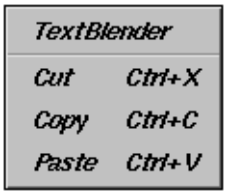


Figure 8-4 Popup Menu

Option menus allow the user to select a single option from a list of options. An option menu appears as a button marked with a horizontal bar, as shown in Figure 8-5.

Figure 8-5 Option Menu Button

The option button is labelled with the currently selected option. When a user presses the left mouse button over the option button, the option menu is displayed, as shown in Figure 8-6. If the user selects a different option from this menu, the label on the button updates to reflect this new value.

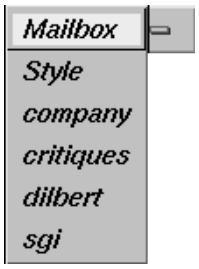


Figure 8-6 An Open Option Menu

Entries in an option menu represent mutually exclusive values of a parameter. They shouldn't be used for actions. Guidelines for using option buttons and option menus are discussed in "Option Buttons" in Chapter 9.

Menu Traversal and Activation

Pull-down, popup, and option menus should use the default IRIS IM model for menu traversal and activation, with two additions. This model is defined in the *OSF/Motif Style Guide*, Chapter 3, and summarized below. The two additional guidelines are also described in the following paragraphs. Furthermore, pull-down menus use mnemonics and keyboard accelerators for traversal and activation; these techniques are described in the next section, "The Menu Bar and Pull-Down Menus."

With the default model, users can use either the mouse or the keyboard to display, traverse, activate, and close menus. With the mouse, users have the additional choice of manipulating menus in either a *spring-loaded* or a *posted* manner:

- To display a pull-down or option menu in a spring-loaded manner, the user positions the pointer over the menu and presses the left mouse button. To display a popup menu in a spring-loaded manner, the user positions the pointer in an area of the window that has a popup menu associated with it and presses the right mouse button. The user

traverses any of these menus by moving the pointer over the menu entries while continuing to hold the mouse button. If the pointer is over a menu entry when the user releases the mouse button, that entry is activated and the menu is removed.

- To display a menu in a posted manner, the user positions the mouse pointer over the menu or over the appropriate area of the window and clicks the appropriate mouse button (left for pull-down and option menus, right for popup). The menu is then displayed with the location cursor on the first available menu entry (that is, the first non-disabled entry). To activate one of the menu entries, the user positions the pointer over the appropriate entry and clicks the left mouse button. To remove the menu, the user clicks the left mouse button anywhere outside the menu. For popup menus, the user can click either the left or right mouse buttons to select an entry or remove the menu.

In addition to supporting this default model for manipulating spring-loaded and posted menus with the mouse, your application should handle the mouse click that closes a posted menu as follows: Even though this click is passed on to the underlying application window, your application should ignore this click so that users don't lose selections they've made in the window just because they display and close menus.

By default, in IRIS IM, users can also display, traverse, activate, and close menus using the keyboard:

1. To display pull-down menus, users first press <F10> to move the keyboard focus to the leftmost menu in the menu bar and then press the down arrow key. To display an option menu, users first move keyboard focus to the option menu button and then press the space bar.
2. Once a menu is displayed, the user can use the up arrow and down arrow keys to traverse a menu. Similarly, the user can use the left arrow and right arrow keys to move from menu to menu across the menu bar.
3. Once a menu is displayed, pressing <Enter> or the space bar activates the item under the location cursor, closes the menu, and returns the keyboard focus to where it was before the menu was displayed.
4. Pressing <Esc> while a menu is displayed closes the menu and returns the keyboard focus to where it was before the menu was displayed. Pull-down menus can also be closed by pressing <F10>.

In addition to supporting this default model for manipulating menus with the keyboard, your application should allow <Shift><F10> to display a popup menu if one is available and move the keyboard focus to the first available entry in the menu. Pressing <Shift><F10> again should close the popup menu and return the keyboard focus to where it was before <Shift><F10> was pressed originally. This behavior is recommended in the *OSF/Motif Style Guide* (where <Shift><F10> is described as the substitute for the <Menu> key), but it isn't supported by default in IRIS IM.

Menu Traversal and Activation Guidelines

In general, when designing traversal and activation for your menus . . .

- Allow users to activate and traverse the menus using the default IRIS IM behaviors for mouse and keyboard actions.
- If a user closes a menu by clicking somewhere outside of the menu, the application should ignore this click so that users don't lose selections they've made in the window just because they display and close menus.
- Allow users to display and close popup menus using the key combination <Shift><F10>. When <Shift><F10> displays a popup menu, the location cursor should be on the first available menu entry. When <Shift><F10> closes the menu, the keyboard focus should be returned to where it was before the menu was displayed.

The Menu Bar and Pull-Down Menus

In most cases, each of an application's primary windows has a menu bar as described in "Menu Bars in Primary Windows" in Chapter 6. Users should be able to access most of an application's functions through its menu bars. This makes it easy for users to see what functions are available to them.

Each menu bar contains several pull-down menus. Each pull-down menu is represented in the menu bar by its title and contains entries that are either an action, a label for a cascading menu, or a separator, as shown in Figure 8-7. Also as shown, the cascading menus contain additional actions. See "What to Put in the Pull-Down Menus" for more information about the content of pull-down menus.

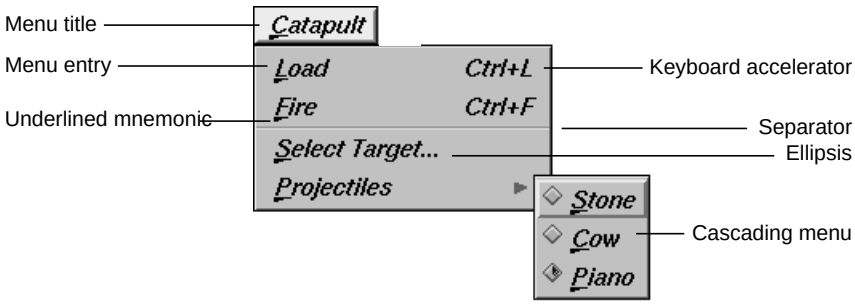


Figure 8-7 Elements of a Pull-Down Menu

Users interact with pull-down menus according to the model described in the previous section, “Menu Traversal and Activation.” In addition, users can access menu entries using mnemonics, which are the underlined characters in the menu titles and on menu entries (see Figure 8-7). To access a menu using a mnemonic, a user moves the pointer into the application window, then holds down the <Alt> key while pressing the character key that matches the underlined character in the menu title. For example, to display the Catapult menu shown in Figure 8-7, the user holds down the <Alt> key and presses the “c” key. Then, to select a projectile from the Projectiles cascading menu, the user presses the “p” key to display the Projectiles cascading menu, and then presses “p” again to select “Piano.” Note that mnemonics are always activated without the <Shift> key, even if the underlined character happens to be uppercase. Choosing appropriate mnemonics is discussed in more detail in “Choosing Mnemonics” later in this chapter.

Menu entries that represent frequently used actions can have keyboard accelerators, as shown in Figure 8-7. These keyboard accelerators are displayed in the menu next to the action and are typically a combination of the <Ctrl> key and one other key. To initiate a menu action using a keyboard accelerator, a user moves the pointer over the window to make it the active window and then presses the key combination shown in the menu entry. For example, instead of selecting “Fire catapult” from the Catapult menu in Figure 8-7, a user could fire the catapult by holding down the <Ctrl> key and pressing the “f” key. When to use keyboard accelerators and how to choose ones which are appropriate are discussed in more detail in the “Choosing Keyboard Accelerators” section later in this chapter.

A menu entry that's followed by an ellipsis (such as the "Select target..." entry shown in Figure 8-7) brings up a dialog that requests more information from the user before any action is performed. When to use an ellipsis in a menu entry is discussed in more detail later in this chapter in "Naming Menu Entries in the Pull-Down Menus." Menu entries that aren't currently available are disabled. This is usually shown by graying out the menu entry, as discussed in "Disabling Menu Entries."

When designing the menu bar and its pull-down menus for your application, you should start with the standard menus described in the next section, "Standard Menus." Then, modify these standard menus to fit your specific application, using the guidelines in the section "What to Put in the Pull-Down Menus."

Standard Menus

Your application needs its own customized set of menus and menu entries, but you should use the standard set as the starting point for the overall menu structure. All of the menu entries discussed in this chapter are optional; many of them are appropriate for most applications (but there are always exceptions), and some entries are generally less common than others. For example, "Print" is a fairly common entry for the "File" menu, but printing doesn't generally make sense for audio applications. "Import" is another entry for the "File" menu, but it's less common than "Print." Figure 8-8 shows a menu bar with all of the standard menus (in the correct order) and their mnemonics. This menu bar includes all of the menus defined in the MenuBar reference page of Chapter 9 in the *OSF/Motif Style Guide*, plus an additional menu, Tools, which is defined in the Indigo Magic environment. The standard menus are described in the following sections.

Figure 8-8 Standard Menus for Menu Bars

Note that for each of the standard menu entries described in the following sections, mnemonics are represented by the underlined character in the entry label. This label also includes an ellipsis if the entry should include them. The entries are listed in the order in which they should appear in menus, and entries that are likely to be less common are indicated. Keyboard

accelerators, if they exist, are listed in the description of the specific menu entry. Appropriate places for separators are shown in the figures depicting the standard menus. Situations where a menu entry should be disabled are included in the description of the menu entry if applicable. Also, the keyboard shortcuts are shown as they should be displayed in the menus (for example, <Ctrl>-s is shown as “Ctrl+S”).

File Menu

The File menu contains entries for actions that are performed on files, such as “Open,” “Save,” and “Print,” and on the application as a whole, such as “Exit.” Figure 8-9 shows the standard File menu with the most common entries; note that its mnemonic is “F.” These standard entries, as well as a few other less common ones (“Reopen,” “Import,” and “Revert”), are described in Table 8-1 in the order in which they should appear in the menu. All of these entries should behave as defined by the File Menu reference page in the *OSF/Motif Style Guide*, Chapter 9, except as noted in the table. Note that “New,” “Open,” “Close,” and “Exit” should display a dialog as described in “Invoking Dialogs” in Chapter 10 if there are unsaved changes to the current document.

Figure 8-9 The Standard File Menu

Table 8-1 File Menu Entries

Menu Entry and Mnemonic	OSF/Motif Behavior	Indigo Magic Additions and Exceptions	Keyboard Accelerator
New	Creates a new, empty file.	If your application allows more than one document window to be open at a time, it should create a new, empty document window; if the current document window is already empty, the action should have no effect. For more information on designing applications that support multiple open documents, see "Standard Application Models" in Chapter 6. If your application requires more information before creating a new document (for example, the user must select a template), this action may bring up a dialog to request the information. In this case, the entry label should be followed by an ellipsis.	Ctrl+N
Open	Brings up a dialog, allowing the user to choose an existing file to open.	If your application allows more than one document window to be open at a time, it should create a new document window to display the specified file; however, if the current document window is already empty, the file should be displayed in the current document window. This new document window shouldn't be a separate instantiation of the application.	Ctrl+O
Reopen[a]	Not defined.	"Reopen" allows a user to return to a file that had been previously opened by the application. Choosing "Reopen" should display a cascading menu of previously opened files; you might choose to limit the length of this list to a maximum of 10 entries. (You should disable this entry if there are no previously opened files—for example, if this is the first time the user has launched the application.) If there are unsaved changes to the current file, your application should display a dialog that asks the user whether to save or discard the changes (see "Invoking Dialogs When Manipulating Files" in Chapter 10).	

Table 8-1 (continued) File Menu Entries

Menu Entry and Mnemonic	OSF/Motif Behavior	Indigo Magic Additions and Exceptions	Keyboard Accelerator
Import[a]	Not defined.	"Import" allows a user to read an existing data file into the current application. This entry can bring up the Indigo Magic file selection dialog (in which case it should be displayed with an ellipsis), and the application should automatically determine the type of the file after it's selected. (See "Types and Modes of Dialogs" in Chapter 10 for details on the Indigo Magic file selection dialog.) Alternatively, this entry can use a cascading menu to display the types of data that your application allows users to import. Each of these entries should be followed by an ellipsis and should bring up the file selection dialog to allow the user to specify the specific file to import. Follow this entry with a separator.	
Save	Saves the current file.	Although some applications disable this entry when there are no changes to be saved, your application should never disable this entry (as described in "Disabling Menu Entries").	Ctrl+S
Save As...	Brings up a dialog and saves the current file with a new name. Also closes the previous file and opens the new one.	If the current document already has a filename, that filename should be the default value in the file selection dialog.	
Revert[a]	Not defined.	"Revert" allows a user to undo all changes made to the current file since the last time the user saved it. (This entry should be disabled if there are no unsaved changes.) Your application should display a warning dialog before executing this action, as described in "Invoking Dialogs When Manipulating Files" in Chapter 10.	
Print	Prints the current file.	If choosing "Print" brings up a dialog to allow the user to select from a list of all available printers, it should be followed by an ellipsis. If the keyboard accelerator is used to activate this entry, the print job should be sent to the default printer.	Ctrl+P
Close	Closes a window and its associated support windows and dialogs, without quitting the current application.	"Close" should be provided on co-primary windows and on support windows if they have menu bars. It shouldn't be provided on the main primary window. (See "Window Types" in Chapter 6 for definitions of these window types and "Window Decorations and the Window Menu" in Chapter 3 for details on the "Close" entry.) Applications that follow the "Multiple Document, No Visible Main" model should exit the application when the last co-primary window is closed. (See ""Multiple Document, No Visible Main" Application Model" in Chapter 6.)	Ctrl+W

Table 8-1 (continued) File Menu Entries

Menu Entry and Mnemonic	OSF/Motif Behavior	Indigo Magic Additions and Exceptions	Keyboard Accelerator
E<u>x</u>it	Closes all windows for the application and quits the application.	"Exit" should always be provided in the main primary window. If users are likely to want to exit your application from a specific co-primary window in the application, that window should include an "Exit" entry in the leftmost menu, in addition to a "Close" entry. (See "Window Decorations and the Window Menu" in Chapter 3 for details on when to use "Exit" and "Close.")	Ctrl+Q

a. These entries are probably less common than the others.

Selected Menu

The Selected menu contains application-specific actions that are performed on the currently selected objects. For example, Directory View windows on the Indigo Magic Desktop display icons representing files. Each Directory View window has a Selected menu that allows users to perform actions on the selected files, such as "Open," "Print," and "Remove." (Note that since actions in the Selected menu act on the selected data while actions in the File menu act on the entire file of data, the same entry—"Print," for example—can mean something different in the two menus.) The Selected menu shouldn't contain editing actions such as "Cut" since these should be in the Edit menu. The mnemonic for the Selected menu should be "S."

Edit Menu

The Edit menu contains actions that transfer data to or from the clipboard, actions that modify the current selection, and "Undo." It contains actions for both the clipboard data exchange model ("Cut," "Copy," and "Paste") and for the primary data exchange model ("Promote"). Both of these data exchange models are described in Chapter 5, "Data Exchange on the Indigo Magic Desktop." Figure 8-10 shows the most common entries in the standard Edit menu; note that its mnemonic should be "E."

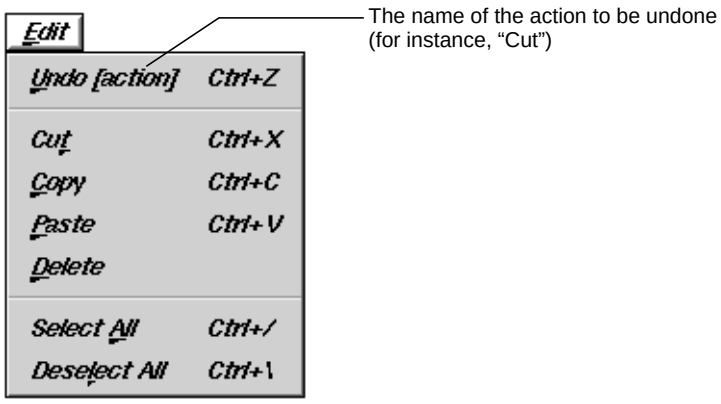


Figure 8-10 The Standard Edit Menu

These standard entries, as well as a few other less common ones (“Clear,” “Promote,” and “Color Editor”), are described in Table 8-2 in the order in which they should appear in the menu. All of these entries should behave as defined in the Edit Menu reference page in Chapter 9 of the *OSF/Motif Style Guide*, except as noted in the table.

Table 8-2 Standard Edit Menu Entries

Menu Entry and Mnemonic	OSF/Motif Behavior	Indigo Magic Additions and Exceptions	Keyboard Accelerator
Undo [action]	Reverses the effect of a previous action. The "Undo" action may apply to actions that the user accomplishes without using the menus—typing text, for example.	At a minimum, your application should be able to undo all of the actions in the Edit menu. If an undo action will change the data significantly and can't be undone, you should display a warning dialog explaining that the change can't be undone and ask for confirmation. See "Types and Modes of Dialogs" in Chapter 10 for information on warning dialogs. The "Undo" entry should be disabled if the last change cannot be undone or if there are no changes. If your application has only a single-level undo (that is, it can undo only the most recent action), after the user selects "Undo," the "Undo" entry should be changed to "Redo [action]." If the user selects "Redo" the application should reverse the effects of the previous "Undo," and toggle the menu entry back to "Undo [action]." If your application has multiple-level undo (that is, it can undo a series of actions), you should provide a separate "Redo" menu entry. Typically, applications don't allow users to undo beyond the saved version of the file; if your application does, you should bring up a warning dialog.	Ctrl+Z
Redo [action]	Not defined.	"Redo" reverses the effect of a previous "Undo" action. It is useful to have a separate "Redo" entry if your application has multiple-level undo. Like "Undo," the "Redo" entry should indicate the action that will be redone (for example, "Redo Cut," "Redo Paste"). If you provide a "Redo" command, place it after the Undo entry and follow it with a separator.	Shift+Ctrl +Z
Cut	Removes the selected data from the application window to the clipboard.	The "Cut" entry should be disabled if there's nothing currently selected in the window.	Ctrl+X
Copy	Copies the selected data to the clipboard without removing it from the application window.	The "Copy" entry should be disabled if there's nothing currently selected in the window.	Ctrl+C
Paste	Copies the contents of the clipboard into the application window.	If there's nothing currently on the clipboard available to be pasted, bring up a dialog saying there's nothing available. See "Invoking Dialogs" in Chapter 10.	Ctrl+V

Table 8-2 (continued) Standard Edit Menu Entries

Menu Entry and Mnemonic	OSF/Motif Behavior	Indigo Magic Additions and Exceptions	Keyboard Accelerator
Clear[a]	Same as "Delete," except that the remaining data isn't reorganized to fill in the space left by the cleared data.	If you provide a "Clear" command, place it before the "Delete" entry. The "Clear" entry should be disabled if there's nothing currently selected in the window.	
Delete	Removes the selected data from the application window.	The "Delete" entry should be disabled if there's nothing currently selected in the window.	
Select All	Selects all of the elements in a component of the application window.	The mnemonic for "Select All" should be "A."	Ctrl+/
Deselect All	Deselects all of the elements in a component of the application window.	The "Deselect All" entry should be disabled if there's nothing currently selected in the window. The mnemonic for "Deselect All" should be "l."	Ctrl+\
Promote[a]	Promotes the current selection to the primary selection.	"Promote" should be included if it can be difficult or time-consuming to recreate a selection in your application, and if your application supports the primary transfer model described in "Supporting the Primary Transfer Model" in Chapter 5. Disable this entry when there's no current selection or when the current selection is already the primary selection; it should be enabled only when the application window has a selection that isn't currently the primary selection. See "Selection" in Chapter 7 for information on selections. The mnemonic for "Promote" should be "m."	<Alt>-<Insert>
Color Editor...[a]	Not defined.	Choosing "Color Editor" invokes the Indigo Magic color chooser, which allows the user to select colors. (See "A Specific Standard Support Window: The Indigo Magic Color Chooser" in Chapter 6.)	

a. These entries are probably less common than the others.

View Menu

The View menu contains entries for application-specific actions that change the user's view of the current data but that don't change the actual data. For example, if your application window has several panes of information, the View menu could provide the user with a way to turn each individual pane on or off. The entries representing the individual panes should be grouped together, and there should be a checkbox in front of each one indicating

whether the pane is currently being displayed or not. (See “Splitting Primary Windows Into Panes” in Chapter 6 for information on multiple-pane windows. Using checkboxes in menus is discussed in more detail in the section on “Using Radio Buttons and Checkboxes in Pull-Down Menus.”) Other entries in the View menu might adjust the scale of the view (zoom in and zoom out), display support elements (such as rulers and grid lines), and hide or display certain parts of the data. The mnemonic for the View menu should be “V.”

Tools menu

The Tools menu contains application-specific entries that allow the user to open support windows for manipulating the data in the parent primary window. For example, a desktop publishing package might have separate support windows that provide special controls for editing graphics, tables, and mathematical equations; access to these support windows would be placed in the Tools menu. See “Support Windows” in Chapter 6 for a discussion of support windows. The mnemonic for the Tools menu should be “T.”

Options menu

The Options menu contains application-specific entries that allow the user to customize the application. For example, a multi-window application might have entries in the Options menu to allow the user to set preferences such as which windows should come up by default when the application is started, and whether window sizes and positions should be saved between sessions. The mnemonic for the Options menu should be “O.”

Help menu

The Help menu contains entries for actions that provide several different kinds of help information to the user. All application windows that have a menu bar should contain a Help menu. Its mnemonic is “H.” The standard entries for the Help menu are discussed in “Providing Help through a Help Menu” in Chapter 4.

What to Put in the Pull-Down Menus

Your application's pull-down menus should include whatever standard menu entries are relevant for your application, plus whatever application-specific entries you need to represent your application's core functionality. The previous section, "Standard Menus," describes when and how to use the standard entries; this section presents guidelines for application-specific modifications and additions to the standard menus. As you decide which of the standard entries to include in your application, consider each of these entries on a case-by-case basis. For example, you almost certainly need an "Exit" entry, but it's possible that none of the other standard File menu entries make sense for your application, including the "File" menu title itself.

Users often learn the functionality of a new application by scanning the menus to see what actions are available and by browsing the online help. Also, when users want to perform some action, they usually look first for that action in the pull-down menus. Thus, all simple, frequently accessed actions should be accessible from the pull-down menus. Be sure to include actions for performing basic operations (such as "Cut," "Paste," or "Save"), for setting the value of an attribute (for example, make selected text bold, or turn grid lines on or off), for online help, and for "Undo" (particularly if users can perform actions that destroy or significantly change their data) . If this important functionality is hidden in a dialog, users won't easily discover it. (See Chapter 10, "Dialogs" for details on when to use dialogs.) Also, don't include more than 10-12 entries in a menu or users will have trouble scanning it; all of your entries should be able to fit on the screen at one time because IRIS IM doesn't support scrolling menus.

Actions that are accomplished using buttons in primary windows should be repeated in the pull-down menus because they're probably the most frequently accessed actions. Including them in the menus gives users one place to look for all actions and allows you to assign keyboard accelerators and mnemonics that are clearly shown in the menu entries. Users should have the option of using the keyboard for frequently used actions rather than being restricted to pointing-and-clicking on buttons. In addition, all simple actions should have an associated menu command even if there's a direct manipulation method or mouse double-click shortcut available for accomplishing the task. Providing menu commands avoids hidden functionality, and helps those users who have difficulty performing double-clicks.

If you think that your users need constant access to a group of actions, your application should make these actions available in a support window. As a second choice, you can use a tear-off menu as described in the *OSF/Motif Style Guide*, section 6.2.3. Support windows are designed to include groups of controls that the user might want to use continuously. Support windows allow for a more flexible layout of controls than tear-off menus do, and support windows can contain all kinds of components, such as labels and text input fields, not just push buttons. (See “Support Windows” in Chapter 6 for information on designing support windows.) Users should be able to access such support windows as well as co-primary windows from the menu bar of their parent window.

Naming Menus in the Menu Bar

All menus in the menu bar should have one-word (capitalized) titles since users may interpret a second word as a separate menu title; don’t use bitmaps as menu titles. Use the standard titles for menus (for example, File and Edit) if they’re applicable to your application, but don’t use a standard title if you’re changing the standard definition. (See “Standard Menus” for standard menu titles and their definitions.) The leftmost menu contains actions that operate on a logical unit of data for the application; it’s generally titled “File” because most applications read and write data files. However, if your application doesn’t manipulate data files, the leftmost menu should reflect the unit of data that the user expects to operate on. For example, the Search tool’s leftmost menu is called Page because the application doesn’t manipulate data files, but it does offer several different pages that define search categories.

If your application does read and write data files but the word “File” might be confusing to users, choose a more appropriate title for the leftmost menu. For example, in MediaMail, a group of documents (mail messages) are stored in a single file referred to as a mail folder. The leftmost menu in the main window is named “Folder” to make it clear that it contains actions that apply to the entire folder of messages rather than to individual messages. If the menu were named “File,” it might not be clear whether the “Open” entry opened a message or a mail folder. Similarly, you can change the names of other standard menus to make them more meaningful. The second menu in the main window of MediaMail is named “Message” because all operations in that menu are performed on the selected messages. This menu could have been called the Selected menu, but “Message” makes it clear what the menu entries act on.

Naming Menu Entries in the Pull-Down Menus

As with menu names in the menu bar, use the standard names for menu entries within pull-down menus if they're applicable to your application; don't use a standard name if you're changing the standard definition. (See "Standard Menus.") Menu entries should be capitalized using the same rules for capitalizing book titles: capitalize the first word and other non-articles, but don't capitalize articles unless they're the first word. Generally, a menu entry should be one of the following:

- A verb—such as "Cut," "Copy," or "Delete."
- A value for a parameter, when the action is to set the parameter to that specific value. For instance, IRIS Showcase has a "Grids" cascading menu with entries corresponding to grid sizes such as "1/8 inch" and "1/4 inch."
- An attribute name, when the action is to assign some entity that attribute—such as whether shapes should be drawn filled or unfilled.
- A window name, if the menu entry brings up a co-primary, support, or dialog window. For example, a Directory View window has a menu entry for setting permissions. This entry brings up a dialog named "Permissions," so the menu entry is also named "Permissions," rather than "Set Permissions."
- The name of a cascading menu (see "Using Cascading Menus" later in this chapter).

If none of these categories applies, choose a one- or two-word phrase that indicates clearly what action will be taken. Include the name of the object that will be acted on if it's needed for clarity. For example, "New," generally indicates that a new data file will be created. If your application doesn't create data files, the menu entry for creating a new entity should be "New *object*" such as "New Folder" (Directory View windows) or "New Page" (Icon Catalog).

You can use graphic labels for menus entries, but keep in mind that graphic labels are often unclear. They work best when used along with a text label, and typically there's not enough room for both graphic and text labels in a menu entry. For those cases where graphics are better descriptions than text (for example, when showing bitmaps or textures), you should probably include these options in a tool palette either as individual buttons or as

entries in an option menu. See “Pushbuttons” and “Option Buttons” in Chapter 9 for more information about these alternatives.

If the entry is something that toggles its state, use one of the following alternatives:

- Toggle the menu entry name to indicate the action that will be taken if the user selects this entry. For example, a menu entry “Show Grid” indicates that the grid isn’t currently shown and that choosing this item will display it. If the user chooses this item, the grid is displayed and the entry should toggle to “Hide Grid.”
- Choose a menu entry name that clearly indicates what action will be taken, place a checkbox indicator next to the menu entry, and use the checkbox to indicate whether or not the action has been taken. For example, the menu entry “Italics” with a checkmark next to it indicates that the current font is an Italic one; the same entry with no checkmark indicates that the current font isn’t Italic. (For more details on the use of checkboxes in menus, see “Using Radio Buttons and Checkboxes in Pull-Down Menus” later in this chapter.)
- If the entry belongs to a group of related entries, all of which toggle their states, place checkbox indicators next to each of them. The entry names should be nouns or attributes that clearly imply their actions (and these names should remain constant rather than toggling), and the entire group should be separated from other menu items by separator lines. (See “Using Radio Buttons and Checkboxes in Pull-Down Menus.”)

A menu entry should be followed by an ellipsis if it brings up a dialog for the purpose of requesting more information from the user before performing the action. The ellipsis does *not* simply mean that the menu entry displays a dialog. For example, the “Save As...” menu entry brings up a dialog that asks the user to enter additional necessary information before the action can be performed. “Help,” on the other hand, brings up a dialog that contains the information that the user requested.

Ordering Menus and Menu Entries in the Pull-Down Menus

Use the order described in “Standard Menus” for standard menus in the menu bar and their entries. If you need to create additional menus for your application, place them between the View and Tools menus. If you need to

change the name of one of the standard menus (as discussed earlier in "Naming Menus in the Menu Bar"), leave this menu in the same order as if it had the standard name. For example, in MediaMail the File menu is renamed Folder and the Selected menu is renamed Message, but Folder is still the leftmost menu and Message is still next to the Folder menu.

Within menus, organize entries into logical groups. If one of your application-specific menu entries is logically related to one of the standard menu entries, place it near that standard entry. If this isn't a good fit, create new menus that group the entries according to function. For example, the Directory View window has an Arrange menu that contains different options for arranging the file icons displayed in the window. Within the logical groups, first place entries in the order in which they need to be used. For example, in the Edit menu, "Copy" is before "Paste" because the user must do a "Copy" operation before doing a "Paste." Secondarily, order them by frequency of use, placing the more frequently used entries closer to the top of the menu. In any case, be sure that when you use "Close" and "Exit," they're always at the end of the leftmost menu, whether or not this menu is named File.

When creating logical groups of entries, use separators to define the groups, but avoid overusing separators because they can make it difficult to scan the entries. Two situations where separators are especially useful are for groups where only one of the entries can be selected at any one time, and groups whose entries represent multiple attributes that can be applied to a single object. As described later in "Using Radio Buttons and Checkboxes in Pull-Down Menus," you should also use radio buttons in the first case and checkboxes in the second.

If the menu contains entries that can be determined only when the user launches the application (for example, a menu listing plug-in modules), alphabetize the entries. If this alphabetized group appears in a menu that contains other entries, place the group at the end of a menu and use a separator between it and the preceding entries.

Using Cascading Menus

As you're organizing your menus, you can use cascading menus, but don't use more than a single level. If you think you need more than one level of cascading menus, try adding a new menu instead, especially if you have numerous items in the cascading menus. If you have only a few items,

consider creating groups of items by using separators, rather than putting them in separate cascading menus.

In general, try to limit your use of cascading menus since users tend to scan only the top-level menus when they're looking for a specific function or trying to learn the functionality of the application. When naming a cascading menu, use a name that suggests what it contains so that users know what functions they're likely to find. For example, in an early version of IRIS Showcase, the grid was under a cascading menu named "Editing Options" in the Edit menu, and users often weren't able to find it. Now, the different grid sizes are under a cascading menu named "Grids" in the View menu.

Using Radio Buttons and Checkboxes in Pull-Down Menus

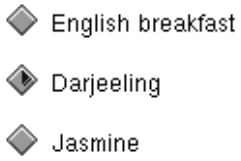


Figure 8-11 Radio buttons

If only one of a group of menu items can be selected at any one time, each item in the group should have a radio button next to it, and only one radio button should be active at any given time. For example, the radio buttons in Figure 8-11 allow the user to choose exactly one type of tea at a time—which is as it should be, because if you ordered two cups of tea at a time the second one would get cold before you could drink it. A set of radio buttons should be separated from other entries in the menu by separator lines.

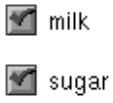


Figure 8-12 Checkboxes

If several of a group of related menu items can be selected at any one time, each item in the group should have a checkbox next to it, and the active entries should be shown with checkmarks. These items typically represent attributes of an object, more than one of which can be applied to the object. For example, the checkboxes in Figure 8-12 allow the user to select milk or sugar, or both, or neither. A set of checkboxes should be separated from other entries in the menu by separator lines.

Choosing Mnemonics

You need to choose single-character mnemonics for any menus or menu entries you create. Each of the menus in the menu bar should have a unique mnemonic, as should each of the entries within any specific menu. Use the standard mnemonics for standard menu titles and entries, as described earlier in "Standard Menus." You can use a standard mnemonic for a different entry if you're not using that standard entry.

If possible, use the first character in the label for the mnemonic. If two menu titles—or two entries in the same menu—have the same first character, use the first character for the mnemonic of the menu title or entry that will be used most frequently. For example, “Save” is used more frequently than “Save As...”, so “Save” has the mnemonic “S” and “Save As...” has the mnemonic “A.” When the first character can’t be used as the mnemonic, try to pick a consonant in the name that’s strongly associated with the word (such as “x” for Maximize in the Window menu). If no such consonant exists, choose the first available vowel (such as “a” for Raise in the Window menu). Note that the mnemonic chosen can be an uppercase or lowercase character in the label, but it must be case insensitive for activation (that is, users don’t need to hold down the <Shift> key).

Choosing Keyboard Accelerators

Use the keyboard accelerators for the standard menu entries as described in “Standard Menus”; don’t use any of the standard accelerators for application-specific entries, even if you’re not using those standard entries. For menu entries you create, provide keyboard accelerators only for the most commonly used actions, not for every menu entry in every pull-down menu. In most cases, use the <Ctrl> key and a character for a keyboard accelerator. (In order to avoid conflicts with mnemonics, don’t use the <Alt> key rather than <Ctrl>.) To make accelerators easier to remember, choose a character that’s associated with the menu entry. For example, the standard keyboard accelerators include <Ctrl-c> for “Copy” and <Ctrl-s> for “Save,” and the Directory View window uses <Ctrl-i> for “Get Info.”

If you have a pair of menu entries that both require keyboard accelerators, and one entry reverses the results of the other, their keyboard accelerators should be related. Choose a character that’s associated with the more frequently used entry (so that its accelerator is <Ctrl-*character*>), and add <Shift> to create the other accelerator (so that its accelerator is <Shift-Ctrl-*character*>, where *character* is the same for both accelerators). For example, the keyboard accelerator for Undo is <Ctrl-z>, and the keyboard accelerator for Redo is <Shift-Ctrl-z>. In general, avoid using multiple modifier keys such as <Shift-Ctrl-*character*>, except for this situation.

Note that any keyboard accelerator that involves a lowercase character should be shown in the menu as “Ctrl+*uppercase_character*” (for example, <Ctrl-s> should be displayed as “Ctrl+S”). This is because uppercase

characters are easier to read in the menus. If the accelerator involves an uppercase character, display it as “Shift+Ctrl+*uppercase_character*” (for example, <Ctrl-S> should be displayed as “Shift+Ctrl+S”).

Disabling Menu Entries

As discussed in “Standard Menus,” menu entries that aren’t currently available should be disabled (grayed out). (See the next section, “Dynamic Menu Entries,” for discussion of the rare cases in which menu entries can be removed from the menu when they’re unavailable.)

In general, disabling entries when selecting them would give the user an error message. For example, if a menu entry works on a selection (such as “Cut” and “Copy”), it should be disabled if there’s no current selection. If selecting the menu entry would result in no action at all (not even an error message), the menu entry shouldn’t be disabled. As an example, choosing “Save” from the File menu saves the current document; if the document hasn’t been edited, selecting “Save” has no real effect, but there’s no need to display an error message, so this menu entry should never be disabled.

Menu entries that launch modeless dialogs should never be disabled. If the dialog isn’t applicable when it’s launched, the OK and Apply buttons on the dialog should be disabled rather than disabling the menu entry that launches the dialog. Suppose the user launches the dialog and the current context of the application is such that the dialog isn’t applicable. Because the dialog is modeless, the user should be able to change the state of the application after the dialog has been launched to put the application in a state where the dialog is now applicable. In contrast, menu entries that launch modal dialogs should be disabled if the dialog isn’t currently applicable because the user must dismiss the modal dialog before changing the state of the application. So, if the modal dialog isn’t applicable when it’s launched, the user has no way to change the state of the application to get it in a state where the dialog would be applicable. See “Dialog Modes” in Chapter 10 for a discussion of modal and modeless dialogs.

Don’t include always-disabled menu entries whose action isn’t available in the current version of your application, so that users don’t waste time looking for a way to enable the entry. For example, if your application doesn’t provide a tutorial, don’t include a disabled menu entry for “Tutorial” in the Help menu. Instead, just leave this entry out of the Help

menu. If a feature requires certain hardware configurations, don't disable its menu entry; instead, have it display an information dialog stating why the feature isn't available.

Dynamic Menu Entries

Dynamic menu entries are strongly discouraged, especially when there are fewer than four such entries. If you have only a few entries that aren't always available, put them in the menu and disable them when they aren't available. You can use dynamic menu entries in those rare cases when almost everything in a menu can change. For example, the *grelnotes* program has a Chapter menu that has entries for each chapter in the current set of release notes. When the user loads a new set of release notes, the entries in the Chapter menu are changed to reflect the new chapter titles. Unless a more obvious ordering is suggested by the content of the entries (for example, the order of chapters), alphabetize the entries in a dynamic menu.

Dynamic menu entries are discouraged because they make it hard for users to learn what entries are in each of the menus since they're visible only when the application is in a specific state. Users are likely to assume that certain functions aren't available when they don't see menu entries for them as they're scanning your application's menus for the first time. They might not realize that they must first get the application in a particular state before they can even see the action. Even when they've been working with your application for a while, they may not look for certain actions in the menus because they think they've already seen the full contents of the menus, which never included the action that they now want. Also, users become accustomed to the spatial positions of items in menus—for example, "Cut" is always the second item in the Edit menu—and will be frustrated if these positions change.

Pull-Down Menu Guidelines

In general, when designing pull-down menus in a menu bar . . .

- ❑ Be sure that users can access most of your application's functionality from the menu entries. At a minimum, make sure that the core functionality can be accessed from the menus.

- ❑ Don't include more than a 10-12 entries in a menu and make sure that all of your entries can fit on the screen at one time.
- ❑ Provide mnemonics for all menus and menu entries. In most cases, the mnemonic should be either the first character of the name or, if there's a conflict, a character that's strongly associated with and included in the name. Use standard mnemonics for standard menus and entries.
- ❑ Limit the use of tear-off menus. Instead, use support windows for groups of controls that users might want to use continuously.

When selecting specific menus and entries for an application window . . .

- ❑ Use the standard menus and menu entries as the basis for the overall design of the menu structure. Include all standard menus and entries that are applicable to your application.
- ❑ Include a Help menu as the rightmost menu.
- ❑ Include an "Undo" menu entry, particularly if users can perform actions that destroy or significantly change their data .
- ❑ Include an "Exit" menu entry for all main windows and for co-primary windows if users will want to completely exit the application from that co-primary window.
- ❑ Include a "Close" menu entry for all co-primary windows and support windows that have menu bars. Don't provide a "Close" entry for main windows.
- ❑ Include menu entries that repeat the functionality of any pushbuttons on the primary window.
- ❑ Include menu entries for actions that are accomplished using a direct manipulation method or a mouse shortcut such as double-clicking.
- ❑ Include menu entries for accessing all primary and support windows that are children of the current window.
- ❑ Don't include entries for functions that aren't available for the current version of your application.

When naming menus . . .

- ❑ Use one-word titles for menus.

- ❑ Use the standard titles for menus (for example, File and Edit) if they're applicable, but change the standard title if this will make the function more clear.
- ❑ Don't use a standard menu title if you're changing the standard definition.

When naming menu entries . . .

- ❑ Use the standard names for standard menu entries, but don't use a standard name for a menu entry that doesn't support the standard behavior.
- ❑ Each entry name should be an action word, the value of a parameter, an attribute name, the name of a cascading menu, or the name of a co-primary, support, or dialog window. Don't use more than two words (except for task-oriented Help menu entries), and avoid using graphic labels for menus entries unless the graphics make the functionality more clear.
- ❑ Choose descriptive names that help users learn the functionality of the application. For cascading menus, choose a name that clearly implies the contents of the menu.
- ❑ Add a word if necessary to be sure the entry clearly indicates what entity will be acted upon. For example, you might use "New *object*" such as "New Folder" or "New Page" rather than just "New."
- ❑ If a menu entry toggles its state, use a checkbox and leave the menu entry name the same for the different states ("Italics"). If this won't be clear, toggle the name so that it indicates what action will be taken if the menu entry is selected ("Show Grid," "Hide Grid").
- ❑ Capitalize the menu entry using the same rules as capitalizing book titles.
- ❑ Display an ellipsis (...) after menu entries that bring up a dialog that requests more information from the user. Don't use ellipses if the dialog simply brings up information that the user requested (for example, a Help dialog).

When ordering menus and menu entries . . .

- ❑ Place the standard menus in the standard order (File, Selected, Edit, View, Tools, Options, Help), even if you have renamed any of these menus. Place any new menus between the View and Tools menus.
- ❑ Place standard menu entries in the standard order. “Close” and “Exit” are always at the end of the leftmost menu whether or not this menu is named File.
- ❑ Group menu entries logically. If a new menu entry is related to one of the standard menu entries, place it near that standard menu entry.
- ❑ Place items in the menu first according to the order they will be used, and second according to their frequency of use (with more frequently used items closer to the top of the menu).
- ❑ Alphabetize entries that can be determined only when the user launches the application. If this alphabetized group appears in a menu that contains other entries, place the group at the end of a menu and use a separator between it and the preceding entries.
- ❑ Use radio buttons for mutually exclusive menu entries, and checkboxes for a group of related menu entries, any number of which can be selected at any one time.
- ❑ Use separators when necessary to group items—for example, to set off a group of related entries that use radio buttons or checkboxes.
- ❑ Limit the use of cascading menus. Never use more than one level of cascading menus.

When selecting keyboard accelerators for menu entries . . .

- ❑ Use standard keyboard accelerators for standard menu entries; don’t use any of the standard accelerators for your own entries, even if you’re not using those standard entries.
- ❑ Provide keyboard accelerators for the most frequently used menu entries. Don’t provide accelerators for all menu entries.
- ❑ Use the key combination <Ctrl>*character*. Don’t use the key combination <Alt>*character* because this conflicts with mnemonics.
- ❑ For pairs of menu entries where one entry reverses the results of the other entry (“Undo” and “Redo”), use <Ctrl>*character* for the most

frequently used entry and <Shift><Ctrl>*character* for the other entry where *character* is the same for both accelerators.

- Display all characters in keyboard accelerators as uppercase (for example, display <Ctrl>s as “Ctrl+S”). For keyboard accelerators that involve uppercase characters, show the <Shift> key as part of the keyboard accelerator (for example, display <Ctrl>S as “Shift+Ctrl+S”).

When deciding when to disable menu entries . . .

- If selecting the menu entry in the current context would give the user an error message, show the menu entry as disabled (dimmed).
- Avoid using dynamic entries. Rather than removing an entry when it’s temporarily unavailable, include it and disable it as appropriate.

Popup Menus

Use popup menus to provide a quick way for users to access the most commonly used functions in the associated work are. For example, you might provide a popup menu containing “Cut,” “Copy,” “Paste,” and “Delete” in a text application. Popup menus should never be the sole access to functions because these menus are hidden. Instead, popup entries typically represent the most commonly used actions from the application window’s pull-down menus. (See “Menu Traversal and Activation” for a description of how users interact with popup menus.)

At most, your application should provide a different popup menu for each main area of the window (that is, for each main field or pane). Note that this differs from the *OSF/Motif Style Guide*, which allows the availability and content of popup menus to vary depending on the element under the pointer or the selection state of the element. Instead of following this model, you should have one set of entries in the popup menu, and enable and disable each of them as appropriate. With one set of entries, users will become familiar with the popup entries more quickly and won’t be confused when entries are sometimes unavailable (see “Dynamic Menu Entries” for discussion of how dynamic entries can be confusing to users).

What to Put in Popup Menus

A popup menu should contain a title, followed by a separator and the individual menu entries, as shown in Figure 8-13. The title should be the name of the application or, if the application has more than one popup menu, it should describe the purpose of the menu. Since the entries typically repeat entries found in the pull-down menus, they should be displayed similarly: in the same order, and with the same or very similar names as in the pull-down menu. Ellipses and keyboard accelerators should be included if they're included in the corresponding entry in the pull-down menu, but don't show mnemonics in popup menus.

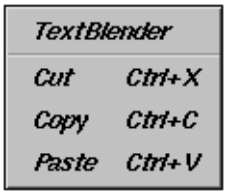


Figure 8-13 Popup Menu

Popup menus generally don't have entries that require radio buttons or checkboxes since these are rarely common enough actions to be included in a popup menu. If you do need to include these kinds of entries in a popup menu, separate them from the rest of the entries with separators and include the radio buttons or checkboxes. See "Using Radio Buttons and Checkboxes in Pull-Down Menus" for more information. Popup menus shouldn't contain cascading menus, nor should they be tear-off menus.

Disabling Popup Menu Entries

As with pull-down menu entries, if one of the entries in a popup menu is unavailable for selection in the current context, that menu entry should be disabled. It shouldn't be removed from the menu.

Popup Menu Guidelines

When choosing when a popup menu should appear . . .

- ❑ At most, provide a different popup menu for each main area (that is main field or main pane) of the window. Don't change the availability of a popup menu based on what graphical element the pointer is over or based on the selection state of any of the graphical elements.

When deciding what to include in a popup menu . . .

- ❑ Include entries for the most commonly used functions from the pull-down menus, and use the same names in the same order as they're displayed in the pull-down menus.
- ❑ Avoid entries that require checkboxes or radio buttons. These are typically not the most commonly used menu functions.
- ❑ Don't make menu entries the sole access to these functions.
- ❑ Don't change the content of the menu based on what graphical element the pointer is over, or based on the selection state or contents of this element. Instead, put all entries in the popup menu for the main area of the window, then enable and disable entries as appropriate.
- ❑ Don't include cascading menus and don't use tear-off menus.

When displaying the contents of the popup menu . . .

- ❑ Include a title that's either the name of the application, or if the application has more than one popup menu, that describes the purpose of the menu.
- ❑ Use only one separator, which goes between the title and the individual menu entries.
- ❑ Show ellipses and keyboard accelerators if these are shown in the corresponding pull-down menu entry, but don't show mnemonics.
- ❑ If selecting the menu entry in the current context would give the user an error message, show the menu entry as disabled (dimmed). Don't remove the menu entry when it's temporarily unavailable.

Chapter 9

Controls

Two types of controls are described in this chapter—those supported in the standard OSF / Motif environment (such as pushbuttons, lists, and scrollbars), and those unique to the Indigo Magic environment (such as enhanced scales, thumbwheels, and dials). These controls can be used in any window of an application. Each description consists of a general description of the control and guidelines for when to use it, how to label it, and how it should behave.

Note that some of the standard controls have been enhanced in the Indigo Magic environment as described in "Enhanced Graphics in the Indigo Magic Look" in Chapter 3. To use these enhanced controls in your application, see "Using the New and Enhanced Widgets" in Chapter 4 of the *Indigo Magic Desktop Integration Guide*. The reference pages in Chapter 9 of the *OSF/Motif Style Guide* provide details on the behavior of the OSF / Motif controls discussed in this chapter. This chapter covers the following controls:

- "Pushbuttons"
- "Option Buttons"
- "Checkboxes"
- "Radio Buttons"
- "Lists"
- "Text Fields"
- "Scrollbars"
- "Indigo Magic Scales"
- "Labels"
- "File Finder"
- "Thumbwheels"
- "Dials"

Pushbuttons

A pushbutton is a button that invokes an operation. Pushbuttons are rectangular and can be labelled with either text or icons, as shown in Figure 9-1. The basic operations for pushbuttons are described in the section "Other Operations," in the reference page for PushButton in the *OSF/Motif Style Guide*, Chapter 9. See "Control Areas in Primary Windows" in Chapter 6 for guidelines on using pushbuttons in control areas and tool palettes, and see "Standard Dialog Actions" in Chapter 10 for guidelines on using pushbuttons in dialogs.

Figure 9-1 Pushbuttons

Pushbutton Guidelines

When using pushbuttons . . .

- In windows with menu bars, use pushbuttons to provide easy access to the most frequently used application-specific functions in the pulldown menus. For primary windows, these pushbuttons appear in the control area of the window.
- In windows without menu bars, use pushbuttons to access help and to close the window.
- Use pushbuttons to create tool palettes, either in support windows or in primary windows.
- Use pushbuttons in the response area of a dialog for the standard actions for that dialog.
- Always have the pushbutton perform the same operation (although the input to that operation may vary based on what data is currently selected). Don't use the same pushbutton to perform different tasks based on some mode of the application.

- Use pushbuttons to perform an action; don't use them merely to set state, such as a parameter value in a dialog box. Use checkboxes, radio buttons, or option menus for this purpose.

When labelling a pushbutton . . .

- Use either a text or graphic label that describes the action associated with the button. With text labels, use an active verb describing the operation the button performs. Each text label should be a single, capitalized word.
- Center the label on the button.
- If the pushbutton opens a dialog to collect more information from the user before the action represented by the pushbutton can be completed, place an ellipsis after the button label. Don't use an ellipsis if the button opens a dialog simply to display some information to the user as an end result of the operation. This use of ellipses is the same as that described for menu entries in the section "Naming Menu Entries in the Pull-Down Menus" in Chapter 8.

When displaying pushbuttons . . .

- If the action associated with a button is temporarily unavailable, disable the button rather than remove it.
- Don't resize pushbuttons when the window is resized.
- Don't use dynamic buttons whose labels change to indicate different functionality depending on the current context. Instead, use multiple buttons and disable buttons that represent functionality that's currently unavailable. With multiple buttons, the functionality is obvious even if some of the buttons aren't currently active. With dynamic buttons, the user has to put the application into the proper context to discover some of the functionality. The one exception to this guideline is the *Cancel* / *Close* button used in Dialogs with the *Apply* button. See "Standard Dialog Actions" in Chapter 10 for information on this special case.

Option Buttons

An option button is a button that displays an option menu. It allows the user to choose one of the options listed in the menu, and its label changes to reflect the currently selected menu entry. Entries in the option menu represent mutually exclusive values of a parameter. Users interact with option menus according to the model described in "Menu Traversal and Activation" in Chapter 8. Figure 9-2 shows an option button and its option menu. Note that the button has a special graphic on it to distinguish it from regular pushbuttons. The basic operations for option buttons are described in the section "Other Operations," in the reference page for OptionButton in the *OSF/Motif Style Guide*, Chapter 9.

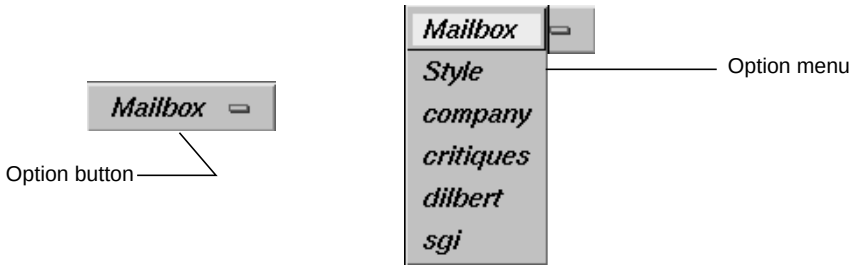


Figure 9-2 Option Button and Option Menu

Option Button Guidelines

When using option buttons . . .

- Use an option button when you want to offer the user about 5-12 mutually exclusive options; use a list for more than 12 choices. If there's enough space, use radio buttons for fewer than 5 choices.
- Don't put radio buttons or checkboxes in an option menu.
- Don't use an option button if the user can select several options at the same time—use a list or a set of checkboxes instead.
- Don't put actions (such as zoom or rotate) in the option menu—use pulldown menus or pushbuttons instead.
- Don't add or delete the choices in the option menu. If the choices must change, use a list.

- ❑ Don’t use cascading menus in the option menu. If there are so many items that they don’t fit conveniently into an option menu, use a scrolling list instead.
- ❑ Don’t use a tear-off entry in an option menu.

When labelling an option button . . .

- ❑ Use the default label for the option button itself, which is the current value of the parameter.
- ❑ Use a second label that describes the parameter that the option button controls. This parameter label should be to the left of the option button and should be followed by a colon (:) and a space (see Figure 9-2). This label is typically a noun.

When labelling the entries in an option menu . . .

- ❑ Use nouns that indicate the possible values of the parameter being set.

When displaying option menus . . .

- ❑ If one of the entries in an option menu is unavailable for selection in the current context, disable the menu entry. Don’t remove the entry from the menu. Note that the user should always be able to display the contents of an option menu even if all of the menu entries are currently disabled.
- ❑ Don’t include a title in option menus.

Checkboxes

A checkbox is a button with two states—on and off. In a group of checkboxes, each can be turned on or off independently. The on state is indicated in the Indigo Magic look by a red check mark, as shown in Figure 9-3. The basic operations for checkboxes are described in the section “Other Operations,” in the reference page for CheckButton in the *OSF/Motif Style Guide*, Chapter 9.

Figure 9-3 Checkboxes

Checkbox Guidelines

When using checkboxes . . .

- Use checkboxes for single attributes or states that can be turned on and off, or for groups of items where multiple items in the group can be selected independently. (Also see "Using Radio Buttons and Checkboxes in Pull-Down Menus" in Chapter 8.)
- Use checkboxes for groups of less than about six items. When dealing with more than a handful of items, use a list that allows multiple elements to be selected at the same time.
- Don't use checkboxes for mutually exclusive options. If only one item in a group of items can be selected at a time, use radio buttons instead.
- Don't use checkboxes for actions; use pushbuttons instead.
- Don't change the choices in the group based on the current context. If you want to offer a dynamic set of choices, use a list.

When labelling checkboxes . . .

- Give each checkbox a label that describes the attribute, state, or option it controls.
- Create a group label for each group of checkboxes, and indent the checkboxes below the label. This group label should be a noun that describes the function of the group.

When displaying checkboxes . . .

- Keep checkboxes updated to reflect the current state of the application and the settings of the current selection (if the settings of the checkboxes

relate to the current selection). For example, if you have a checkbox for turning underlining on and off and the user selects some text, the checkbox should be updated to reflect whether or not the selection is underlined.

- Disable checkboxes representing choices that aren't currently available. Don't remove the checkboxes.

Radio Buttons

A radio button is a button with two states—on and off. Unlike checkboxes, radio buttons are always used in groups. Only one of a group of radio buttons can be turned on at any given time. The on state is indicated in the Indigo Magic look by a blue triangle, as shown in Figure 9-4. The basic operations for radio buttons are described in the section "Other Operations," in the reference page for RadioButton in the *OSF/Motif Style Guide,* Chapter 9.

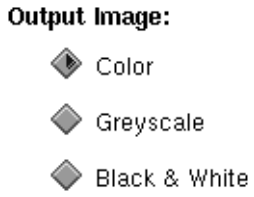


Figure 9-4 Radio Buttons

Radio Button Guidelines

When using radio buttons . . .

- Use radio buttons in groups, never as single buttons. If you need to use a single button that shows an on / off state, use a checkbox instead. (Also see "Using Radio Buttons and Checkboxes in Pull-Down Menus" in Chapter 8.)
- Use radio buttons for mutually exclusive options. If more than one item in the group can be selected at a time, use checkboxes or a list instead.

- Use radio buttons when you want to offer the user fewer than six options. If you have more than six options, or if screen space is extremely limited, use an option button instead. (See the section “Option Buttons” earlier in this chapter.) If you have more than 12 options, you should consider using a list where only a single element can be selected at a time. (See the section “Lists” later in this chapter.)
- Don’t use radio buttons for actions; use pushbuttons instead.
- Don’t change the choices in a group of radio buttons based on the current context. If you want to offer a dynamic set of choices, use a list because users expect the elements of a list to change occasionally, but they don’t expect radio buttons to change.

When labelling radio buttons . . .

- Give each radio button a label that describes the attribute or option it controls.
- Create a group label for each group of radio buttons, and indent the radio buttons below the label. This group label should be a noun that describes the function of the group.

When displaying radio buttons . . .

- Keep radio buttons updated. If the settings of the radio buttons depend on the current selection, they should be updated when the user makes a new selection so that they reflect the settings of the new selection.
- Disable radio buttons representing options that aren’t currently available. Don’t remove the radio buttons.

Lists

A list allows the user to choose from a series of elements. It can allow the user to choose a single element at a time or choose multiple elements at once. Lists should have vertical and horizontal scrollbars when necessary. (See “Scrollbars” later in this chapter.) When allowing users to select elements in the list, you should follow the selection guidelines described in “Selection” in Chapter 7. Figure 9-5 shows a list with vertical and horizontal scrollbars. The basic operation of lists is described in the section “Other Operations,” in the reference page for List in the *OSF/Motif Style Guide*, Chapter 9.

Figure 9-5 List

List Guidelines

When using lists . . .

- Use a list when you want to allow the user to choose a single option from a large list (that is, more than 15 options). If you have fewer than 15 options, use either an option button (best for 5-15 options; see “Option Buttons” earlier in this chapter) or a set of radio buttons (best for 2-5 options; see “Radio Buttons” earlier in this chapter).
- Use a list when you want to allow the user to choose several options from a list of six or more elements. If you have fewer options, use checkboxes (see “Checkboxes” earlier in this chapter).
- If you want to allow the user to choose elements from a dynamic list of options, use a list regardless of the number of options. (Option menus and groups of checkboxes or radio buttons should represent static lists of options.)

When labelling a list . . .

- Label the list with a noun that indicates the function of the elements in the list.
- Place the label directly above and either left-aligned with or slightly to the left of the first element of the list.

When labelling the list entries . . .

- If the elements in the list represent operations to perform, they should be active verbs. Otherwise, they should be nouns.

When displaying lists . . .

- When a window using a list is first opened, the currently selected list elements should be highlighted and the list should be scrolled to display these. If multiple elements are selected, scroll the list so that the first selected one appears at the top of the viewing area. See “Selection” in Chapter 7.
- Allow users to select elements in the list according to the selection guideline discussed in “Selection” in Chapter 7.
- Disable list elements that aren’t currently available.
- Allow the list to autoscroll (the default behavior) if the user is making a selection and the selection goes outside the range of the displayed elements. See “Selection” in Chapter 7.

Text Fields

Text fields can be single-line or multi-line. Single-line text fields don’t have scrollbars, even if all of the text can’t be displayed horizontally in the field. Multi-line text fields should have vertical and horizontal scrollbars when necessary. (See “Scrollbars” later in this chapter.)

Text fields can be either editable or noneditable. Editable and noneditable text fields have different colored backgrounds to indicate to the user whether the information can be changed. These background colors vary depending on what scheme the user has selected (see “Schemes for Colors and Fonts” in Chapter 3 for information on schemes).

Indigo Magic offers an enhanced text field; it allows the application to select a section of text and flag it with an error status. The error selection shows up with a special background color to distinguish it from an ordinary text selection. For example, a debugger might use the error selection to indicate to the user which section of code was causing an error. In addition, the enhanced text field control allows you to specify the following:

- both the foreground and background colors of the selected text (See “Selection” in Chapter 7 for information on selection.)
- the background color for text that’s marked with an error status
- whether to show the text cursor only when the text component currently has keyboard focus (See “Keyboard Focus and Navigation” in Chapter 7 for information on keyboard focus.)

The basic operations for text fields are described in the section “Other Operations” in the reference page for Text in the *OSF/Motif Style Guide*, Chapter 9.

Text Field Guidelines

When using text fields . . .

- ❑ Use single-line, editable text fields to display values of parameters that users can edit.
- ❑ Use single-line, noneditable text fields to display values of parameters that users can’t edit, whenever these values either change over time or might need to be selected by the user. If the value doesn’t change and the user doesn’t need to select it, use a label.
- ❑ Don’t use a text field if you need to display and edit pathnames; use the Indigo Magic File Finder instead.
- ❑ Use text fields for values that change over time; don’t use labels.

When labelling text fields . . .

- ❑ Label each editable or noneditable text field, unless the field represents the bulk of a window and the field’s function is clear.
- ❑ For single-line text fields, place the label to the left of the text field, and follow the label with a colon (:) and a space. The label should be vertically centered within the text field.

When displaying text fields . . .

- ❑ Use the default selection and highlighting discussed in “Selection” in Chapter 7.

- Allow the user to cancel a text edit in progress by pressing <Esc>. That is, once the user has selected text and started to replace it with new text, <Esc> should cancel any changes that the user has made.
- Keep text fields updated. When a window using a text field is first opened, the default or current setting (if either exists) for the text field should be shown.
- Make the text automatically scroll if the user is making a selection and the selection goes outside the range of the displayed elements.
- When an editable text field can't be edited in the current context but the information is still useful to the user, change it to a noneditable text field. If the information isn't useful to the user (that is, the user doesn't need to know the value and won't need to select it), disable the text field.

Scrollbars

A scrollbar "scrolls" the data in a viewing region to change the portion of the data that's visible. Scrollbars can be either horizontal or vertical. ("Enhanced Graphics in the Indigo Magic Look" in Chapter 3 describes enhancements to the scrollbar's appearance.) Figure 9-6 shows a scrollbar.

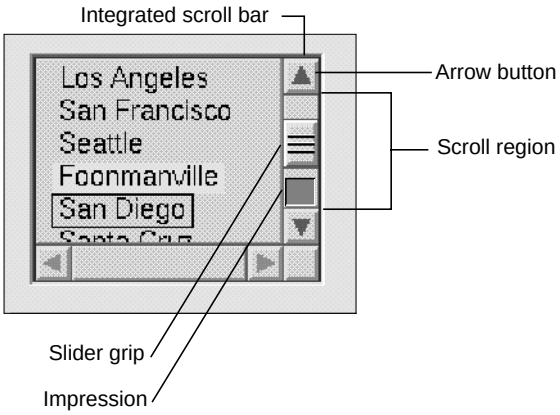


Figure 9-6 Scrollbar

A scrollbar includes the scroll region (shown as a trough), which represents the size of the entire scrollable element with arrow buttons at each end. If there's data that can be scrolled, the scrollbar also includes a slider that indicates the relative position and portion of the data currently being displayed. As the user moves the slider, a temporarily indented impression of the slider indicates the position of the slider before the user began moving it. This indented impression disappears when the user releases the mouse button to complete the scroll action. The basic operations for scrollbars are described in the section "Other Operations," in the reference page for ScrollBar in the *OSF/Motif Style Guide,* Chapter 9.

Scrollbar Guidelines

When using scrollbars . . .

- Use scrollbars to pan an associated view.
- Use scrollbars with components that can be resized such that all of the available information contained in the component can't be displayed at one time. Typical scrollable components include work areas in primary windows, lists, multiple line text fields, and data display areas in primary or support windows.
- Use scrollbars with a list when the number of elements in the list doesn't fit in the viewing region (vertical scrollbar), when the elements are too wide to fit in the viewing region (horizontal scrollbar), or when the window containing the list can be resized such that either of these situations can occur. See "Lists" earlier in this chapter for information.
- Use scrollbars with multi-line text regions when the data can't all be displayed vertically or horizontally or when the window can be resized such that this is true. See "Text Fields" earlier in this chapter for information.
- Don't use scrollbars with single-line text fields. See "Text Fields" earlier in this chapter for information.
- Don't use scrollbars for zooming or for rotation. Use an Indigo Magic thumbwheel instead. See "Thumbwheels" later in this chapter.
- Don't use scrollbars to choose a value in a range; use the Indigo Magic scale instead.

When displaying scrollbars . . .

- Place vertical scrollbars along the right of the element being scrolled, and place horizontal scrollbars along the bottom of the element being scrolled.
- Keep scrollbars updated. When a window using a scrollbar is first opened, the scrollbar should reflect the current area being displayed in the scrolled region.
- Update the data in the scrolled area continuously as the user drags the slider along the scroll region. This gives the feeling of direct, continuous control. Don't wait until the user has released the slider to update the data, because users often use the current view of the data to determine when to stop dragging the slider.
- When a component is being scrolled, don't scroll it beyond the first or last elements. That is, there should be no extra white space before the first element or after the last element. The exception to this rule is scrolling text elements that represent physical pages (for example, in a desktop publishing application).
- Make all components that use scrollbars automatically scroll when the user makes a selection that goes outside of the data currently being displayed. Also, make the component automatically scroll if the user performs an operation that moves the cursor outside of the current view (for example, if the user inserts or deletes text that moves the cursor outside of the current view). In this case, the view should be automatically scrolled so that the cursor is shown when the operation is finished.
- When using the <Page Up>, <Page Down>, <Ctrl>-<Page Up>, or <Ctrl>-<Page Down> key sequences to scroll a page at a time, leave one unit of overlap from the previous page to make it easier for the user to preserve the current context. This unit is application-specific; it might be a line of text, an item in a list, a row of icons, or a specfic number of pixels (for example, in a drawing region). By default, this behavior is automatic for IRIS IM list and text components.
- Remove the slider from the scrollbar when all of the data is currently being displayed. Don't remove the scrollbar or disable it in some other fashion.
- Allow the user to cancel scroll actions by pressing <Esc>. By default, if the user presses the <Esc> key while dragging the slider along the scroll

region, the scroll action is canceled, and both the data and the slider are returned to the position they had before the user initiated the scroll action.

Indigo Magic Scales

Scales can be used either to allow users to change a value in a given range or to display a value in a range. The size of the control shows the size of the range. When the scale is being used to allow users to specify or change a value, the slider indicates the current value in the range and can be dragged by the user. When the scale is being used for display only, there's no slider for the user to control. Figure 9-7 shows the Indigo Magic scale in both modes. The basic operations for scales are described in the section "Other Operations" in the reference page for Scale in the *OSF/Motif Style Guide*, Chapter 9. For specific details on using the Indigo Magic scale in your application, see the **SgScale**(SGI) reference page.

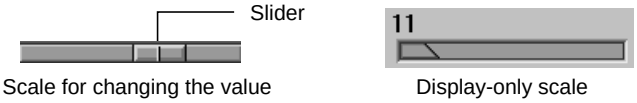


Figure 9-7 Indigo Magic Scale

Indigo Magic Scale Guidelines

When using the Indigo Magic scale . . .

- ❑ Use scales to allow users to change a value in a given range. Use scales in display-only mode to display values that the user can't control. For example, use a display-only scale as a percent-done indicator to show progress in a Working dialog. (See "Working Dialogs" in Chapter 10.)
- ❑ Don't use scales for scrolling.

When labelling a scale . . .

- ❑ Label it with the current value for the scale.
- ❑ If the function of the scale isn't immediately apparent, give the scale an additional label that indicates its purpose.

When displaying scales . . .

- ❑ Keep scales updated. When a window using a scale is first opened, the slider of the scale should show the current setting for the scale control.
- ❑ For sliders where the user can change the value, update the value being manipulated as the user moves the slider. It should give the impression of direct, continuous manipulation. For sliders that also manipulate an object, update the object continuously as well. For sliders that are used only to display values, the slider should be immediately updated to reflect the new value as the value changes.
- ❑ Allow the user to cancel a scale operation by pressing <Esc>. If the user presses the <Esc> key while manipulating the scale, the action should be canceled, and the scale should return to the position it had before the user initiated the action.

Labels

Labels are noneditable text or graphical objects. They aren't selectable.

Label Guidelines

When using labels . . .

- ❑ Use labels for displaying text information that the user won't need to edit or select.
- ❑ Use labels for labeling controls as described under the individual controls in this chapter.
- ❑ Use labels for labeling groups of controls. When used to label a group of controls, the label should be followed by a colon (:) and a space, and should be placed either to the left of the item in the upper left corner of the group or above and slightly to the left of the item in the upper left corner of the group.
- ❑ Use labels for simple instructions when necessary. Before adding instructions to any of your application windows, however, first try to design some alternatives that might make the instructions unnecessary. For example, if these instructions are necessary because the user

interface works in a nonstandard way, redesigning the interface to be more standard is likely to make the instructions unnecessary.

- ❑ Place labels on the background of the window (that is, the part of the window that isn't recessed).

When displaying labels . . .

- ❑ Don't change the text or graphic on a label. If this information will change, consider putting it in a noneditable text field instead; users don't expect label text to change.
- ❑ Disable labels when the controls they represent are disabled. Don't disable group labels.

File Finder

The File Finder is an Indigo Magic control. It allows users to navigate the file hierarchy quickly and to specify directories and files easily, using drag and drop of desktop icons. The File Finder is pictured in Figure 9-8.

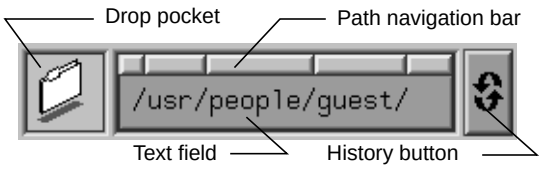


Figure 9-8 The File Finder

The File Finder includes several pieces:

- text field—allows the user to enter the pathname for a file or directory.
- drop pocket—displays the desktop icon representing the current file or directory whose name is displayed in the text field. A user can also drop a desktop icon into this drop pocket and have the text field automatically update with the pathname of the icon.
- path navigation bar—each button represents the directory being displayed below it in the text field. A user can quickly navigate to ancestor directories by clicking on any of these buttons.

- history button—similar to an option menu button; maintains a list of directories that the user already visited while using this control. The user can select any of these previously visited directories to return immediately to that directory.

For specific details on using the file finder in your application, see the **SgFinder**(3X) reference page.

File Finder Guidelines

When using the File Finder . . .

- ❑ Use the File Finder when the user needs to enter the pathname of a directory or file. This allows the user to drag and drop desktop icons to specify the file and to navigate the file hierarchy.
- ❑ When a window using a file finder is first opened, the text field in the file finder should show the default or current value of the pathname, if any. This value should also be placed in the history list under the history button.

Thumbwheels

The thumbwheel is an Indigo Magic control that allows users to specify or change a value, either in a given range (for instance, when zooming) or in an infinite range (for instance, when rotating a 3D object). Users change the current value by direct manipulation of the wheel (that is, by clicking and dragging). The thumbwheel can also include a “home button” that returns the thumbwheel to a default value. Thumbwheels can be oriented either horizontally or vertically. Figure 9-9 shows a thumbwheel. For specific details on using the thumbwheel in your application, see the **SgThumbWheel**(3X) reference page.

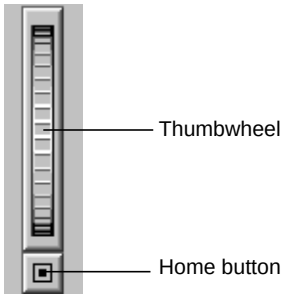


Figure 9-9 Thumbwheel

Thumbwheel Guidelines

When using thumbwheels . . .

- Use thumbwheels to change the values of continuous variables (that is, variables that don't have discrete values). For discrete values, consider a scale or dial instead.
- Use thumbwheels with finite ranges for zooming operations and thumbwheels with infinite range for rotating objects.
- When a thumbwheel is used to change a value that has a clear default, provide a home button. For example, a Directory View window has a thumbwheel that allows the user to set the size of the desktop icons. Pressing the home button on this thumbwheel sets the icons to their default size.
- Use thumbwheels when screen real estate is extremely limited.
- Don't use a thumbwheel for panning; use a scrollbar instead. A scrollbar gives the user much more information about the object being scrolled than a thumbwheel could.

When displaying a thumbwheel . . .

- Update the object or value being manipulated as the user moves the thumbwheel. The thumbwheel should give the impression of direct, continuous manipulation.

Dials

The dial is an Indigo Magic control that allows users to specify or change a value in a given range. Users change the current value by direct manipulation of the dial—by dragging or by clicking on the appropriate tic mark that represents the desired value. The appearance and the behavior of the dial can be modified. For example, the angular range in degrees through which the dial is allowed to rotate and the color of the dial and tic marks can be changed. Figure 9-10 shows two dials with different appearance options. For a complete list of options for dials, and other specific details on using dials in your application, see the **SgDial**(3X) reference page.

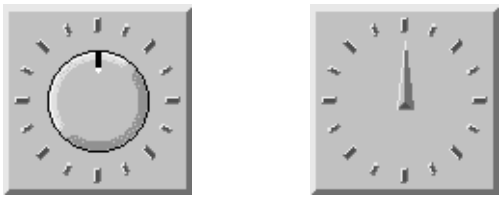


Figure 9-10 Dials

Dial Guidelines

When using dials . . .

- ❑ Use dials as an alternative to scales for setting parameters. Dials are best for numeric parameters where the range of allowable values is small and the values are discrete.

When labelling dials . . .

- ❑ Place a label either directly below or directly above the dial, specifying the parameter that the dial controls.
- ❑ When you have a group of dials, place each dial label in the same position relative to its dial (that is, either all the labels should be below the dials or all the labels should be above the dials).

When displaying dials . . .

- ❑ When a window using a dial is first opened, the dial should show the current setting.

- As a dial is rotated, update the value being manipulated to reflect the new value on the dial. The dial should give the impression of direct, continuous manipulation. Also, if the dial is controlling an object, continuously update the object as the dial is manipulated.

Chapter 10

Dialogs

Dialogs are transient windows that your application uses either to communicate something important to the user (for example, that a pending action could cause some data to be lost), or to obtain a specific piece of information from the user (for example, which file should be opened). Users interact quickly with dialogs and then dismiss them. This chapter covers dialogs in the following sections:

- "Types and Modes of Dialogs" discusses the standard types of dialogs, when to use them, and whether they should be modal or not (that is, whether they should prevent the user from doing anything else until the dialog is dismissed).
- "Designing Dialogs" discusses general dialog design issues—such as what window decorations dialogs should have, how the information in dialogs should be laid out, and what the standard actions are (in the form of push buttons). It also covers specific design and content issues for the various types of dialogs listed in the previous section.
- "Invoking Dialogs" describes the most common situations that require dialogs and which types of dialogs to use in them.

Types and Modes of Dialogs

Dialogs are used to give information to the user or to get information from the user; once they've served their purpose, they go away. Dialogs that give information to the user are instigated by the application; these application-generated dialogs present important messages for the user's immediate attention. Dialogs whose purpose is to get information from the user are displayed as the result of a user action (such as pushing a button or selecting a menu entry). An example of such a user-requested dialog occurs when the user selects "Open..." from the File menu; the application should bring up the Indigo Magic File Selection dialog so that the user can specify which file should be opened.

The *OSF/Motif Style Guide* defines several types of application-generated and user-requested dialogs. The most common of these are listed in Table 10-1, along with a brief description of when each might be used in an application and whether they should be modal or not. Figure 10-1 shows each of the standard OSF / Motif dialogs with the Indigo Magic look, and Figure 10-2 shows the Indigo Magic File Selection dialog. Each of these standard dialogs are discussed in more detail in the rest of this chapter and in their reference pages in Chapter 9 of the *OSF/Motif Style Guide*. Dialog modes are defined and discussed in the next section (“Dialog Modes”).

Table 10-1 Types of Dialogs, Their Modality, and When to Use Them

Type of Dialog	When to Use It	Modality
Prompt	To ask users for specific information.	Modeless or modal
Error	To tell users about an error they’ve made in interacting with your application.	Application-modal
Warning	When there’s an action pending that will cause users to lose data.	Application-modal
Question	To ask users a specific question that they must respond to before continuing to interact with the application. Note that although Warning dialogs can also ask users a question, that question relates to a pending action that’s destructive.	Application-modal
Working	When an operation takes more than 5 seconds to complete. This dialog gives users the chance to cancel or stop the operation. Note that you might have to choose one of several different platforms as your standard for estimating times of operations. Also note that the pointer shape might need to change to the watch if the Working dialog is modal; see “Standard Pointer Shapes and Colors” in Chapter 11.	Modeless or modal
Information	To give users information that’s of immediate importance. This type of dialog should be used sparingly; use a status area in one of your primary application windows for the less important messages (see “Status Areas in Primary Windows” in Chapter 6).	Modeless
File Selection	To allow users to navigate the file hierarchy and choose a file. Note that the Indigo Magic File Selection dialog, which is shown in Figure 10-2, is slightly different from the standard OSF / Motif File Selection dialog.	Modeless

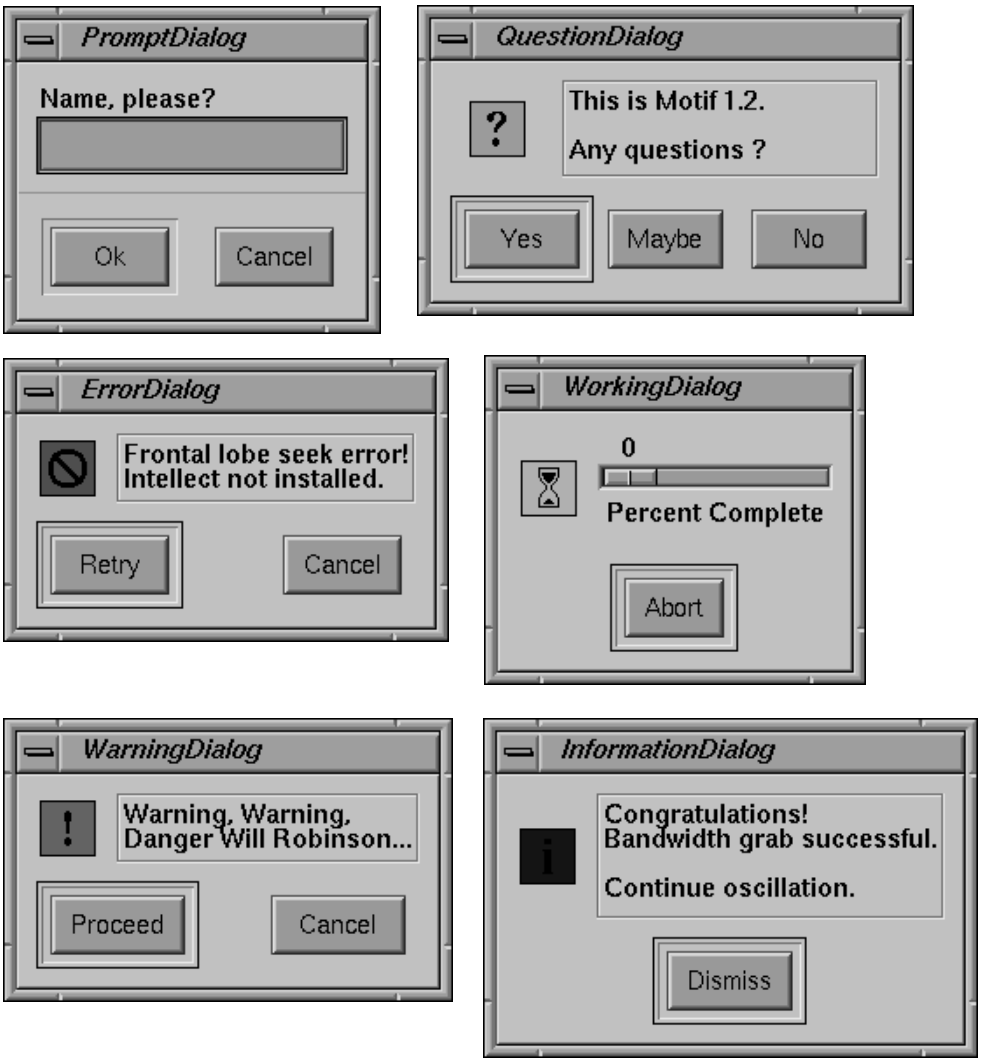


Figure 10-1 Sample Prompt, Error, Warning, Question, Working, and Information Dialogs

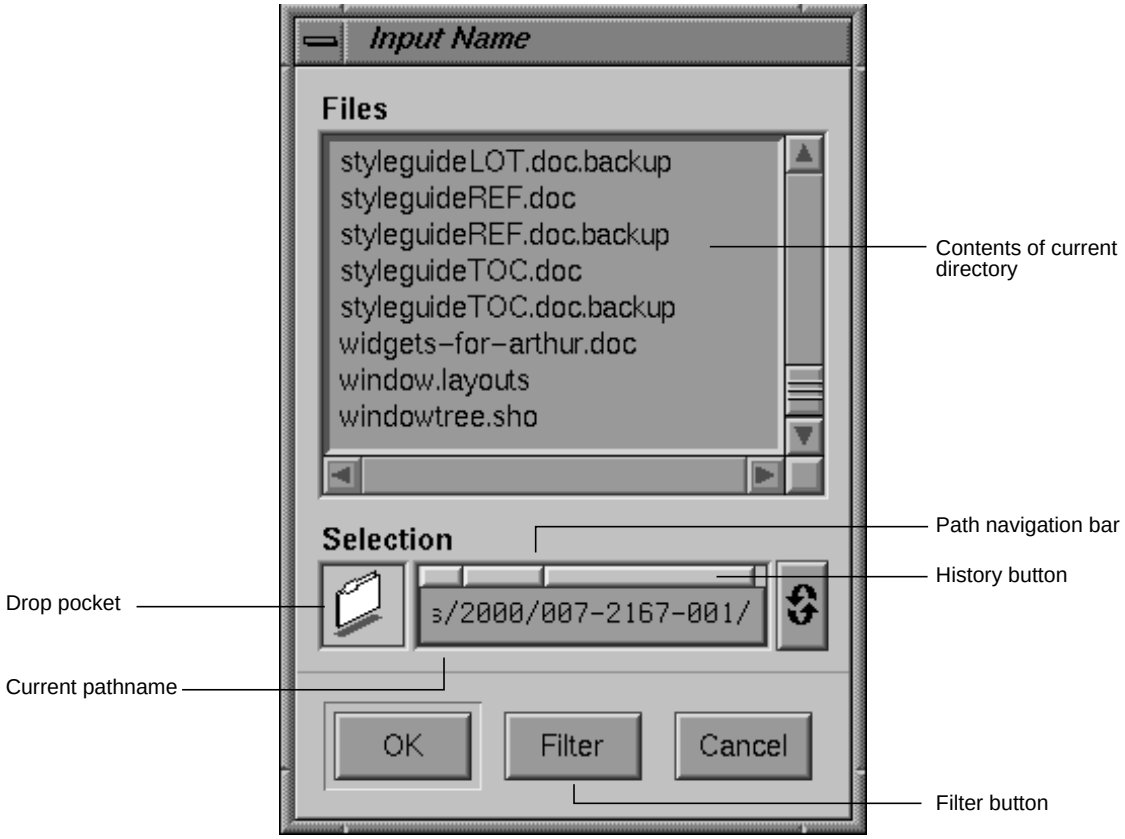


Figure 10-2 The Indigo Magic File Selection Dialog

Both the Indigo Magic and IRIS IM File Selection dialogs provide lists of the contents of the current directory and a text input field; the Indigo Magic list contains both the files and the subdirectories of the current directory, while the IRIS IM list presents these in two separate lists. The Indigo Magic File Selection dialog also allows users to navigate through the file hierarchy using the drop pocket, path navigation bar, and history button. As discussed in detail in “File Finder” in Chapter 9, these components allow users to drop file or directory icons in the drop pocket, traverse to ancestors of the current directory, or return to any directory visited previously. In addition, the Indigo Magic dialog presents a Filter button (rather than the IRIS IM text

input field), which brings up a dialog that allows the user to enter the filter string.

Dialog Modes

As listed in Table 10-1, dialogs can have different modes whereby the application can require the user to respond to the dialog before continuing with other actions in the application. The following are the most commonly used modes defined by OSF/Motif:

Modeless
: Modeless dialogs, such as the Indigo Magic File Selection dialog, don't require the user to respond before continuing. The user can interact with any other window associated with the application and with any other application.

Primary-modal
: Primary-modal dialogs require the user to respond to the dialog before continuing to interact with the dialog's parent window or any other ancestor window. Note that these ancestor windows can't receive mouse or keyboard input until the user has responded to the dialog.

Application-modal
: Application-modal dialogs require the user to respond to the dialog before continuing to interact with the application. Note that none of the application's windows (except the dialog) can receive mouse or keyboard input until the user has responded to the dialog. An example of this type is a dialog that asks the user for a root password.

In addition to these modes, OSF/Motif defines a system-modal dialog that requires the user to interact with the dialog before doing anything else on the system. You shouldn't use system-modal dialogs because your application should never need to restrict users' activities to this degree.

Modal dialogs typically show static information, but modeless dialogs should display dynamically updated information as the current state changes. Otherwise, the dialog will become useless. For example, the Indigo Magic File Selection dialog dynamically updates itself if a user changes the file hierarchy while it's displayed; if it didn't, the user could select a file that no longer exists, for example.

As listed in Table 10-1, File Selection and Information dialogs should be modeless. Error, Warning, and Question dialogs should be application-modal. Working and Prompt dialogs can be modeless or modal, depending on what they are being used for in the application. For example, the desktop displays a Working dialog when you're copying a large directory from a remote system using a directory view, but you can still do other things in the Directory View while the copy is in progress. In other situations, you might not want to allow user input until your application has completed a particular operation. For example, when a user opens a large folder in MediaMail, no other actions can be performed in that window until the folder has been read in completely. See "Working Dialogs" on page 218 for details on the design of Working dialogs.

Guidelines for Using the Various Types and Modes of Dialogs

When choosing the type and mode of a dialog . . .

- Use a Prompt dialog to ask users for specific information. This dialog can be modeless or modal.
- Use an application-modal Error dialog to tell users about an error they've made in interacting with your application.
- Use an application-modal Warning dialog when there's an action pending that will cause users to lose data.
- Use an application-modal Question dialog to ask users a specific question that they must respond to before continuing to interact with the application.
- Use a Working dialog when an operation takes more than 5 seconds to complete. This dialog can be modeless or modal.
- Use a modeless Information dialog to give users information that's of immediate importance. Use this type of dialog sparingly.
- Use the modeless Indigo Magic File Selection dialog to allow users to navigate the file hierarchy and choose a file.
- Don't use system-modal dialogs.
- Use modal dialogs to show static information, and update modeless dialogs dynamically as the current state changes.

Designing Dialogs

This section discusses general guidelines that apply to the design of all dialogs—for example, what window decorations dialogs should have, what size they should be, what information they should display when they first come up, and generally how their information should be laid out. It also describes the standard actions (in the form of pushbuttons) that dialogs should contain. Finally, this section covers guidelines for the content of the specific types of dialogs.

Keep in mind that as discussed in Chapter 6, “Application Windows,” every dialog is associated with a specific primary or support window (its parent). The parent window should be visible and mapped to the screen so that dialogs work properly across Desks, as noted in “Desks” in Chapter 3.

Decorations, Initial State, and Layout of Dialogs

All dialogs should have the window decorations and Window menu entries listed in Table 3-1 and described in “Window Decorations and the Window Menu” in Chapter 3. These decorations and menu entries allow the user to:

- Move a dialog using the title bar. Since 4Dwm doesn’t guarantee that a dialog will be placed in a specific location, a user may need to move the dialog to access information in order to figure out the appropriate response to the dialog. Note that a dialog’s title bar should follow the guidelines discussed in “Rules for Labeling the Title Bar in Windows Other Than Main” in Chapter 3. A proper label allows users to quickly identify the type of dialog and the application to which it belongs.
- Resize a dialog that contains resizable components such as text input fields and scrolling lists. See “Window Decorations and the Window Menu” and “Window Size” in Chapter 3 for specific guidelines on when a dialog should be resizable.

Note that these window decorations and menu entries don’t include operations either for minimizing a dialog (since dialogs can’t be minimized independently of their parent window) or for exiting an application from a dialog.

When a dialog is opened, its size, placement, keyboard focus, and information displayed should follow these guidelines:

- The default size should allow all of the components and information to be displayed in their entirety. Users shouldn't have to resize a dialog to see its contents.
- The dialog should be placed automatically on the screen—either near (but not overlapping) any related information in the parent window, or in the center of the parent window (if the contents of the dialog aren't related to the contents of the parent window). For more information on choosing a screen location, see "Window Placement" in Chapter 3.
- The keyboard focus should be in the field with which the user is most likely to want to interact. For example, if there are text input fields, the focus should probably be in one of those fields. In general, dialogs should follow the keyboard focus and keyboard navigation guidelines discussed in "Keyboard Focus Policy and Navigation Within a Window" in Chapter 7.
- The information being displayed in the dialog should always match the current state of the application. If the dialog is modeless, this information should be dynamically updated, as described in "Dialog Modes" on page 209.

All dialogs you create should include a response area that contains standard dialog actions (pushbuttons) tailored to the type and purpose of the dialog. The next section ("Standard Dialog Actions") discusses what the appropriate buttons are for this area. In addition to the response area, Prompt dialogs should include in their layout an input area that consists of whatever controls are necessary for selecting objects or setting application parameters. Instead of this input area, Error, Warning, Question, Working, and Information dialogs should include a message area, as shown in Figure 10-3. The message area consists of an icon and text region that displays the dialog's message.

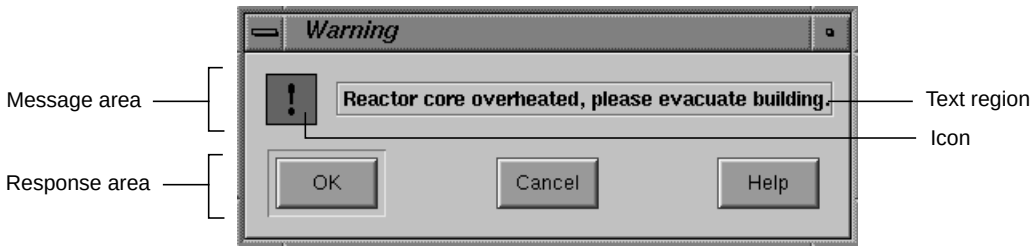


Figure 10-3 Warning Dialog Layout

Note that dialogs shouldn't have menu bars; they're intended for short, quick user input rather than for accessing lots of functionality. Also, dialogs don't contain secondary work areas; if you need additional work areas, use a support window instead. (See "Support Windows" in Chapter 6 for information.) Also note that the pre-designed Indigo Magic File Selection dialog has a somewhat different set of elements and layout than the other types of dialogs.

Standard Dialog Actions

All dialogs include a response area that contains a horizontal row of pushbuttons across the bottom of the dialog. The standard dialog pushbuttons (or actions) are Yes, No, OK, Close, Apply, Retry, Stop, Pause, Resume, Clear, Reset, Cancel, and Help, and they should appear in that order. Your dialogs will typically contain some subset of these buttons and possibly additional ones; the additional buttons should appear after the OK and Apply buttons but before the Cancel and Help buttons.

All of these standard actions except Clear are defined in the reference page for DialogBox in the *OSF/Motif Style Guide*. Clear, which is used in Indigo Magic applications, should clear all of the text input fields in the dialog. Note that this differs from the "Reset" action, which resets all controls in a dialog (not just the text fields) to default
values.

Choosing Specific Actions for Your Dialogs

When choosing which of the standard actions to include in your specific dialog, use the guidelines listed in the *OSF/Motif Style Guide*, Sections 6.2.1.7 and 6.2.4.2, with the following additions and exceptions:

- Most dialogs should have a Help button. If the situation is stated clearly, you might not need a Help button.
- Avoid using both OK and Apply on the same dialog because it often confuses users. The only time you should consider using both is when the number of users who will want to make one set of changes, apply them, and close the dialog is equal to the number of users who will want to make and apply multiple sets of changes before closing the dialog.

- To decide between OK and Apply, determine whether users are more likely to use the dialog to make one set of changes at a time (if so, use OK) or whether they're more likely to want to make and apply changes repeatedly before closing the dialog (in this case, use Apply).

  If you can't decide which of these scenarios best describes your users, use Apply rather than OK. With Apply, users who want to make a single set of changes must press an extra button (Apply, then Close, instead of just OK), which is at most a minor annoyance. On the other hand, using OK by itself forces users who want to make several sets of changes to re-launch the dialog for each set of changes, which can be annoying.

- Any dialog that has an Apply button should also include a Cancel / Close button. When the dialog is opened, the button is labelled "Cancel." After the user applies some irreversible change, the label on the button changes to "Close" to inform users that the action is irreversible. This button doesn't indicate whether or not there are pending changes to be applied.

- Working dialogs should have a Cancel button that allows users to cancel an operation and return the application to the state it was in before the operation began. If you can't return your application to the pre-operation state, you should still allow users to stop the operation at the current point in the processing. It's even better to allow the user a choice of actions—for example, Pause (with the option of later resuming) and Cancel.

- By default, pressing the <Esc> key within a dialog is equivalent to clicking a Cancel button. This is true even if the dialog doesn't have an explicit Cancel button.

Choosing Default Actions

For many dialogs, you should choose one of the actions in the response area to be the default action. By default, the default pushbutton is visually distinguished from the other buttons (for example, the OK button in Figure 10-3), and it's activated when the user presses the <Enter> key while the dialog is the active window. If other buttons in the response area can accept keyboard focus, they become the default button when they have the focus—that is, they're visually distinct from the other buttons, and pressing <Enter> causes them to be activated. When none of these other buttons has

the keyboard focus, the default button status returns to the original default button.

The following bullets describe common default actions for certain types of dialogs:

- The default action for Information dialogs, which typically have buttons only for OK and Help, should be OK.
- The default action for Question, Warning, Error, and any other dialogs that contain buttons but no text fields should be the response that users are most likely to select. For example, a Warning dialog that asks, “Do you really want to do this destructive action?” should have the affirmative response as the default action. Note that as discussed in the next section, “Labeling Dialog Buttons,” each button name should clearly indicate the specific action that will occur if that button is clicked.
- The default action for dialogs that have only one text field and no other controls than the buttons in the response area (such as the File Selection and Prompt dialogs) should be the action that the user is most likely to select after entering a text string.

Dialogs that contain multiple text fields should *not* have a default action since many applications require users to press <Enter> after entering data in a text field. Users tend to press <Enter> regardless of whether they have to, and they expect that action to ensure that their data is entered; they don’t expect that action to invoke the dialog’s default action.

Labeling Dialog Buttons

When labeling dialog buttons, use the OSF / Motif standard names except in the following cases:

- Replace the “Yes” and “No” labels in Warning and Question dialogs with button names that clearly indicate the specific action that will occur if the button is clicked. The buttons replacing Yes and No perform the action and close the dialog. As an example, consider the Warning dialog shown in Figure 10-4.

Figure 10-4 Warning Dialog With Save, Discard, and Cancel Buttons

- Replace the “OK” or “Apply” labels in Prompt or Warning dialogs with button names that clearly indicate the specific action that will occur if the button is clicked (for example, Open, Save, Print). You shouldn’t replace these names when the button is used for more than one purpose—for example, when the file browser is used to specify a name for a new file, the OK button can be used to both name the file and display the contents of a directory.

  You also shouldn’t replace either of these names on those rare instances when OK and Apply are used together in a Prompt dialog.

Content of Specific Types of Dialogs

In addition to the general guidelines discussed in the two previous sections (“Decorations, Initial State, and Layout of Dialogs” and “Standard Dialog Actions”), you should follow the more specific guidelines for the different types of dialogs presented in this section. (The dialog types are defined in Table 10-1.)

Prompt Dialogs

Prompt dialogs use a variety of controls to collect information from the user, including text input fields, a list of all possible choices, radio buttons, checkboxes, and option menus. Try to collect this information in related chunks—that is, avoid collecting unrelated pieces of information in the same dialog, and don’t launch multiple dialog boxes one right after the other to

collect several pieces of information if these pieces are frequently collected at the same time.

Error Dialogs

All Error dialogs should include a description of the error, step-by-step instructions for how to recover from it (or a pointer to information on how to recover from it if the instructions are long), and a pointer to more information about why the error might have occurred. If the error involves a specific entity (for instance, a file, user, or host), name the entity in the error message, as shown in Figure 10-5. Invoke Error dialogs only when they're directly relevant to the user; for example, don't tell the user that the printer is out of paper until the user has a job in the queue.

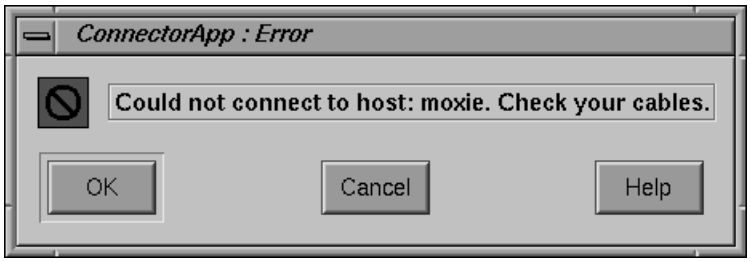


Figure 10-5 Error Dialog With Specific Entity

Warning Dialogs

Warning dialogs should clearly state what data is likely to be lost and why, and they should give the user a chance to cancel the action.

Question Dialogs

Limit your use of Question dialogs to those situations where the user couldn't have provided the information in advance. Also, don't use Question dialogs for questions that relate to a pending destructive action—for these cases, use Warning dialogs instead.

Working Dialogs

For Working dialogs, you should dynamically indicate how much of the operation is complete with the Indigo Magic scale used as a percent-done indicator, as shown in Figure 10-6. (See “Indigo Magic Scales” in Chapter 9 for more information about these indicators.)

Figure 10-6 Working Dialog with Indigo Magic Scale

As described earlier in this chapter in “Choosing Specific Actions for Your Dialogs,” Working dialogs should include at least one way to interrupt the task in progress. If the dialog is modal, you should also switch from the general-purpose pointer to the watch pointer in the dialog’s parent window. If for some reason you’re unable to include any buttons in the Working dialog (such as Cancel, Pause, Resume, or Help), you should switch to the watch pointer in the Working dialog to indicate that user input will be ignored while the operation is in progress. See “Standard Pointer Shapes and Colors” in Chapter 11 for more information about the watch pointer.

Guidelines for Designing Dialogs

When choosing the window decorations, initial state, and layout of dialogs . . .

- ❑ Associate every dialog with a primary or support window (its parent) that’s mapped to the screen.
- ❑ Use the window decorations and Window menu entries listed in Table 3-1 and described in “Window Decorations and the Window

Menu" and "Rules for Labeling the Title Bar in Windows Other Than Main" in Chapter 3.

- ❑ Have the default size large enough to allow all of the components and information to be displayed in their entirety.
- ❑ Place the dialog on the screen either near (but not overlapping) any related information in the parent window, or in the center of the parent window if the contents of the dialog aren't related to the contents of the parent window.
- ❑ Locate the initial keyboard focus in the field with which the user is most likely to want to interact.
- ❑ Be sure the information being displayed in the dialog matches the current state of the application. If the dialog is modeless, update this information dynamically.
- ❑ Include a response area that contains standard dialog actions (pushbuttons) tailored to the type and purpose of the dialog. Also include an input area that consists of whatever controls are necessary for selecting objects or setting application parameters in Prompt dialogs. Include a message area in Error, Warning, Question, Working, and Information dialogs.
- ❑ Don't include secondary work areas; if you need additional work areas, use a support window instead.
- ❑ Don't include menus. If the dialog includes so much functionality that menus are necessary, you should probably use a support window.

When choosing pushbutton actions for dialogs . . .

- ❑ Use a subset of the standard dialog actions (Yes, No, OK, Close, Apply, Retry, Stop, Pause, Resume, Clear, Reset, Cancel, and Help), and have them appear in that order. If you include additional buttons, they should appear after the OK and Apply buttons but before the Cancel and Help buttons.
- ❑ Include a Help button unless the situation is explained thoroughly in the dialog.
- ❑ Avoid using both OK and Apply on the same dialog.
- ❑ To decide between OK and Apply, determine whether users are more likely to use the dialog to make one set of changes at a time (if so, use

OK), or whether they're more likely to want to make and apply changes repeatedly before closing the dialog (in this case, use Apply).

- ❑ Include a Cancel / Close button on any dialog that has an Apply button.
- ❑ Include a Cancel button on Working dialogs and, if possible, a Pause button (with the option of later resuming).

When choosing and creating default actions . . .

- ❑ Whenever appropriate, choose one of the actions to be the default action.
- ❑ Have OK be the default action for Information dialogs (which typically have buttons only for OK and Help).
- ❑ Have the response that users are most likely to select be the default action for Question, Warning, Error, and any other dialogs that contain buttons but no text fields.
- ❑ Have the response that users are most likely to select after entering a text string be the default action for dialogs that have only one text field. Use no other controls than the buttons in the response area (such as the File Selection and Prompt dialogs).
- ❑ Don't have a default action for dialogs that contain multiple text fields.

When labeling dialog buttons . . .

- ❑ Replace the "Yes" and "No" labels in Warning and Question dialogs with button names that clearly indicate the specific action that will occur if the button is clicked.
- ❑ Replace the "OK" or "Apply" labels in Prompt or Warning dialogs with button names that clearly indicate the specific action that will occur if the button is clicked, unless the button is used for more than one purpose, or in the rare instance that "OK" and "Apply" are used together in a Prompt dialog.
- ❑ In all other cases, use the OSF / Motif standard names.

When deciding what content to include in specific types of dialogs . . .

- ❑ Use Prompt dialogs to collect information in related chunks—that is, avoid collecting unrelated pieces of information in the same dialog, and

don't launch multiple dialog boxes sequentially to collect several pieces of information if these pieces are frequently collected at the same time.

- Include a description of the error, step-by-step instructions for how to recover from it, and a pointer to more information in Error dialogs. If the error involves a specific entity (for instance, a file, user, or host), name the entity in the error message.
- Invoke Error dialogs only when they're directly relevant to the user; for example, don't tell the user that the printer is out of paper until the user has a job in the queue.
- State what data is likely to be lost and why, and give the user a chance to cancel the action in Warning dialogs.
- Limit your use of Question dialogs to those situations where the user couldn't have provided the information in advance.
- Don't use Question dialogs for questions that relate to a pending destructive action—for these cases, use Warning dialogs instead.
- Dynamically indicate how much of the operation is complete with the Indigo Magic scale used as a percent-done indicator in Working dialogs.
- Switch from the general-purpose pointer to the watch pointer in the parent window of a modal Working dialog.

Invoking Dialogs

Users expect to encounter dialogs in certain situations. This section describes the most common such situations and gives an example of the dialog your application should provide if users can encounter this situation when interacting with your application. (For more information about the standard menu entries referred to in this section, see "Standard Menus" in Chapter 8.)

Since dialogs are designed to get the user's attention, overuse of them will be distracting to the user. Similarly, application-generated dialogs shouldn't be used to provide general status information; you should use a status area in the associated primary or support window instead (see "Status Areas in Primary Windows" in Chapter 6).

Invoking Dialogs When Manipulating Files

Users expect to be prompted with a dialog whenever they choose a menu entry that includes an ellipsis. These dialogs prompt the user for information that's necessary before the action can be completed, as described in the following common examples.

- "Open..." from the File (or leftmost) menu should display the Indigo Magic File Selection dialog shown in Figure 10-2. The first time this dialog is opened for an application, it should show the current working directory and no specific file. Subsequently, it should come up in the state it was last in when the user dismissed it—that is, it should show the last directory the user traversed to, and the file that the user opened the last time should be selected. (See Table 10-1 and Figure 10-2 earlier in this chapter for more information about the Indigo Magic File Selection dialog.)
- "Save As..." from the File (or leftmost) menu should display the Indigo Magic File Selection dialog. If the file being saved doesn't exist yet, the dialog should show the current working directory and no specific file. If the file exists, the dialog should show that file's directory, and the current name of the file should be selected. If the user uses "Save As..." to change the name of an existing file, your application should create a copy of the existing file with the new name, close the previous file, and open the new file. (See "File Menu" in Chapter 8.)

When users open, close, or save changes to files, they should be prompted with the Warning dialog shown in Figure 10-4 whenever these actions will cause data to be lost. The standard situations in which this can arise include the following (see "Application Models" in Chapter 6 for a discussion of the single- and multiple-document application models described in the following paragraphs):

- In an application that allows only one document to be open at a time, when the user chooses to open another (new or existing) document and there are unsaved changes in the currently opened document. The user can open another document by selecting "New" from the File menu, by selecting "Open..." from the File menu and using the File Selection dialog, or by choosing a file from the "Reopen" cascading menu in the File menu.
- When the user chooses "Close" from the File menu in a co-primary window, and the co-primary window contains data that will be lost if

the window is closed. This situation is especially common in applications that support multiple open documents because for these applications, closing a co-primary window is equivalent to closing a file.

- When the user chooses “Exit” from the File menu, and at least one open co-primary window contains data that will be lost if the application is exited. For applications that support multiple open documents, the user should be prompted with a separate dialog box for each file that’s currently open and has unsaved changes.

Other common file-related situations that require dialogs include:

- When the user chooses “Save” from the File menu, and the current file is untitled. In this situation, the user should be prompted with the Indigo Magic File Selection dialog, as described above for “Save As....”
- When the user is interacting with the File Selection dialog and chooses a filename that already exists. In this situation, the user should be prompted with the Warning dialog shown in Figure 10-7.

Figure 10-7 Warning Dialog for Overwriting a File

- When the user chooses “Revert” from the File menu, and the file currently has unsaved changes. In this situation, the user should be prompted with the Warning dialog shown in Figure 10-7.

Figure 10-8 Warning Dialog for Reverting to Previous Version

Other Situations for Invoking Dialogs

When the user chooses “Product Information” from the Help menu, you should display an Information dialog like the one shown in Figure 10-9. (See “Product Information” in Chapter 4 for some suggestions about what to include in Product Information dialogs.)

Figure 10-9 Product Information Dialog

When users initiate an operation that takes more than five seconds to complete, your application should display a Working dialog. In this situation, the user should be prompted with the Working dialog shown in Figure 10-6. Note that you might have to choose one of several different platforms as your standard for estimating times of operations. See “Working

Dialogs" earlier in this chapter for more information about the content of these dialogs.

When the user chooses "Paste" from the Edit menu and there's nothing currently on the clipboard to be pasted, bring up an Information dialog that says "The clipboard is empty" and that contains the single button "OK." (See "Edit Menu" in Chapter 8.)

Guidelines for Invoking Dialogs

When determining when to display a dialog and which dialog to display . . .

- ❑ Limit the use of dialogs to those cases when they're absolutely necessary. Don't use dialogs to provide general status information—use a status area in the associated primary or support window instead.
- ❑ Invoke a dialog whenever users choose a menu entry that includes an ellipsis.
- ❑ Display the Indigo Magic File Selection dialog when the user chooses "Open..." from the File menu. The first time this dialog is opened, it should show the current working directory and no specific file. Subsequently, it should come up in the state it was last in when the user dismissed it.
- ❑ Display the Indigo Magic File Selection dialog when the user chooses "Save As..." from the File menu. If the file being saved doesn't exist yet, the dialog should show the current working directory and no specific file. If the file exists, the dialog should show that file's directory, and the current name of the file should be selected.
- ❑ When users open, close, or save changes to files, prompt them with a Warning dialog whenever these actions will cause data to be lost:
    - In an application that allows only one document to be open at a time, when the user chooses to open another (new or existing) document and there are unsaved changes in the currently opened document. (For example, the user chooses "New," "Open," or "Reopen" from the File menu.)
    - When the user chooses "Close" from the File menu in a co-primary window, and the co-primary window contains data that will be lost if the window is closed.

- When the user chooses “Exit” from the File menu, and at least one open co-primary window contains data that will be lost if the application is exited. For applications that support multiple open documents, prompt the user with a separate dialog box for each file that’s currently open and has unsaved changes.

❑ Prompt users with the File Selection dialog when they choose Save from the File menu and the current file is untitled. The behavior is the same as the “Save As...” entry in this situation.

❑ Prompt users with a Warning dialog when they’re interacting with the File Selection dialog and they choose a filename that already exists.

❑ Prompt users with a Warning dialog when they choose “Revert” from the File menu and the file currently has unsaved changes.

❑ Display an Information dialog when a user chooses the “Product Information” entry from the Help menu.

❑ Display the Working dialog when users initiate an operation that takes more than five seconds to complete. Note that you might have to choose one of several different platforms as your standard for estimating times of operations.

Chapter 11

User Feedback

Your application should supply feedback to users so that they know it's working and what it's doing. This chapter covers the following topics:

- "Types of Feedback" briefly describes various types of feedback users expect your application to provide; it also tells you where to look in this guide for more information about each type of feedback.
- "Pointer Shapes and Colors" discusses when to use each of the standard pointer shapes and provides guidelines for designing your own pointer shapes.

Types of Feedback

Your application should provide feedback to users using the techniques described in this section. Note that most of these techniques are covered in other chapters of this guide, as indicated; these other chapters also include the explicit checklist guidelines you should follow, so they're not repeated here.

Provide Graphic Feedback

Appropriate desktop icons for your application's executable and data files allow users to readily identify your application, files that were created using your application, and the current state of the application (that is, running or not running). You should design executable and data file icons to provide this sort of graphic feedback, as discussed in Chapter 2, "Icons."

The Indigo Magic look includes graphic modifications that were made to standard IRIS IM in order to improve the level of user feedback. For instance, locate highlight visually indicates which components are live functional objects and which are passive graphics. In addition, scrollbars were redesigned to keep track of their initial positions, and radio buttons and

checkboxes show their state more emphatically. Your application should use the Indigo Magic look, which is discussed in "The Indigo Magic Look: Graphic Features and Schemes" in Chapter 3.

Users expect the pointer to change shape to reflect the current state of the window—for example, when the application is busy processing and can't accept keyboard or mouse input, the pointer should change to a watch. Guidelines for pointer shapes are discussed later in this chapter, in "Pointer Shapes and Colors."

As users select data in a window, that data should be highlighted to show what's included in the current selection. The data should remain highlighted even when the window isn't the currently active window. See "Highlighting a Selection" in Chapter 7 for more information about highlighting selections.

Keep Information Up to Date

As users set particular values in components such as radio buttons, check boxes, lists, and option menus, your application should always indicate the current values so that the user knows what they are. For example, the Language control panel highlights the current values for "Locations" and "Keyboards" in the two corresponding lists. Radio buttons, checkboxes, lists, and option menus are discussed in more detail in Chapter 9, "Controls."

Even if users can change values or data without using an explicitly provided component, your application should still endeavor to display the current information. For instance, users can change the file hierarchy using the shell; if your application displays information affected by such a change (such as a directory view), the display should update dynamically as the user makes the change. (See "File Alteration Monitor (FAM)" in Chapter 4 for more information about using the Indigo Magic FAM service to monitor changes to the file system.) If it's impossible or if it would have a significantly adverse effect on your application's performance to make the display dynamic, choose a design that looks static. For example, you might use label text, which looks like it's a permanent part of the background, rather than text fields, which look like they should be updated constantly. The desktop uses this strategy for the "business cards" that are displayed when a user double-clicks a person icon.

When component settings apply to a specific object, the displayed components should reflect the values for the currently selected object (if there is one). For example, if you select some text in an IRIS Showcase file, the “Font Family,” “Font Size,” Bold / Italic / Underline, and color options in the Master Gizmo are updated to display the characteristics of the selected text.

Provide Messages to the User

In addition to providing immediate graphic feedback through your application’s icons, components, and pointers, you should give users textual messages that describe your application’s status. Keep in mind that by default the window manager for the Indigo Magic Desktop, *4Dwm*, sends *stdout* and *stderr* messages to the Console window, which users typically keep minimized. (Users can choose to have *stderr* messages appear in a dialog box by using the Desktop Settings control panel available from the Desktop->Customize cacading menu in the Toolchest, and they can of course un-minimize the Console window.) Because of these default settings, you can’t be sure that users will notice messages sent to *stdout* and *stderr,* so you should use dialogs or status areas in your application instead.

In particular, you should use dialogs to provide warning messages, error messages, and work-in-progress feedback, as discussed in Chapter 10, “Dialogs.” You should also define an area on primary windows for status messages in the cases discussed in “Status Areas in Primary Windows” in Chapter 6. Finally, you should change the label (or possibly the image) of your application’s minimized window when appropriate to provide feedback; “Minimized Windows” in Chapter 3 discusses when to use this technique.

General User Feedback Guidelines

- Provide graphic feedback with appropriate desktop icon designs, by using the Indigo Magic look, by changing pointer shapes appropriately, and by highlighting selected text.
- Be sure your application displays up-to-date information—in controls and components (display the settings that correspond to the currently selected object or the current values), and in information displays (such

as directory views). If the information being displayed can't be dynamically updated, choose a design that looks static.

- Provide textual message to the user through dialogs, through status areas on your primary windows, and by changing the label of your minimized window when appropriate.

Pointer Shapes and Colors

Your application should use different pointer shapes to indicate when it's busy or in a particular mode, or when one of its windows isn't accepting input. This section discusses when to use the standard pointer shapes and when and how to design new pointer shapes.

Standard Pointer Shapes and Colors

The *OSF/Motif Style Guide* defines several standard pointer shapes. Your application should use these standard shapes for the purposes described in Table 11-1; the table also notes any additions and exceptions to the OSF / Motif policies for using these pointers. If your application requires functionality beyond what's described below, you should design your own new pointers, as described in "Designing New Pointer Shapes," rather than extend the functionality of these standard ones.

Table 11-1 Standard Pointer Shapes and Colors

Pointer	Name	Purpose	Additions and Exceptions to OSF/Motif Style
	upper-left arrow	General-purpose pointer, used for selecting data, setting the values of controls, and initiating actions (for example, by clicking on a button).	In the Indigo Magic environment, this pointer should be red with a white outline (rather than black with a white outline) so that it's easier to see against most typical user-customized background colors and patterns.
	upper-right arrow	Indicates that a menu is being displayed and that the application is waiting for the user to select a menu item or remove the menu.	This is the default pointer when a menu is pulled down from a menu bar, popped up from the right mouse button, or displayed from an option menu button. (See Chapter 8, "Menus," for details on the various types of menus.)
	watch	Indicates that an operation is in progress in the area, and that all mouse-button and keyboard events are ignored in the area.	Use this pointer instead of the hourglass because the watch is a more universally recognized symbol for time. Also, use this pointer if you estimate that the operation generally takes more than 3 seconds. (Note that you might have to choose one of several different platforms as your standard for estimating times of operations.) For less than 3 seconds, maintain the current pointer shape to avoid distracting users. For more than 5 seconds, use a work-in-progress dialog in addition to the watch pointer. (See Chapter 10, "Dialogs.")
	I-beam pointer	Indicates that your application is in text-editing mode. (Note that this I-beam pointer is different from the I-beam text insertion cursor.)	OSF / Motif allows this pointer to be used for indicating that the pointer is over an editable text component in a window that uses implicit focus. Since you should use explicit focus when moving within a window, you don't need the I-beam pointer for this purpose. However, you can use it to indicate that your application is in text-editing mode; this might be useful if your application can edit both text and graphics, for example.
	question mark	Indicates that the user is in context-sensitive help mode and needs to click on an area of the screen to specify the exact help information requested.	none
	cross hair	Used to make fine position selections; for example, to indicate a pixel to fill in a drawing program, or to select the endpoint of a line.	none
	resize	Indicates positions when resizing an area.	none

Table 11-1 (continued) Standard Pointer Shapes and Colors

Pointer	Name	Purpose	Additions and Exceptions to OSF/Motif Style
	4-directional arrow	Indicates that either a move operation or a resize operation (before the resize direction has been determined) is in progress.	none

The *OSF/Motif Style Guide* defines a few pointers that you shouldn't need to use:

- Hourglass—Use the watch instead of the hourglass because the watch is a more universally recognized symbol for time.
- X—Reserved for use by the window manager.

OSF / Motif also defines a caution pointer to be used for indicating that all mouse and keyboard events are ignored in the area until the user performs an expected action in a primary modal or application modal dialog. You can use this pointer in your application if you want; note that many Indigo Magic applications don't use it because at this time there's no automatic support for it in IRIS IM. (See "Dialog Modes" in Chapter 10 for more information on primary and application modal dialogs.)

Designing New Pointer Shapes

You might find it necessary to design new pointer shapes that represent functionality specific to your application, particularly if your application has modes. In these cases, the pointer shape can be used to indicate the current mode. For example, a paint program typically has different tools or modes that the user can select; the pointer shape might resemble a specific brush style, spray paint can, eraser, or I-beam pointer, depending on the tool selected. When you design new pointer shapes, follow the guidelines listed in the reference page on Pointer Shapes in the *OSF/Motif Style Guide*: create a pointer shape that gives a hint about its purpose and is easy to see, avoid shapes that create visual clutter, and make its hotspot easy to locate. (The hotspot identifies where mouse actions occur.)

Pointer Shapes and Colors Guidelines

When deciding which pointers to use in your application . . .

- ❑ Use the standard pointers when possible.
- ❑ Use the upper-left pointing arrow as a general-purpose pointer; this pointer should be red with a white outline.
- ❑ Use the upper-right pointing arrow when a menu is pulled down from a menu bar, popped up from the right mouse button, or displayed from an option menu button.
- ❑ Use the watch pointer for operations that take more than 3 seconds. (For less than 3 seconds, maintain the current pointer; for more than 5 seconds, also use a work-in-progress dialog.)
- ❑ Use the I-beam pointer to indicate that your application is in a text-editing mode, but don't use it to indicate implicit focus over a text object within a window.
- ❑ Use the question mark to indicate that the user is in context-sensitive help mode.
- ❑ Use the sighting pointer (crosshair) to make fine position selections.
- ❑ Use resize pointers to indicate positions when resizing an area.
- ❑ Use the 4-directional arrow to indicate that either a move operation or a resize operation is in progress.
- ❑ Don't use the hourglass pointer; use the watch pointer instead.
- ❑ Don't use the X pointer (it's reserved for the window manager).
- ❑ Don't assign new functionality to the standard pointer shapes; instead, design your own new pointer shape.

When designing new pointer shapes . . .

- ❑ Create a pointer shape that gives a hint about its purpose.
- ❑ Make the shape easy to see.
- ❑ Make the hotspot easy to locate.
- ❑ Avoid shapes that would create visual clutter.

Appendix A

Summary of Guidelines

This appendix contains an all-inclusive list of the guidelines for designing applications for the Indigo Magic Desktop Environment. Its purpose is to serve as a master checklist and summary; to understand the rationale for these guidelines, refer back to the chapters of this manual where they are discussed in detail. The guidelines are grouped as follows:

- "Icon Appearance Design Guidelines" on page 237
- "Icon Behavior Guidelines" on page 238
- "Application Icon Accessibility Guidelines" on page 239
- "Indigo Magic Look Guidelines" on page 239
- "Application Window Characteristic Guidelines" on page 240
- "Guidelines for Keyboard Focus Across Windows" on page 243
- "Minimized Window Guidelines" on page 243
- "Desks Guidelines" on page 245
- "Session Management Guidelines" on page 245
- "Software Installation Guideline" on page 245
- "Guidelines for Designing Online Help" on page 246
- "Guidelines for Creating SGIHelp Content" on page 248
- "Desktop Variables Guidelines" on page 250
- "File Monitoring Guideline" on page 251
- "Data Exchange Guidelines" on page 251
- "Application Model Guidelines" on page 252
- "Primary Window Guidelines" on page 253
- "Support Window Guidelines" on page 254
- "Pointer Behavior Guidelines" on page 255

- “Keyboard Focus and Navigation Guidelines” on page 255
- “Selection Guidelines” on page 256
- “Drag and Drop Guidelines” on page 257
- “Menu Traversal and Activation Guidelines” on page 257
- “Pull-Down Menu Guidelines” on page 258
- “Popup Menu Guidelines” on page 261
- “Pushbutton Guidelines” on page 262
- “Option Button Guidelines” on page 263
- “Checkbox Guidelines” on page 264
- “Radio Button Guidelines” on page 265
- “List Guidelines” on page 266
- “Text Field Guidelines” on page 268
- “Scrollbar Guidelines” on page 269
- “Indigo Magic Scale Guidelines” on page 270
- “Label Guidelines” on page 271
- “File Finder Guidelines” on page 272
- “Thumbwheel Guidelines” on page 272
- “Dial Guidelines” on page 273
- “Guidelines for Using the Various Types and Modes of Dialogs” on page 274
- “Guidelines for Designing Dialogs” on page 274
- “Guidelines for Invoking Dialogs” on page 277
- “General User Feedback Guidelines” on page 279
- “Pointer Shapes and Colors Guidelines” on page 279

Icon Appearance Design Guidelines

For any icon you create . . .

- ❑ Provide a meaningful, distinctive symbol that gives your product an identity and that allows users to readily identify your application and its corresponding custom data files, if any.
- ❑ Keep your design fairly simple because desktop icons can be displayed at very small sizes.
- ❑ Make sure that your icon can be identified across the range of viewing sizes.
- ❑ Color most of your icon using the *icon color* predefined by IconSmith so that your icon's state is easy to detect.
- ❑ Use two or more areas of accent colors to help your icon stand out against user-customized background colors.
- ❑ Avoid small areas of color (2-4 pixels) because they're difficult to see against patterned backgrounds.
- ❑ Include an outline around your custom symbol, and use the *outline color* supplied by IconSmith.
- ❑ Avoid or use sparingly intense, strongly saturated colors and the specific colors used by the Indigo Magic Desktop—bright yellow, dim yellow, royal blue, light gray-green, cadet blue, and Navajo white. These colors make it difficult to distinguish between certain icon states and to find your icon against the background colors of many desktop tools.
- ❑ Orient your icon so that it displays a three-quarter view that faces the lower right corner of the screen.

When designing an application icon . . .

- ❑ Include the magic carpet, the generic executable symbol, with your application's symbol.
- ❑ Indicate the state of the application (not running vs. running) by providing two different application symbols and by moving the magic carpet from a horizontal (not running) to vertical (running) position. Remember that your application symbols should resemble a progressive animation when viewed in succession.

- ❑ Make sure that your application symbols do not obscure the magic carpet in either its horizontal or vertical position.

If your application saves data in a custom file format . . .

- ❑ Design a unique data file format symbol that is readily associated with your application icon design and also indicates how the data is used.
- ❑ If your application is document-based, include the generic data file symbol (stack of papers) in your design.
- ❑ If your data file icon does not use the generic data file symbol, create an appropriately shaped shadow for your file icon and use the predefined *shadow color* supplied by IconSmith.

Icon Behavior Guidelines

When creating an FTR to define your application icon's behavior . . .

- ❑ Provide a CMD OPEN rule that launches the application. This allows the user to open your application either by selecting your application icon and then choosing "Open" from the corresponding Selected menu, or by double-clicking your application icon.
- ❑ Provide a CMD ALTOPEN rule that opens the Launch dialog box shown in Figure 2-12 with the path to your executable displayed in the text field of this window. This allows the user to open the Launch dialog box by double-clicking your application icon while holding down the **`Alt`** key.
- ❑ Provide a CMD DROP rule that launches your application with the file specified by the dropped icon. If your application doesn't understand the type of file represented by the icon dropped on it, your application should provide an appropriate error message to the user rather than launching. This allows the user to launch your application with a specific file by dragging the file icon and dropping it on your application icon.

When creating an FTR for your file icon . . .

- ❑ Provide a CMD OPEN rule that launches your application and automatically opens the file represented by the file icon. This allows the user to open a file created by your application either by selecting the file

icon and then choosing “Open” from the corresponding Selected menu, or by double-clicking the file icon.

- ❑ Provide a CMD PRINT rule that sends the file represented by the file icon to the specified printer. This allows the user to send your application’s data files to the default printer by selecting the file icon and then choosing “Print” from the corresponding Selected menu. It also allows the user to send your application’s data files to any printer by dragging the file icon and dropping it on an icon that represents the specific printer.

Application Icon Accessibility Guidelines

When making your application icons accessible to users . . .

- ❑ Place your application icon on the Applications page in the Icon Catalog. If you produce a suite of software applications, consider creating your own page.
- ❑ In your documentation, refer users to the appropriate page in the Icon Catalog after they’ve installed your application.
- ❑ When naming your executable, use the product name or choose a name that’s strongly associated with the product.
- ❑ When naming your executable, use only lowercase letters. Don’t use numbers, spaces, or special characters such as underlines or periods.
- ❑ Make sure that a link to your executable (preferred method) or the executable itself resides in a directory in the user’s default search path. Ideally, place a link to your executable in the */usr/sbin* directory. This helps ensure that users can quickly find your application icon using the Find an Icon tool.

Indigo Magic Look Guidelines

When designing the look for your application . . .

- ❑ Use the Indigo Magic look rather than the standard IRIS IM look.
- ❑ Use the pre-packaged color and font schemes supplied by Silicon Graphics rather than designing your own colors and fonts.

Application Window Characteristic Guidelines

In general, when deciding on the characteristics for your application windows . . .

- ❑ Determine which category (main, co-primary, support, or dialog) each application window belongs to and assign characteristics appropriately.

When setting up your window decorations . . .

- ❑ Include a Window menu button for all windows.
- ❑ Include resize handles only if the window contains resizable components such as work areas, scrolling lists, and text input fields.
- ❑ Include a *Minimize* button for all primary windows. Do not include this button on support windows or dialogs.
- ❑ Include a *Maximize* button only if the window contains resizable components.

(To see the above window decoration requirements arranged according to window type, see Table 3-1.)

When designing the Window menus for your application windows . . .

- ❑ Include "Restore Alt+F5" for all primary windows. Include it for support windows and dialogs only if the menu contains a "Maximize" entry.
- ❑ Include "Move Alt+F7" for all windows.
- ❑ Include "Size Alt+F8" and resize handles for windows that contain resizable components such as works areas, scrolling lists, and text input fields.
- ❑ Include "Minimize Alt+F9" and the *Minimize* button for all primary windows. Do not include the Minimize entry for support windows or dialogs.
- ❑ Include "Maximize Alt+F10" for windows that are resizable, that is, they have a "Size Alt+F8" entry.
- ❑ Include "Raise Alt+F2" for all windows.
- ❑ Include "Lower Alt+F3" for all windows.

- Include "Close Alt+F4" for all windows except the main primary window.
- Include "Exit Alt+F12" for the main primary window. Include "Exit Alt+F12" for those co-primary windows from which users can quit the application. "Exit" always has the same behavior, that is, it quits the application, no matter how it's activated. Don't include "Exit" for support windows or dialogs.

(To see the above Window menu requirements arranged according to window type, see Table 3-1.)

- Always use the default behaviors for the Window menu entries except for "Exit." Don't add functionality to these commands. When users choose "Exit," your application must perform any necessary clean up, such as prompting the user to save unsaved changes before quitting.
- Don't add application-specific entries to this menu. Users don't expect application-specific entries in the Window menu.
- Don't add a title to the Window menu.
- Don't use the keyboard accelerators <Alt-F2>, <Alt-F3>, <Alt-F4>, <Alt-F5>, <Alt-F7>, <Alt-F8>, <Alt-F9>, <Alt-F10>, or <Alt-F12> for other functions in your application. They are reserved for the *4Dwm* Window menu entries.

When specifying the label in the title bar . . .

- For all categories of windows, limit the length of each title bar label such that the entire label displays when the window is viewed at its default size.
- Don't include application-critical information or general status information in the title bar such as the current page number or whether a file is in view-only mode.
- For main windows, first determine if your application uses document files. If it is not document-based, use the application name only. If it is document-based, use the application name followed by a colon and the filename (or Untitled if new file) in the format *AppName : filename* and update the label whenever the filename changes. Don't use the full pathname unless that information is required for users to distinguish one window from another. If your application is displaying remotely,

add the host name followed by a colon at the beginning of the title bar label in the format *Host : AppName ...* .

- For co-primary windows used in multiple document models, use the format *AppName : Filename* (or *AppName :* Untitled if a new file). For co-primary windows used in the "single document, multiple primaries" model, use the format *AppName : Function*. Make sure that the function matches the menu entry or the label on the button that invokes it.
- For support windows, use the application name and function in the format: *AppName : Function*. Make sure that the function closely matches the menu entry or the label on the button that invokes it.
- For dialog windows, use the application name, followed by the type of dialog in the format: *AppName : DialogType*, where *DialogType* is "Prompt," "Error," "Warning," "Question," "Information," "Working," or "File Selection."
- Leave spaces between strings and colons in a label.

When determining the default, minimum, and maximum sizes for your windows . . .

- Specify a default size for each window.
- If the window is resizable, specify a minimum size at which all controls and work areas will be visible and large enough to be usable. If the window is not resizable, set the minimum size equal to the default size.
- If the window is resizable, specify a maximum size such that your application window doesn't expand to fill screen space unnecessarily. If the window is not resizable, set the maximum size equal to the default size.

When considering window placement . . .

- Set a preferred window position for all primary windows. Don't set a required window position for primary windows.
- Try to anticipate other application windows that may be displayed with your application and set your preferred default position appropriately.

Guidelines for Keyboard Focus Across Windows

When designing your application windows . . .

- ❑ Make sure that your application works well under implicit focus across windows.
- ❑ Don't have your application move the pointer to another location on the screen. Always allow the user to control the position of the pointer on the screen.

When incorporating a "pointer grab" function into your application . . .

- ❑ If the user is always going to specify the data to capture with a single action such as a single mouse click or a single mouse drag, use the single-action pointer grab model; otherwise use the multiple-action pointer grab model.
- ❑ Display a standard or modified sighting pointer whenever your application window grabs keyboard focus. This indicates that the keyboard focus belongs to your application's window and that the pointer isn't currently following implicit focus across windows.

Minimized Window Guidelines

When designing images for your minimized primary windows . . .

- ❑ Use a color image rather than a two-color bitmap.
- ❑ Design your images to look best at the default size of 85x67 pixels.
- ❑ If your application is based on a single document model, create separate images for each of the primary windows. If your application is based on a multiple document model, create one image for the main window and a second image to use for all co-primary windows.
- ❑ Choose images that clearly identify the window that is minimized. If you have multiple images, make sure that the separate images work well together.
- ❑ Make sure that the images you use for minimized windows will be understood by an international audience.

- ❑ Don't use a snapshot of the desktop icon for the image. This could be confused with the real icon.

When choosing labels for your minimized primary windows . . .

- ❑ Limit the label to approximately twelve characters. If you need a few more characters than this, check that your label will fit with the default size and font for minimized windows.
- ❑ If your application is not document-based, use the application name as the minimized window label for the minimized main window. Use the label *Function* for minimized co-primary windows where *Function* is the same function as in the co-primary window's title bar.
- ❑ If your application is document-based and follows one of the single-document models, use *Filename* (or "Untitled" for new files) for the minimized main window label. Use *Function* for minimized co-primary window labels where *Function* is the same function as in the co-primary window's title bar.
- ❑ If your application is document-based and follows one of the multiple-document models, use the application name as the label for the main window (if it is visible). The co-primary windows in these models represent the multiple documents and should have the minimized window label *Filename* (or "Untitled" for new files).

When determining the behavior for a window that the user has chosen to minimize . . .

- ❑ Decide which operations should and should not continue to be processed while the window is minimized.
- ❑ Indicate status with the minimized window label if your application is typically minimized during long processes.
- ❑ Use the default screen locations supplied by *4Dwm* for the minimized window. Don't specify your own screen location.

Desks Guidelines

When designing your application . . .

- ❑ Make sure that all windows with associated support or dialogs are visible and mapped to the screen so that the support windows and dialogs appear only on the desk where their parent window displays.
- ❑ Don't design your application to manage the screen background.

Session Management Guidelines

When designing your application . . .

- ❑ Have your application create a command line that will launch the application and restore its current state. This current state should minimally include reopening any files that are currently open under the application and opening any primary or support windows that are currently open.
- ❑ Update this command line as the state of the application changes.
- ❑ If your application allows users to create and edit data files, have *4Dwm* notify your application when the user chooses "Log Out."

If your application is running when the user chooses "Log Out" and there are unsaved changes for a specific file . . .

- ❑ Save these changes into another file and name it something logical such as *original_file_name*.save. When the application is restarted at login, post a dialog that tells the user that this file with unsaved changes exists and query the user to determine whether to open the original file or the file with the unsaved changes.
- ❑ If you cannot implement the preferred strategy described above, ignore the user's unsaved changes. Do not automatically save the user's changes by default.

Software Installation Guideline

- ❑ Make sure that users can install and remove your application through the Software Manager, an Indigo Magic Desktop utility.

Guidelines for Designing Online Help

When designing access to online help for your application . . .

- ❑ Provide access in each window of your application from either a Help menu if the window has a menu bar or a *Help* button if the window doesn't have a menu bar.
- ❑ Use SGIHelp. This provides users with a familiar viewer and familiar navigation techniques when reading the online help for your application.

When defining the types of online help for your application . . .

- ❑ Provide context-sensitive help, overview information, task-oriented help, a list of keyboard shortcuts, product information, and an index of help topics.
- ❑ Provide context-sensitive help for all primary and support windows.
- ❑ Enable context-sensitive help mode when the user either chooses the "Click for Help" entry in a Help menu (if the window has a menu bar) or presses <Shift>-<F1> (whether or not the window has a menu bar). Change the pointer to a question mark when context-sensitive help mode is enabled.
- ❑ At a minimum, provide separate context-sensitive help for each control area, work area, status area, and menu in the window. This help should describe the purpose of the corresponding area and should include cross-references to task-oriented help topics which describe tasks which use this area.
- ❑ Provide overview information for all main windows whether help is provided from a menu or a button. This overview should briefly describe the functionality of the entire application.
- ❑ For co-primary and support windows that include a menu bar, provide overview information that describes the functionality of that specific window.
- ❑ Provide task-oriented information for all windows. This information should include step-by-step instructions for how to accomplish all of the tasks available in the current window.
- ❑ For windows with a menu bar, provide access to an index of help topics. This index should list all available help topics for the application

including those that are generated using the context-sensitive help mode and those that are available directly from the Help menu. In addition, users should be able to browse the index and select topics for reading.

- ❑ Provide keyboard shortcut information for all main windows (whether help is provided from a menu or a button) and for co-primary and support windows that include a menu bar. This information should include the mnemonics, accelerators, and function keys available for the entire application and not just for the current window.
- ❑ Provide product information for all main windows (whether help is provided from a menu or a button) and for co-primary and support windows that include a menu bar. This information should minimally include the product name and version number. It might also include other general product information such as copyright and trademarking, licensing, and customer support access.
- ❑ Display product information using an Information dialog so that users who don't install an application's online help can still access version number and customer support information.

When providing a Help menu in an application window . . .

- ❑ Include a "Click for Help" entry to enable context-sensitive help mode with the keyboard accelerator <Shift>-<F1>.
- ❑ Include an "Overview" entry for main windows. For co-primary and support windows, include an entry labeled "Overview for <*window name*>".
- ❑ Include entries that represent a list of tasks that users can accomplish in the current window. If this list of tasks is more than ten or twelve entries, use cascading menus. These entries shouldn't have mnemonics or keyboard accelerators.
- ❑ Include an "Index" entry that allows the user to access the help index.
- ❑ Include a "Keys & Shortcuts" entry to display all keyboard shortcuts for the application.
- ❑ Include a "Product Information" entry.

When providing a *Help* button in an application window . . .

- ❑ Provide a *Help* button for all windows that don't have a menu bar.

- ❑ For main windows, provide overview, task-oriented, keyboard shortcuts, and product information when the user clicks this button.
- ❑ For co-primary and support windows, provide overview and task-oriented information when the user clicks this button.
- ❑ For dialogs, provide help that focuses on the main purpose of the dialog and describes how to use the dialog.
- ❑ For primary and support windows that include a *Help* button, also provide access to context-sensitive help when the user presses <Shift>-<F1>. Dialogs typically don't support context-sensitive help mode.

Guidelines for Creating SGIHelp Content

When writing any online help for your product . . .

- ❑ Create separate help "cards" for each help topic.
- ❑ Limit each help card to no more than three viewer windows full of information.
- ❑ Write a descriptive heading for each help card.
- ❑ If a particular help topic needs supplemental information, provide links to that information rather than repeating it in the current card.
- ❑ Use language your users will understand.
- ❑ Use figures when appropriate. SGIHelp allows users to view graphics inline with the help text.

When writing the "Click for Help" context-sensitive information for your application . . .

- ❑ Begin by listing the individual controls and areas of your application windows that you need to describe.
- ❑ At a minimum, provide separate help cards for each group of controls and areas in that window.
- ❑ Provide descriptions in terms of the user tasks the components support.
- ❑ Don't include procedural, task-oriented information with the context-sensitive information—include links to the appropriate task-oriented topics instead.

When writing the overview help cards for your application . . .

- ❑ Restrict the content to information about what the product does, not how to use it.
- ❑ Limit the text to one or two viewer windows of information.
- ❑ Use the heading “Overview” for the main window’s overview help card and “Overview of <*window name*>” for co-primary and support windows with overview help cards.

When writing the task-oriented information for your application . . .

- ❑ Begin by listing the tasks that users will want to accomplish with your application.
- ❑ For each task, list the step-by-step instructions users will need to accomplish that task. If these instructions span more than three or four viewer windows, try to divide this topic into several smaller help topics.
- ❑ Provide a brief summary paragraph at the beginning of the help card, followed by the step-by-step information.

When writing the keyboard shortcuts information for your application . . .

- ❑ Include all shortcuts for your application in a single card—mnemonics, keyboard accelerators, and function keys.

When creating the index for your help topics . . .

- ❑ Match the titles in the index as closely as possible to the titles of the help cards.
- ❑ Place the topics in the index in the following order—overview, list of tasks, context-sensitive topics, and keyboard shortcuts.

When writing help information that will be available from a *Help* button rather than from a Help menu . . .

- ❑ For the main application window, the help card should contain an overview of your application, task-oriented information, a list of all keyboard shortcuts, and product information.
- ❑ For *Help* buttons not on the main application window of your application, present only the help information for the specific window.

- ❑ If the amount of information on this one help card spans more than three or four viewer windows of information, after the overview or summary information at the beginning of the help card, place links which take users directly to the other chunks of help information contained in that card.

After writing your online help . . .

- ❑ Have reviewers examine your help content online rather than reviewing a printed copy. Help topics will "read" differently depending on which paths readers (reviewers) traveled to get there.
- ❑ Have reviewers check the titles of the help topics to make sure they are descriptive and appropriate.
- ❑ Have reviewers test out all links to make sure they are appropriate.

Desktop Variables Guidelines

In general . . .

- ❑ Always honor the user's desktop customization settings. Never override or ignore them.

When considering color and font schemes for your application. . .

- ❑ Use the pre-packaged color and font schemes supplied by Silicon Graphics rather than designing your own.

When considering window placement . . .

- ❑ Set a preferred window position for all primary windows. Don't set a required window position for primary windows.
- ❑ Try to anticipate other application and tool windows that may be displayed with your application and set your preferred default position appropriately.

To allow users to control the language for your application . . .

- ❑ Check the value of the default language each time your application is launched. Don't reset this value while the application is running.

To allow users to control the mouse double-click speed for your application . . .

- ❑ Check the value of the double-click speed each time your application is launched. Don't reset this value while the application is running.

If users will be editing and / or browsing ASCII text files in your application . . .

- ❑ Make their preferred editor (specified in the Desktop control panel) available for use on text files.
- ❑ Check the value for the preferred editor each time your application is launched, but don't reset this value while your application is running.
- ❑ If users can only browse the ASCII text files, launch the editor in read-only mode.

File Monitoring Guideline

- ❑ If your application needs to stay in sync with the state of any part of the file system, use FAM. Don't have your application directly poll the file system to detect changes.

Data Exchange Guidelines

If your application contains data that users may wish to transfer to other applications or across separate instantiations of your application . . .

- ❑ Support the Clipboard Transfer Model using the "Cut," "Copy," and "Paste" entries in the Edit menu. In this model, the clipboard is a global entity that's shared by all applications. Your application shouldn't use these entries to refer to a clipboard that's private to your application.
- ❑ When supporting the Clipboard Transfer Model, don't select or highlight newly pasted data after a "Paste" operation.
- ❑ Support the Primary Transfer Model. Assert ownership of the primary selection when the user begins to make a selection. Insert data at the location of the pointer when the user clicks the middle mouse button (which isn't necessarily at the insertion cursor).

- ❑ When supporting the Primary Transfer Model, don't select or highlight newly transferred data after a transfer operation.
- ❑ Use *persistent always selection* highlighting (keep the current selection highlighted even when your application loses the primary selection), unless the only action that can be performed on the selection is to copy the data using primary data transfer. In this case, use *nonpersistent selection* highlighting—that is, remove the selection highlight when the selection is no longer the primary selection.
- ❑ When supporting the Primary Transfer Model, if the current active window has a selection that isn't the primary selection, reinstate this selection as the primary selection if the user presses <Alt-Insert>. Additionally, you can include a "Promote" entry in the Edit menu to perform the same function.
- ❑ When supporting the Primary Transfer Model, when the user begins to modify a selection, such as adding elements to it, reassert ownership of the primary selection if your application does not currently own it.
- ❑ When supporting both Clipboard Transfer and Primary Transfer, keep the primary selection independent from the clipboard. When the user begins to make a selection in your application, assert ownership of the primary selection but do not change the ownership of the clipboard. When the user chooses "Cut" or "Copy" from an Edit menu in your application, assert ownership of the clipboard but do not change the ownership of the primary selection.

Application Model Guidelines

For all applications . . .

- ❑ Choose an appropriate application model for combining the different types of windows in your application.
- ❑ Use only the allowable parent-child window relationships and keep your application window hierarchy shallow.

Primary Window Guidelines

When designing a primary window . . .

- ❑ Use a menu bar unless all of the window's functionality is available through pushbuttons. Don't use a "floating" menu bar in a separate window.
- ❑ Support keyboard accelerators for Close (Ctrl-W) and Exit (Ctrl-Q) as appropriate, even if the window doesn't have a menu bar.

When designing a scrollable work area in a primary window . . .

- ❑ Use a vertical scrollbar on the right side of the work area when the data being displayed in the work area may not fit in a vertical direction. Use a horizontal scrollbar directly below the work area when the data may not fit in a horizontal direction. Don't use scrollbars if you're certain the data will fit.
- ❑ Disable the appropriate scrollbar when all the data is visible in a given direction. Don't remove the scrollbar.
- ❑ Make each scrollbar span the entire height or width of the work area. Don't include controls or status information in the scrollbar region.

When designing control areas in a primary window . . .

- ❑ Place controls below horizontal scrollbars or to the left of work areas.
- ❑ Provide pushbuttons for the most frequently accessed application-specific functions from the pull-down menus. Don't use pushbuttons for standard menu entries such as Open, Save, Close, Exit, Cut, Copy, Paste, and Help.
- ❑ Use pushbuttons only for functions that appear in menus, unless the pushbuttons are part of a tool palette.
- ❑ Provide an area for command-line input, if appropriate, in addition to (not in place of) pushbuttons.

To display status information . . .

- ❑ Use a status area along the bottom of a primary window if your application needs to post frequent messages about its status. Provide vertical scrollbars for this area so that users can view previously displayed messages.

- ❑ Use a status area to display messages that the user doesn't have to respond to rather than posting this noncritical information in dialogs. However, don't put critical warning or error messages in the status area (use a dialog instead).
- ❑ Don't use the status area to display help information.

When dividing a primary window into panes . . .

- ❑ Divide panes using separator lines. If users might need to resize the pane, also include a sash control.
- ❑ Try to limit the number of panes in a single window to four with no more than three sash controls.
- ❑ If certain panes are optional, allow users to hide or show these individual panes using entries in the "View" menu.

Support Window Guidelines

When designing support windows . . .

- ❑ Use them to provide access to sets of related controls.
- ❑ Allow users to access them either through entries in the Tools menu or through pushbuttons in a tool palette in the parent primary window.
- ❑ Be sure that each support window has a visible parent primary window that's mapped to the screen.

When designing the layout of a support window . . .

- ❑ Make the layout simple and static. Don't include multiple panes of information.
- ❑ Include a response area for standard actions that's similar to the one dialogs have.
- ❑ Don't include a menu bar in most cases, but do support the keyboard accelerator for Close (Ctrl-W).

When opening support windows . . .

- ❑ Avoid overlapping the work area of the parent window.

- ❑ Bring them up as modeless secondary windows.

When allowing the user to make color choices . . .

- ❑ Use the Indigo Magic color chooser whenever you want to offer the user an unrestricted choice of colors. For a restricted choice of colors, use a palette of colors to choose from, a list, an option button, or a set of radio buttons, depending on the number of choices available.

Pointer Behavior Guidelines

When designing your application . . .

- ❑ Always allow the user to control the location of the pointer; your application shouldn't change the position of the pointer.
- ❑ Don't change the gain or acceleration characteristics of the pointer. If your application requires fine manipulation, provide a zoom feature in the View menu.

Keyboard Focus and Navigation Guidelines

When designing keyboard focus and navigation for your application windows . . .

- ❑ Use explicit focus for navigating among components within a window.
- ❑ Support at least the minimum required functionality from the keyboard, such as navigating to and entering data into editable text fields, using mnemonics and keyboard accelerators to access menu entries, and scrolling any scrollable component. Keep in mind that some users use alternate input devices that rely on having functions available from the keyboard.
- ❑ When the window becomes active for the first time, give focus to the component that the user is most likely to want to interact with using the keyboard. When a user returns the keyboard focus to a window that was previously the active window, return the keyboard focus to where it was when the user moved the focus out of that window.

- ❑ Put each component that requires the use of arrow keys to control it in its own field. The following components are by default put in fields of their own: editable text fields, lists, scrollbars, and sashes.
- ❑ Don't use the default keyboard navigation keys for other purposes. These keys are <Tab>, <Ctrl>-<Tab>, <Shift>-<Tab>, <Ctrl>-<Shift>-<Tab>, the arrow keys, <F10>, <Shift>-<F10>, and <Ctrl> in combination with a left mouse button click.

Selection Guidelines

For each collection of data . . .

- ❑ Use one of the four recommended selection models—single selection, browse selection, range selection, or discontiguous selection. Don't use the multiple selection model.
- ❑ Automatically scroll the data as the user drags the pointer out of the scrollable data display region.
- ❑ Determine if your users will need to create or modify a selection using the keyboard. If so, then support the keyboard actions defined in Section 4.1.6 of the *OSF/Motif Style Guide*. (These actions are automatically supported if you use the IRIS IM list or text components.)

When highlighting a selection . . .

- ❑ Update the highlighting continuously as the user initiates and extends the selection.
- ❑ Use persistent always highlighting, unless the only reason a user can select this data is to transfer it using the primary transfer model. In this case, use nonpersistent highlighting.

When managing multiple collections of data in a single window . . .

- ❑ Deselect the previous selection whenever the user makes a new selection in any of the collections for cases where the user can select data in only one collection at a time.
- ❑ Apply the operation to the collection that most recently had a selection made in it when the user can select data in more than one collection at a

time and there are mouse, keyboard, or menu commands that can be applied to more than one of the collections.

Drag and Drop Guidelines

When designing drag and drop for your application . . .

- Cancel a drag and drop operation if the user presses <Esc>, and leave both the object and the target as they were before the operation was initiated.
- Use the left mouse button for both selecting and dragging non-text objects. Use the standard cursor in this case.
- Use the middle mouse button for dragging text, and replace the cursor with a drag icon when the text is being dragged.

Menu Traversal and Activation Guidelines

In general, when designing traversal and activation for your menus . . .

- Allow users to activate and traverse the menus using the default IRIS IM behaviors for mouse and keyboard actions.
- If a user closes a menu by clicking somewhere outside of the menu, the application should ignore this click so that users don't lose selections they've made in the window just because they display and close menus.
- Allow users to display and close popup menus using the key combination <Shift><F10>. When <Shift><F10> displays a popup menu, the location cursor should be on the first available menu entry. When <Shift><F10> closes the menu, the keyboard focus should be returned to where it was before the menu was displayed.

Pull-Down Menu Guidelines

In general, when designing pull-down menus in a menu bar . . .

- ❑ Be sure that users can access most of your application's functionality from the menu entries. At a minimum, make sure that the core functionality can be accessed from the menus.
- ❑ Don't include more than a 10-12 entries in a menu and make sure that all of your entries can fit on the screen at one time.
- ❑ Provide mnemonics for all menus and menu entries. In most cases, the mnemonic should be either the first character of the name or, if there's a conflict, a character that's strongly associated with and included in the name. Use standard mnemonics for standard menus and entries.
- ❑ Limit the use of tear-off menus. Instead, use support windows for groups of controls that users might want to use continuously.

When selecting specific menus and entries for an application window . . .

- ❑ Use the standard menus and menu entries as the basis for the overall design of the menu structure. Include all standard menus and entries that are applicable to your application.
- ❑ Include a Help menu as the rightmost menu.
- ❑ Include an "Undo" menu entry, particularly if users can perform actions that destroy or significantly change their data .
- ❑ Include an "Exit" menu entry for all main windows and for co-primary windows if users will want to completely exit the application from that co-primary window.
- ❑ Include a "Close" menu entry for all co-primary windows and support windows that have menu bars. Don't provide a "Close" entry for main windows.
- ❑ Include menu entries that repeat the functionality of any pushbuttons on the primary window.
- ❑ Include menu entries for actions that are accomplished using a direct manipulation method or a mouse shortcut such as double-clicking.
- ❑ Include menu entries for accessing all primary and support windows that are children of the current window.

- ❑ Don't include entries for functions that aren't available for the current version of your application.

When naming menus . . .

- ❑ Use one-word titles for menus.
- ❑ Use the standard titles for menus (for example, File and Edit) if they're applicable, but change the standard title if this will make the function more clear.
- ❑ Don't use a standard menu title if you're changing the standard definition.

When naming menu entries . . .

- ❑ Use the standard names for standard menu entries, but don't use a standard name for a menu entry that doesn't support the standard behavior.
- ❑ Each entry name should be an action word, the value of a parameter, an attribute name, the name of a cascading menu, or the name of a co-primary, support, or dialog window. Don't use more than two words (except for task-oriented Help menu entries), and avoid using graphic labels for menus entries unless the graphics make the functionality more clear.
- ❑ Choose descriptive names that help users learn the functionality of the application. For cascading menus, choose a name that clearly implies the contents of the menu.
- ❑ Add a word if necessary to be sure the entry clearly indicates what entity will be acted upon. For example, you might use "New *object*" such as "New Folder" or "New Page" rather than just "New."
- ❑ If a menu entry toggles its state, use a checkbox and leave the menu entry name the same for the different states ("Italics"). If this won't be clear, toggle the name so that it indicates what action will be taken if the menu entry is selected ("Show Grid," "Hide Grid").
- ❑ Capitalize the menu entry using the same rules as capitalizing book titles.
- ❑ Display an ellipsis (...) after menu entries that bring up a dialog that requests more information from the user. Don't use ellipses if the dialog

simply brings up information that the user requested (for example, a Help dialog).

When ordering menus and menu entries . . .

- ❑ Place the standard menus in the standard order (File, Selected, Edit, View, Tools, Options, Help), even if you have renamed any of these menus. Place any new menus between the View and Tools menus.
- ❑ Place standard menu entries in the standard order. “Close” and “Exit” are always at the end of the leftmost menu whether or not this menu is named File.
- ❑ Group menu entries logically. If a new menu entry is related to one of the standard menu entries, place it near that standard menu entry.
- ❑ Place items in the menu first according to the order they will be used, and second according to their frequency of use (with more frequently used items closer to the top of the menu).
- ❑ Alphabetize entries that can be determined only when the user launches the application. If this alphabetized group appears in a menu that contains other entries, place the group at the end of a menu and use a separator between it and the preceding entries.
- ❑ Use radio buttons for mutually exclusive menu entries, and checkboxes for a group of related menu entries, any number of which can be selected at any one time.
- ❑ Use separators when necessary to group items—for example, to set off a group of related entries that use radio buttons or checkboxes.
- ❑ Limit the use of cascading menus. Never use more than one level of cascading menus.

When selecting keyboard accelerators for menu entries . . .

- ❑ Use standard keyboard accelerators for standard menu entries; don’t use any of the standard accelerators for your own entries, even if you’re not using those standard entries.
- ❑ Provide keyboard accelerators for the most frequently used menu entries. Don’t provide accelerators for all menu entries.
- ❑ Use the key combination <Ctrl>*character*. Don’t use the key combination <Alt>*character* because this conflicts with mnemonics.

- For pairs of menu entries where one entry reverses the results of the other entry (“Undo” and “Redo”), use <Ctrl>*character* for the most frequently used entry and <Shift><Ctrl>*character* for the other entry where *character* is the same for both accelerators.
- Display all characters in keyboard accelerators as uppercase (for example, display <Ctrl>s as “Ctrl+S”). For keyboard accelerators that involve uppercase characters, show the <Shift> key as part of the keyboard accelerator (for example, display <Ctrl>S as “Shift+Ctrl+S”).

When deciding when to disable menu entries . . .

- If selecting the menu entry in the current context would give the user an error message, show the menu entry as disabled (dimmed).
- Avoid using dynamic entries. Rather than removing an entry when it’s temporarily unavailable, include it and disable it as appropriate.

Popup Menu Guidelines

When choosing when a popup menu should appear . . .

- At most, provide a different popup menu for each main area (that is main field or main pane) of the window. Don’t change the availability of a popup menu based on what graphical element the pointer is over or based on the selection state of any of the graphical elements.

When deciding what to include in a popup menu . . .

- Include entries for the most commonly used functions from the pull-down menus, and use the same names in the same order as they’re displayed in the pull-down menus.
- Avoid entries that require checkboxes or radio buttons. These are typically not the most commonly used menu functions.
- Don’t make menu entries the sole access to these functions.
- Don’t change the content of the menu based on what graphical element the pointer is over, or based on the selection state or contents of this element. Instead, put all entries in the popup menu for the main area of the window, then enable and disable entries as appropriate.
- Don’t include cascading menus and don’t use tear-off menus.

When displaying the contents of the popup menu . . .

- ❑ Include a title that's either the name of the application, or if the application has more than one popup menu, that describes the purpose of the menu.
- ❑ Use only one separator, which goes between the title and the individual menu entries.
- ❑ Show ellipses and keyboard accelerators if these are shown in the corresponding pull-down menu entry, but don't show mnemonics.
- ❑ If selecting the menu entry in the current context would give the user an error message, show the menu entry as disabled (dimmed). Don't remove the menu entry when it's temporarily unavailable.

Pushbutton Guidelines

When using pushbuttons . . .

- ❑ In windows with menu bars, use pushbuttons to provide easy access to the most frequently used application-specific functions in the pulldown menus. For primary windows, these pushbuttons appear in the control area of the window.
- ❑ In windows without menu bars, use pushbuttons to access help and to close the window.
- ❑ Use pushbuttons to create tool palettes, either in support windows or in primary windows.
- ❑ Use pushbuttons in the response area of a dialog for the standard actions for that dialog.
- ❑ Always have the pushbutton perform the same operation (although the input to that operation may vary based on what data is currently selected). Don't use the same pushbutton to perform different tasks based on some mode of the application.
- ❑ Use pushbuttons to perform an action; don't use them merely to set state, such as a parameter value in a dialog box. Use checkboxes, radio buttons, or option menus for this purpose.

When labelling a pushbutton . . .

- Use either a text or graphic label that describes the action associated with the button. With text labels, use an active verb describing the operation the button performs. Each text label should be a single, capitalized word.
- Center the label on the button.
- If the pushbutton opens a dialog to collect more information from the user before the action represented by the pushbutton can be completed, place an ellipsis after the button label. Don't use an ellipsis if the button opens a dialog simply to display some information to the user as an end result of the operation. This use of ellipses is the same as that described for menu entries in the section "Naming Menu Entries in the Pull-Down Menus" in Chapter 8.

When displaying pushbuttons . . .

- If the action associated with a button is temporarily unavailable, disable the button rather than remove it.
- Don't resize pushbuttons when the window is resized.
- Don't use dynamic buttons whose labels change to indicate different functionality depending on the current context. Instead, use multiple buttons and disable buttons that represent functionality that's currently unavailable. With multiple buttons, the functionality is obvious even if some of the buttons aren't currently active. With dynamic buttons, the user has to put the application intothe proper context to discover some of the functionality. The one exception to this guideline is the *Cancel* / *Close* button used in Dialogs with the *Apply* button. See "Standard Dialog Actions" in Chapter 10 for information on this special case.

Option Button Guidelines

When using option buttons . . .

- Use an option button when you want to offer the user about 5-12 mutually exclusive options; use a list for more than 12 choices. If there's enough space, use radio buttons for fewer than 5 choices.
- Don't put radio buttons or checkboxes in an option menu.

- ❑ Don't use an option button if the user can select several options at the same time—use a list or a set of checkboxes instead.
- ❑ Don't put actions (such as zoom or rotate) in the option menu—use pulldown menus or pushbuttons instead.
- ❑ Don't add or delete the choices in the option menu. If the choices must change, use a list.
- ❑ Don't use cascading menus in the option menu. If there are so many items that they don't fit conveniently into an option menu, use a scrolling list instead.
- ❑ Don't use a tear-off entry in an option menu.

When labelling an option button . . .

- ❑ Use the default label for the option button itself, which is the current value of the parameter.
- ❑ Use a second label that describes the parameter that the option button controls. This parameter label should be to the left of the option button and should be followed by a colon (:) and a space (see Figure 9-2). This label is typically a noun.

When labelling the entries in an option menu . . .

- ❑ Use nouns that indicate the possible values of the parameter being set.

When displaying option menus . . .

- ❑ If one of the entries in an option menu is unavailable for selection in the current context, disable the menu entry. Don't remove the entry from the menu. Note that the user should always be able to display the contents of an option menu even if all of the menu entries are currently disabled.
- ❑ Don't include a title in option menus.

Checkbox Guidelines

When using checkboxes . . .

- ❑ Use checkboxes for single attributes or states that can be turned on and off, or for groups of items where multiple items in the group can be

selected independently. (Also see "Using Radio Buttons and Checkboxes in Pull-Down Menus" in Chapter 8.)

- ❑ Use checkboxes for groups of less than about six items. When dealing with more than a handful of items, use a list that allows multiple elements to be selected at the same time.
- ❑ Don't use checkboxes for mutually exclusive options. If only one item in a group of items can be selected at a time, use radio buttons instead.
- ❑ Don't use checkboxes for actions; use pushbuttons instead.
- ❑ Don't change the choices in the group based on the current context. If you want to offer a dynamic set of choices, use a list.

When labelling checkboxes . . .

- ❑ Give each checkbox a label that describes the attribute, state, or option it controls.
- ❑ Create a group label for each group of checkboxes, and indent the checkboxes below the label. This group label should be a noun that describes the function of the group.

When displaying checkboxes . . .

- ❑ Keep checkboxes updated to reflect the current state of the application and the settings of the current selection (if the settings of the checkboxes relate to the current selection). For example, if you have a checkbox for turning underlining on and off and the user selects some text, the checkbox should be updated to reflect whether or not the selection is underlined.
- ❑ Disable checkboxes representing choices that aren't currently available. Don't remove the checkboxes.

Radio Button Guidelines

When using radio buttons . . .

- ❑ Use radio buttons in groups, never as single buttons. If you need to use a single button that shows an on / off state, use a checkbox instead. (Also see "Using Radio Buttons and Checkboxes in Pull-Down Menus" in Chapter 8.)

- ❑ Use radio buttons for mutually exclusive options. If more than one item in the group can be selected at a time, use checkboxes or a list instead.
- ❑ Use radio buttons when you want to offer the user fewer than six options. If you have more than six options, or if screen space is extremely limited, use an option button instead. If you have more than 12 options, you should consider using a list where only a single element can be selected at a time.
- ❑ Don't use radio buttons for actions; use pushbuttons instead.
- ❑ Don't change the choices in a group of radio buttons based on the current context. If you want to offer a dynamic set of choices, use a list because users expect the elements of a list to change occasionally, but they don't expect radio buttons to change.

When labelling radio buttons . . .

- ❑ Give each radio button a label that describes the attribute or option it controls.
- ❑ Create a group label for each group of radio buttons, and indent the radio buttons below the label. This group label should be a noun that describes the function of the group.

When displaying radio buttons . . .

- ❑ Keep radio buttons updated. If the settings of the radio buttons depend on the current selection, they should be updated when the user makes a new selection so that they reflect the settings of the new selection.
- ❑ Disable radio buttons representing options that aren't currently available. Don't remove the radio buttons.

List Guidelines

When using lists . . .

- ❑ Use a list when you want to allow the user to choose a single option from a large list (that is, more than 15 options). If you have fewer than 15 options, use either an option button (best for 5-15 options; or a set of radio buttons (best for 2-5 options).

- ❑ Use a list when you want to allow the user to choose several options from a list of six or more elements. If you have fewer options, use checkboxes.
- ❑ If you want to allow the user to choose elements from a dynamic list of options, use a list regardless of the number of options. (Option menus and groups of checkboxes or radio buttons should represent static lists of options.)

When labelling a list . . .

- ❑ Label the list with a noun that indicates the function of the elements in the list.
- ❑ Place the label directly above and either left-aligned with or slightly to the left of the first element of the list.

When labelling the list entries . . .

- ❑ If the elements in the list represent operations to perform, they should be active verbs. Otherwise, they should be nouns.

When displaying lists . . .

- ❑ When a window using a list is first opened, the currently selected list elements should be highlighted and the list should be scrolled to display these. If multiple elements are selected, scroll the list so that the first selected one appears at the top of the viewing area. See “Selection” in Chapter 7.
- ❑ Allow users to select elements in the list according to the selection guideline discussed in “Selection” in Chapter 7.
- ❑ Disable list elements that aren’t currently available.
- ❑ Allow the list to autoscroll (the default behavior) if the user is making a selection and the selection goes outside the range of the displayed elements. See “Selection” in Chapter 7.

Text Field Guidelines

When using text fields . . .

- Use single-line, editable text fields to display values of parameters that users can edit.
- Use single-line, noneditable text fields to display values of parameters that users can't edit, whenever these values either change over time or might need to be selected by the user. If the value doesn't change and the user doesn't need to select it, use a label.
- Don't use a text field if you need to display and edit pathnames; use the Indigo Magic File Finder instead.
- Use text fields for values that change over time; don't use labels.

When labelling text fields . . .

- Label each editable or noneditable text field, unless the field represents the bulk of a window and the field's function is clear.
- For single-line text fields, place the label to the left of the text field, and follow the label with a colon (:) and a space. The label should be vertically centered within the text field.

When displaying text fields . . .

- Use the default selection and highlighting discussed in "Selection" in Chapter 7.
- Allow the user to cancel a text edit in progress by pressing <Esc>. That is, once the user has selected text and started to replace it with new text, <Esc> should cancel any changes that the user has made.
- Keep text fields updated. When a window using a text field is first opened, the default or current setting (if either exists) for the text field should be shown.
- Make the text automatically scroll if the user is making a selection and the selection goes outside the range of the displayed elements.
- When an editable text field can't be edited in the current context but the information is still useful to the user, change it to a noneditable text field. If the information isn't useful to the user (that is, the user doesn't need to know the value and won't need to select it), disable the text field.

Scrollbar Guidelines

When using scrollbars . . .

- Use scrollbars to pan an associated view.
- Use scrollbars with components that can be resized such that all of the available information contained in the component can't be displayed at one time. Typical scrollable components include work areas in primary windows, lists, multiple line text fields, and data display areas in primary or support windows.
- Use scrollbars with a list when the number of elements in the list doesn't fit in the viewing region (vertical scrollbar), when the elements are too wide to fit in the viewing region (horizontal scrollbar), or when the window containing the list can be resized such that either of these situations can occur.
- Use scrollbars with multi-line text regions when the data can't all be displayed vertically or horizontally or when the window can be resized such that this is true.
- Don't use scrollbars with single-line text fields.
- Don't use scrollbars for zooming or for rotation. Use an Indigo Magic thumbwheel instead.
- Don't use scrollbars to choose a value in a range; use the Indigo Magic scale instead.

When displaying scrollbars . . .

- Place vertical scrollbars along the right of the element being scrolled, and place horizontal scrollbars along the bottom of the element being scrolled.
- Keep scrollbars updated. When a window using a scrollbar is first opened, the scrollbar should reflect the current area being displayed in the scrolled region.
- Update the data in the scrolled area continuously as the user drags the slider along the scroll region. This gives the feeling of direct, continuous control. Don't wait until the user has released the slider to update the data, because users often use the current view of the data to determine when to stop dragging the slider.

- When a component is being scrolled, don't scroll it beyond the first or last elements. That is, there should be no extra white space before the first element or after the last element. The exception to this rule is scrolling text elements that represent physical pages (for example, in a desktop publishing application).
- Make all components that use scrollbars automatically scroll when the user makes a selection that goes outside of the data currently being displayed. Also, make the component automatically scroll if the user performs an operation that moves the cursor outside of the current view (for example, if the user inserts or deletes text that moves the cursor outside of the current view). In this case, the view should be automatically scrolled so that the cursor is shown when the operation is finished.
- When using the <Page Up>, <Page Down>, <Ctrl>-<Page Up>, or <Ctrl>-<Page Down> key sequences to scroll a page at a time, leave one unit of overlap from the previous page to make it easier for the user to preserve the current context. This unit is application-specific; it might be a line of text, an item in a list, a row of icons, or a specfic number of pixels (for example, in a drawing region). By default, this behavior is automatic for IRIS IM list and text components.
- Remove the slider from the scrollbar when all of the data is currently being displayed. Don't remove the scrollbar or disable it in some other fashion.
- Allow the user to cancel scroll actions by pressing <Esc>. By default, if the user presses the <Esc> key while dragging the slider along the scroll region, the scroll action is canceled, and both the data and the slider are returned to the position they had before the user initiated the scroll action.

Indigo Magic Scale Guidelines

When using the Indigo Magic scale . . .

- Use scales to allow users to change a value in a given range. Use scales in display-only mode to display values that the user can't control. For example, use a display-only scale as a percent-done indicator to show progress in a Working dialog. (See "Working Dialogs" in Chapter 10.)

- ❑ Don’t use scales for scrolling.

When labelling a scale . . .

- ❑ Label it with the current value for the scale.
- ❑ If the function of the scale isn’t immediately apparent, give the scale an additional label that indicates its purpose.

When displaying scales . . .

- ❑ Keep scales updated. When a window using a scale is first opened, the slider of the scale should show the current setting for the scale control.
- ❑ For sliders where the user can change the value, update the value being manipulated as the user moves the slider. It should give the impression of direct, continuous manipulation. For sliders that also manipulate an object, update the object continuously as well. For sliders that are used only to display values, the slider should be immediately updated to reflect the new value as the value changes.
- ❑ Allow the user to cancel a scale operation by pressing <Esc>. If the user presses the <Esc> key while manipulating the scale, the action should be canceled, and the scale should return to the position it had before the user initiated the action.

Label Guidelines

When using labels . . .

- ❑ Use labels for displaying text information that the user won’t need to edit or select.
- ❑ Use labels for labeling controls as described under the individual controls in this chapter.
- ❑ Use labels for labeling groups of controls. When used to label a group of controls, the label should be followed by a colon (:) and a space, and should be placed either to the left of the item in the upper left corner of the group or above and slightly to the left of the item in the upper left corner of the group.
- ❑ Use labels for simple instructions when necessary. Before adding instructions to any of your application windows, however, first try to

design some alternatives that might make the instructions unnecessary. For example, if these instructions are necessary because the user interface works in a nonstandard way, redesigning the interface to be more standard is likely to make the instructions unnecessary.

- ❑ Place labels on the background of the window (that is, the part of the window that isn't recessed).

When displaying labels . . .

- ❑ Don't change the text or graphic on a label. If this information will change, consider putting it in a noneditable text field instead; users don't expect label text to change.
- ❑ Disable labels when the controls they represent are disabled. Don't disable group labels.

File Finder Guidelines

When using the File Finder . . .

- ❑ Use the File Finder when the user needs to enter the pathname of a directory or file. This allows the user to drag and drop desktop icons to specify the file and to navigate the file hierarchy.
- ❑ When a window using a file finder is first opened, the text field in the file finder should show the default or current value of the pathname, if any. This value should also be placed in the history list under the history button.

Thumbwheel Guidelines

When using thumbwheels . . .

- ❑ Use thumbwheels to change the values of continuous variables (that is, variables that don't have discrete values). For discrete values, consider a scale or dial instead.
- ❑ Use thumbwheels with finite ranges for zooming operations and thumbwheels with infinite range for rotating objects.

- When a thumbwheel is used to change a value that has a clear default, provide a home button. For example, a Directory View window has a thumbwheel that allows the user to set the size of the desktop icons. Pressing the home button on this thumbwheel sets the icons to their default size.
- Use thumbwheels when screen real estate is extremely limited.
- Don't use a thumbwheel for panning; use a scrollbar instead. A scrollbar gives the user much more information about the object being scrolled than a thumbwheel could.

When displaying a thumbwheel . . .

- Update the object or value being manipulated as the user moves the thumbwheel. The thumbwheel should give the impression of direct, continuous manipulation.

Dial Guidelines

When using dials . . .

- Use dials as an alternative to scales for setting parameters. Dials are best for numeric parameters where the range of allowable values is small and the values are discrete.

When labelling dials . . .

- Place a label either directly below or directly above the dial, specifying the parameter that the dial controls.
- When you have a group of dials, place each dial label in the same position relative to its dial (that is, either all the labels should be below the dials or all the labels should be above the dials).

When displaying dials . . .

- When a window using a dial is first opened, the dial should show the current setting.
- As a dial is rotated, update the value being manipulated to reflect the new value on the dial. The dial should give the impression of direct,

continuous manipulation. Also, if the dial is controlling an object, continuously update the object as the dial is manipulated.

Guidelines for Using the Various Types and Modes of Dialogs

When choosing the type and mode of a dialog . . .

- ❑ Use a Prompt dialog to ask users for specific information. This dialog can be modeless or modal.
- ❑ Use an application-modal Error dialog to tell users about an error they've made in interacting with your application.
- ❑ Use an application-modal Warning dialog when there's an action pending that will cause users to lose data.
- ❑ Use an application-modal Question dialog to ask users a specific question that they must respond to before continuing to interact with the application.
- ❑ Use a Working dialog when an operation takes more than 5 seconds to complete. This dialog can be modeless or modal.
- ❑ Use a modeless Information dialog to give users information that's of immediate importance. Use this type of dialog sparingly.
- ❑ Use the modeless Indigo Magic File Selection dialog to allow users to navigate the file hierarchy and choose a file.
- ❑ Don't use system-modal dialogs.
- ❑ Use modal dialogs to show static information, and update modeless dialogs dynamically as the current state changes.

Guidelines for Designing Dialogs

When choosing the window decorations, initial state, and layout of dialogs . . .

- ❑ Associate every dialog with a primary or support window (its parent) that's mapped to the screen.

- Use the window decorations and Window menu entries listed in Table 3-1 and described in “Window Decorations and the Window Menu” and “Rules for Labeling the Title Bar in Windows Other Than Main” in Chapter 3.
- Have the default size large enough to allow all of the components and information to be displayed in their entirety.
- Place the dialog on the screen either near (but not overlapping) any related information in the parent window, or in the center of the parent window if the contents of the dialog aren’t related to the contents of the parent window.
- Locate the initial keyboard focus in the field with which the user is most likely to want to interact.
- Be sure the information being displayed in the dialog matches the current state of the application. If the dialog is modeless, update this information dynamically.
- Include a response area that contains standard dialog actions (pushbuttons) tailored to the type and purpose of the dialog. Also include an input area that consists of whatever controls are necessary for selecting objects or setting application parameters in Prompt dialogs. Include a message area in Error, Warning, Question, Working, and Information dialogs.
- Don’t include secondary work areas; if you need additional work areas, use a support window instead.
- Don’t include menus. If the dialog includes so much functionality that menus are necessary, you should probably use a support window.

When choosing pushbutton actions for dialogs . . .

- Use a subset of the standard dialog actions (Yes, No, OK, Close, Apply, Retry, Stop, Pause, Resume, Clear, Reset, Cancel, and Help), and have them appear in that order. If you include additional buttons, they should appear after the OK and Apply buttons but before the Cancel and Help buttons.
- Include a Help button unless the situation is explained thoroughly in the dialog.
- Avoid using both OK and Apply on the same dialog.

- ❑ To decide between OK and Apply, determine whether users are more likely to use the dialog to make one set of changes at a time (if so, use OK), or whether they're more likely to want to make and apply changes repeatedly before closing the dialog (in this case, use Apply).
- ❑ Include a Cancel / Close button on any dialog that has an Apply button.
- ❑ Include a Cancel button on Working dialogs and, if possible, a Pause button (with the option of later resuming).

When choosing and creating default actions . . .

- ❑ Whenever appropriate, choose one of the actions to be the default action.
- ❑ Have OK be the default action for Information dialogs (which typically have buttons only for OK and Help).
- ❑ Have the response that users are most likely to select be the default action for Question, Warning, Error, and any other dialogs that contain buttons but no text fields.
- ❑ Have the response that users are most likely to select after entering a text string be the default action for dialogs that have only one text field. Use no other controls than the buttons in the response area (such as the File Selection and Prompt dialogs).
- ❑ Don't have a default action for dialogs that contain multiple text fields.

When labeling dialog buttons . . .

- ❑ Replace the "Yes" and "No" labels in Warning and Question dialogs with button names that clearly indicate the specific action that will occur if the button is clicked.
- ❑ Replace the "OK" or "Apply" labels in Prompt or Warning dialogs with button names that clearly indicate the specific action that will occur if the button is clicked, unless the button is used for more than one purpose, or in the rare instance that "OK" and "Apply" are used together in a Prompt dialog.
- ❑ In all other cases, use the OSF / Motif standard names.

When deciding what content to include in specific types of dialogs . . .

- ❑ Use Prompt dialogs to collect information in related chunks—that is, avoid collecting unrelated pieces of information in the same dialog, and don't launch multiple dialog boxes sequentially to collect several pieces of information if these pieces are frequently collected at the same time.
- ❑ Include a description of the error, step-by-step instructions for how to recover from it, and a pointer to more information in Error dialogs. If the error involves a specific entity (for instance, a file, user, or host), name the entity in the error message.
- ❑ Invoke Error dialogs only when they're directly relevant to the user; for example, don't tell the user that the printer is out of paper until the user has a job in the queue.
- ❑ State what data is likely to be lost and why, and give the user a chance to cancel the action in Warning dialogs.
- ❑ Limit your use of Question dialogs to those situations where the user couldn't have provided the information in advance.
- ❑ Don't use Question dialogs for questions that relate to a pending destructive action—for these cases, use Warning dialogs instead.
- ❑ Dynamically indicate how much of the operation is complete with the Indigo Magic scale used as a percent-done indicator in Working dialogs.
- ❑ Switch from the general-purpose pointer to the watch pointer in the parent window of a modal Working dialog.

Guidelines for Invoking Dialogs

When determining when to display a dialog and which dialog to display . . .

- ❑ Limit the use of dialogs to those cases when they're absolutely necessary. Don't use dialogs to provide general status information—use a status area in the associated primary or support window instead.
- ❑ Invoke a dialog whenever users choose a menu entry that includes an ellipsis.
- ❑ Display the Indigo Magic File Selection dialog when the user chooses "Open..." from the File menu. The first time this dialog is opened, it

should show the current working directory and no specific file. Subsequently, it should come up in the state it was last in when the user dismissed it.

- ❑ Display the Indigo Magic File Selection dialog when the user chooses "Save As..." from the File menu. If the file being saved doesn't exist yet, the dialog should show the current working directory and no specific file. If the file exists, the dialog should show that file's directory, and the current name of the file should be selected.
- ❑ When users open, close, or save changes to files, prompt them with a Warning dialog whenever these actions will cause data to be lost:
  - In an application that allows only one document to be open at a time, when the user chooses to open another (new or existing) document and there are unsaved changes in the currently opened document. (For example, the user chooses "New," "Open," or "Reopen" from the File menu.)
  - When the user chooses "Close" from the File menu in a co-primary window, and the co-primary window contains data that will be lost if the window is closed.
  - When the user chooses "Exit" from the File menu, and at least one open co-primary window contains data that will be lost if the application is exited. For applications that support multiple open documents, prompt the user with a separate dialog box for each file that's currently open and has unsaved changes.
- ❑ Prompt users with the File Selection dialog when they choose Save from the File menu and the current file is untitled. The behavior is the same as the "Save As..." entry in this situation.
- ❑ Prompt users with a Warning dialog when they're interacting with the File Selection dialog and they choose a filename that already exists.
- ❑ Prompt users with a Warning dialog when they choose "Revert" from the File menu and the file currently has unsaved changes.
- ❑ Display an Information dialog when a user chooses the "Product Information" entry from the Help menu.
- ❑ Display the Working dialog when users initiate an operation that takes more than five seconds to complete. Note that you might have to choose one of several different platforms as your standard for estimating times of operations.

General User Feedback Guidelines

- ❑ Provide graphic feedback with appropriate desktop icon designs, by using the Indigo Magic look, by changing pointer shapes appropriately, and by highlighting selected text.
- ❑ Be sure your application displays up-to-date information—in controls and components (display the settings that correspond to the currently selected object or the current values), and in information displays (such as directory views). If the information being displayed can't be dynamically updated, choose a design that looks static.
- ❑ Provide textual message to the user through dialogs, through status areas on your primary windows, and by changing the label of your minimized window when appropriate.

Pointer Shapes and Colors Guidelines

When deciding which pointers to use in your application . . .

- ❑ Use the standard pointers when possible.
- ❑ Use the upper-left pointing arrow as a general-purpose pointer; this pointer should be red with a white outline.
- ❑ Use the upper-right pointing arrow when a menu is pulled down from a menu bar, popped up from the right mouse button, or displayed from an option menu button.
- ❑ Use the watch pointer for operations that take more than 3 seconds. (For less than 3 seconds, maintain the current pointer; for more than 5 seconds, also use a work-in-progress dialog.)
- ❑ Use the I-beam pointer to indicate that your application is in a text-editing mode, but don't use it to indicate implicit focus over a text object within a window.
- ❑ Use the question mark to indicate that the user is in context-sensitive help mode.
- ❑ Use the sighting pointer (crosshair) to make fine position selections.
- ❑ Use resize pointers to indicate positions when resizing an area.

- ❑ Use the 4-directional arrow to indicate that either a move operation or a resize operation is in progress.
- ❑ Don't use the hourglass pointer; use the watch pointer instead.
- ❑ Don't use the X pointer (it's reserved for the window manager).
- ❑ Don't assign new functionality to the standard pointer shapes; instead, design your own new pointer shape.

When designing new pointer shapes . . .

- ❑ Create a pointer shape that gives a hint about its purpose.
- ❑ Make the shape easy to see.
- ❑ Make the hotspot easy to locate.
- ❑ Avoid shapes that would create visual clutter.

Index

Numbers

4-directional arrow pointer, 232
4Dwm window manager
  See also window managers
*4Dwm*window manager, 4

A

activation
  menus, 154-156
<Alt-Insert>, 107
<Alt> key, 157
appearance, Indigo Magic. *See* Indigo Magic look
application icons (Desktop icons), 24-25
application models, 43-44, 115-120
application-modal dialogs, 209
applications
  data exchange, 101-109
  editing text, 96-98
  locations, 35
  naming, 34
  processing while minimized, 64, 67
  session management, and, 69-70
Auto Window Placement, 52, 95

B

backgrounds
  desks, and, 68
<Begin> key, 11
BMenu, 10
browse selection model, 142
BSelect, 10
BTransfer, 10
buttons, 184-185
  dialogs, and, 184
  option buttons, 153-154, 186-187
  pull-down menus, and, 167

C

<Cancel> key, 11
Cancel / Close button, 185
cascading menus, 153, 171-172
categories
  windows, 42-43, 114
checkboxes, 187-189
  Indigo Magic look, 39, 227
  menus, in, 170, 172
“Clear” option (in Edit menu), 165
“Click for Help” option (in Help menu), 79, 82
clipboard data, nonpersistent, 102
clipboard transfer model, 102-103
  primary transfer model, independence, 103, 107

"Close" option (in File menu), 161, 222
"Close" option (in window menu), 46, 47
co-primary windows, 42, 114, 121-128
  See also windows
  command line input, 125
  control areas, 124-125
  decorations, 44-46
  Help button, 84
  keyboard shortcuts, 123
  menu (in title bar), 44-46
  menu bars, 123-124
  panes, 126
  popup menus, 126
  status areas, 126
  title bar labels, 49-50
  work areas, scrollable, 124
collections, 142-144
  multiple, 145-146
color chooser, 131-132
"Color Editor" option (in Edit menu), 131, 165
color schemes, 41-42
colors
  Desktop icons, 19-22
  text fields, 192-193
command line input
  windows, 125
context-sensitive help, 79, 82
  writing help content, 89
"Context-Sensitive Help" option (in Help menu), 82
control areas
  co-primary windows, 124-125
  main windows, 124-125
control panels
  Desktop, 96-98, 229
  Language, 95
  Mouse Settings, 96, 133
  Window Settings, 52, 68
controls, 183-203
  buttons, 184-185
  checkboxes, 187-189
  dials, 202-203
  File Finder, 148-149, 199-200
  labels, 198-199
  lists, 190-192
  option buttons, 186-187
  radio buttons, 189-190
  scales, 197-198
  scrollbars, 194-197
  text fields, 192-194
  thumbwheels, 200-201
"Copy" option (in Edit menu), 102-103, 164
copying data, 102-103, 104-107
cross hair pointer, 231
<Ctrl> key, 157, 173-174
<Ctrl><Page Down> key, 196
<Ctrl><Page Up> key, 196
<Ctrl><Shift><Tab> key, 139
<Ctrl><Tab> key, 139
cursors
  location, 136
Customize control panels, 6
cut and paste. *See* data exchange
"Cut" option (in Edit menu), 102-??, 164
Cut" option (in Edit menu), ??-103
cutting data, 102-103

D

data exchange, 101-109
  <Alt-Insert>, 107
  clipboard data, nonpersistent, 102
  clipboard transfer model, 102-103
    primary transfer model, independence, 103, 107
  copying data, 102-103, 104-107
  cutting data, 102-103
  data types supported, 108
  drag and drop, 147-150

highlighting selected data, 104-107, 228
nonpersistent selection model, 107
pasting data, 102-103, 104-107
persistent always selection model, 107
primary selection, 104-107
reinstating, 107
primary transfer model, 104-107
clipboard transfer model, independence, 103, 107
selections, 142-147
data types
data exchange, supported, 108
data, selecting, 143-144
decorations
dialogs, 44-46, 211
windows, 44-46
"Delete" option (in Edit menu), 165
"Deselect All" option (in Edit menu), 165
deselecting data, 143-144
Desks, 67-68
Desks Overview, 6, 67
window titles, and, 48
Desktop
icons. *See* Desktop icons
overview, 3-11
Desktop control panel, 96-98
stderr messages, 229
Desktop icons, 4, 13-35, 227
See also minimized windows
application icons, 24-25
behavior, programming, 29-32
colors, 19-22, 27
components, 16-17
designing appearance, 14-29
file icons, 26-27
generic data file symbol, 27
generic executable symbol (magic carpet), 16, 24
Icon Catalog, 4, 32-33
icon color, 19-21
launching applications, 6-7, 29-31
orientation, 22
outline color, 21
printing files, 30
shadow color, 27
size, 19
states, 7-9, 19-20, 24-25, 227
user interaction, 6-9
Desktop services, 73-100
Auto Window Placement, 95
editor, preferred, 96-98
File Alteration Monitor (FAM), 100
Language control panel, 95
mouse
double-click speed, 96
online documentation, 94
online help, 76-94
context-sensitive, 79, 82
Help button, 80, 83-85
Help menu, 82-83
index, 80, 83
keyboard shortcuts, 81, 83
overview, 80, 83
product information, 81-82, 83
task-oriented, 80, 83
writing help content, 88-94
schemes, 95
software installation, 74-75
dialogs, 43, 114, 205-226
actions, 213-216
choosing, 213-214
default, 214-215
standard, 213
button labels, 215-216
buttons, and, 184
decorations, 44-46, 211
designing, 211-221
desks, and, 68
error, 206, 217
file selection, 206, 222-223
Help button, 84

information, 206
invoking, 221-226
menu (in title bar), 44-46, 211
modes, 209-210
placement, 211
prompt, 206, 216
pull-down menus, and, 167, 170
question, 206, 217
size, 211
title bar labels, 49-50
types, 205-209
user feedback, 229
warning, 206, 217
working, 206, 218, 224
See also windows
dials, 202-203
directories
monitoring changes, 100
Directory Views, 4
disabling menu entries, 174-175, 180
discontiguous selection model, 142
displays, updating, 228-229
documentation, online, 94
double-click speed, mouse, 96
<down arrow> key, 140, 155
drag and drop, 147-150
non-text objects, 148-149
pointers, 149
text, 149
drag icons, 149
drag state, Desktop icons, 8
drop pocket, 199
drop-accepting state, Desktop icons, 8
dynamic buttons
buttons
dynamic, 185
dynamic menu entries, 175

E

Edit menu, 162-165
editor, preferred, 96-98
enabling menu entries, 174-175, 180
<Enter> key, 11, 155
error dialogs, 206, 217
<Escape> key, 11, 155, 194, 196, 198
“Exit” option (in File menu), 162, 223
“Exit” option (in window menu), 46, 47
explicit focus, 56, 135-136

F

<F1> key, 11
<F10> key, 140, 155
FAM (File Alteration Monitor), 100
feedback. *See* user feedback
File Alteration Monitor (FAM), 100
File Finder, 148-149, 199-200
file icons (Desktop icons), 26-27
File menu, 159-162
file selection dialogs, 206, 222-223
File Typing Rules (FTRs), 29-32
files
finding, 199-200
monitoring changes, 100
printing, 30
Find an Icon tool, 4, 34-35
focus, keyboard. *See* keyboard focus
font schemes, 41-42
FTRs (File Typing Rules), 29-32

G

generic data file symbol (Desktop icons), 27
generic executable symbol (Desktop Icons), 16, 24
graphic feedback, 227-228

H

hardware description, 10-11
Help button, 80, 83-85
  writing help content, 92
help cards, 88-89
Help menu, 82-83, 166
  "Click for Help" option, 79
  "Index" option, 80
  "Keys & Shortcuts" option, 81
  "Overview" option, 80
  "Product Information" option, 81
  task options, 80
help. *See* online help
<Help> key, 11
highlighting selected data, 104-107, 145, 228
history button, 200
home button, 200-201
<Home> key, 11
hourglass pointer, 232

I

I-beam pointer, 231
Icon Catalog, 4, 32-33
icon color, Desktop icons, 19-21
icons. *See* Desktop icons, minimized windows
images
  minimized windows, 61-63
implicit focus, 56, 135
"Import" option (in File menu), 161
"Index" option (in Help menu), 80, 83
index, help topics, 80, 83
  writing help content, 91
Indigo Magic look, 38-42
  user feedback, 227
information dialogs, 206
input focus. *See* keyboard focus
installing software, 74-75
internationalization, 95

K

keyboard accelerators, 157, 173-174
keyboard description, 10-11
keyboard focus, 56-61, 135-141
  dialogs, 212
keyboard navigation, 135-140
keyboard shortcuts
  co-primary windows, 123
  help, 81, 83
    writing help content, 91
  main windows, 123
  support windows, 128
keys
  special, 10-11, 79, 82, 139-140, 155-156, 157, 172-174, 194, 196, 198
  substitutes to Motif-compliant keys, 11
"Keys & Shortcuts" option (in Help menu), 81, 83

L

labels, 198-199
  buttons, 185
  checkboxes, 188
  dialog buttons, 215-216
  dials, 202

lists, 191-192
minimized windows, 63-64, 68, 229
option buttons, 187
radio buttons, 190
scales, 197
text fields, 193
Language control panel, 95
launching applications
Desktop icons, 6-7, 29-31
<left arrow> key, 140, 155
left mouse button, 10
data exchange, 104
lists, 190-192
locate highlight, 39, 227
located state, Desktop icons, 7
location cursor, 136
locations
applications, 35
look, Indigo Magic. *See* Indigo Magic look
"Lower" option (in window menu), 47

M

magic carpet, icon component, 16, 24
main windows, 42, 114, 121-128
See also windows
command line input, 125
control areas, 124-125
decorations, 44-46
Help button, 83-84
keyboard shortcuts, 123
menu (in title bar), 44-46
menu bars, 123-124
panes, 126
popup menus, 126
status areas, 126
title bar labels, 48-49
work areas, scrollable, 124
maximize button (in title bar), 47
"Maximize" option (in window menu), 47
maximum window size, 50-51
menu bars, 151-153, 156-179
co-primary windows, 123-124
main windows, 123-124
menus
naming, 168
ordering, 170-171
<Menu> key, 11
menus, 151-181
activation, 154-156
cascading menus, 153, 171-172
entries
checkboxes, 170, 172
disabling, 174-175, 180
dynamic, 175
enabling, 174-175, 180
naming, 169-170
ordering, 170-171
radio buttons, 170, 172
toggles, 170
Indigo Magic look, 39
keyboard accelerators, 157, 173-174
menu bars, 123-124, 151-153, 156-179
mnemonics, 157, 172-173
naming, in menu bars, 168
option menus, 153-154, 186-187
naming entries, 187
ordering, in menu bars, 170-171
popup, 126, 139, 179-181
contents, 180
popup menus, 153
posted, 155
pull-down, 151-153, 156-179
buttons, and, 167
contents, 167-172
dialogs, and, 167, 170
naming entries, 169-170
naming, in menu bars, 168

ordering entries, 170-171
ordering, in menu bars, 170-171
support windows, and, 168
spring-loaded, 154
standard, 158-166
submenus, 153, 171-172
tear-off, 168
traversal, 154-156
types, 151-154
window (in title bar), 44-46, 211
messages, user, 229
middle mouse button, 10
data exchange, 104
minimize button (in title bar), 47
"Minimize" option (in window menu), 47
minimized windows, 61-66
See also Desktop icons
images, 61-63
labels, 63-64, 68, 229
processing while minimized, 64, 67
showing status, 64-65
minimum window size, 50-51
mnemonics, menus, 157, 172-173
modeless dialogs, 209
modes
dialogs, 209-210
supports windows, and, 130
monitoring, files and directories, 100
mouse
buttons, 10
data exchange, 104
description, 10
double-click speed, 96
movement, 133-134
mouse navigation, 140-141
Mouse Settings control panel, 96, 133
"Move" option (in window menu), 46
multiple collections, 145-146
"multiple document, no visible main" application model, 44, 120
"multiple document, visible main" application model, 44, 118-119
multiple selection model, 143
multiple-action pointer grab model, 56, 59-60

N

naming
applications, 34
navigation. *See* keyboard navigation, mouse navigation
neutral state, Desktop icons, 7
"New" option (in File menu), 160, 222
nonpersistent selection model, 107, 145

O

online documentation, 94
online help, 76-94
context-sensitive, 79, 82
writing help content, 89
Help button, 80, 83-85
writing help content, 92
Help menu, 82-83
index, 80, 83
writing help content, 91
keyboard shortcuts, 81, 83
writing help content, 91
overview, 80, 83
writing help content, 90
product information, 81-82, 83
task-oriented, 80, 83
writing help content, 90-91
writing help content, 88-94
"Open" option (in File menu), 160, 222

open state, Desktop icons, 8
option buttons, 153-154, 186-187
option menus, 153-154, 186-187
  naming entries, 187
Options menu, 166
outline color, Desktop icons, 21
overview help, 80, 83
  writing help content, 90
“Overview” option (in Help menu), 80, 83

P

packaging software for installation, 74-75
<Page Down> key, 196
<Page Up> key, 196
panes
  co-primary windows, 126
  main windows, 126
“Paste” option (in Edit menu), 102-103, 164, 225
pasting data, 102-103, 104-107
path navigation bar, 199
persistent always selection model, 107, 145
placement
  dialog, 211
  windows, 52
pointer, 133-134
  designing, 232
  drag and drop, 149
  shapes, state and, 134, 228, 230-232
pointer grab, 56-61
  multiple-action model, 56, 59-60
  single-action model, 56-58
popup menus, 139, 153, 179-181
  co-primary windows, 126
  contents, 180
  disabling entries, 180
  main windows, 126
posted menus, 155
preferred editor, 96-98
primary selection, 104-107, 142
  reinstating, 107
primary transfer model, 104-107
  clipboard transfer model, independence, 103, 107
primary windows. *See* main windows, co-primary windows, windows
primary-modal dialogs, 209
“Print” option (in File menu), 161
printing files
  Desktop icons, 30
processing while minimized, 64, 67
product information, 81-82, 83, 224
“Product Information” option (in Help menu), 81, 83
”Product Information” option (in Help menu), 224
“Promote” option (in Edit menu), 107, 165
prompt dialogs, 206, 216
pull-down menus, 151-153, 156-179
  buttons, and, 167
  contents, 167-172
  dialogs, and, 167, 170
  entries
    naming, 169-170
    ordering, 170-171
  naming, in menu bars, 168
  ordering, in menu bars, 170-171
  support windows, and, 168
pushbuttons. *See* buttons

Q

question dialogs, 206, 217
question mark pointer, 231

R

radio buttons, 189-190
  Indigo Magic look, 39, 227
  menus, in, 170, 172
“Raise” option (in window menu), 46, 47
range selection model, 142
“Redo” option (in Edit menu), 164
“Reopen” option (in File menu), 160, 222
resize handles (in title bar), 47
resize pointer, 231
restarting applications
  session management, 69-70
“Restore” option (in window menu), 46
<Return> key, 11
“Revert” option (in File menu), 161, 223
<right arrow> key, 140, 155
right mouse button, 10

S

“Save As” option (in File menu), 161, 222
“Save” option (in File menu), 161, 223
scales, 197-198
schemes, 41-42, 95
screen backgrounds
  desks, and, 68
scrollable lists, 190-192
scrollable work areas
  co-primary windows, 124
  main windows, 124
scrollbars, 194-197
  Indigo Magic look, 38-39, 227
Search tool, 6
secondary windows. *See* support windows, dialogs, windows
“Select All” option (in Edit menu), 165
Selected menu, 162
selected state, Desktop icons, 7
selecting data, 143-144
selections, 142-147
  collections, 142-144
    multiple, 145-146
  highlighting, 145
  models, 142-144
session management, 68-71
SGIHelp system. *See* online help
shadow color, Desktop icons, 27
<Shift><F1> key, 79, 82
<Shift><F10> key, 11, 139, 156
<Shift><Tab> key, 139
“single document, multiple primaries” application model, 43, 117-118
“single document, one primary” application model, 43, 116
single selection model, 142
single-action pointer grab model, 56-58
size
  dialogs, 211
  windows, 50-51
“Size” option (in window menu), 47
software installation, 74-75
Software Manager, 74-75
Software Packager, 74-75
spring-loaded menus, 154
standard menus, 158-166
states
  applications and sessions management, 69-70
  Desktop icons, 7-9, 19-20, 24-25, 227
  pointers, shapes and, 134, 228, 230-232
status areas, 229
  co-primary windows, 126
  main windows, 126

stderr, 229
stdout, 229
submenus, 153, 171-172
support windows, 42, 114, 128-133
  See also windows
  color chooser, 131-132
  decorations, 44-46
  design, 128-130
  desks, and, 68
  Help button, 84
  keyboard shortcuts, 128
  menu (in title bar), 44-46
  parent windows, 128
  pull-down menus, and, 168
  title bar labels, 49-50
  Tools menu, and, 128
system-modal dialogs, 209

T

<Tab> key, 139
task-oriented help, 80, 83
  writing help content, 90-91
tear-off menus, 168
text fields, 192-194
  enhanced, 192-193
thumbwheels, 200-201
title bar labels, 47-50, 68
tool palettes, 125, 184
Toolchest, 4
Tools menu, 166
  support windows, and, 128
traversal
  menus, 154-156

U

“Undo” option (in Edit menu), 164
<up arrow> key, 140, 155
updating displays, 228-229
upper-left arrow pointer, 231
upper-right arrow pointer, 231
user feedback, 227-233
  graphic, 227-228
  messages, 229
  updating displays, 228-229
user messages, 229

V

View menu, 165

W

warning dialogs, 206, 217
watch pointer, 231
widgets
  outlines in Indigo Magic look, 38
window managers
  See also 4Dwm window manager
  4Dwm, 4
Window Settings control panel, 52, 68
windows, 37-71, 113-134
  categories, 42-43, 114
  decorations, 44-46
  keyboard focus, 56-61
  menu (in title bar), 44-46
  minimized, 61-66

placement, 52
size, 50-51
title bar labels, 47-50, 68
work areas, scrollable
co-primary windows, 124
main windows, 124
working dialogs, 206, 218, 224

X

X pointer, 232

We'd Like to Hear From You

As a user of Silicon Graphics documentation, your comments are important to us. They help us to better understand your needs and to improve the quality of our documentation.

Any information that you provide will be useful. Here is a list of suggested topics to comment on:

- General impression of the document
- Omission of material that you expected to find
- Technical errors
- Relevance of the material to the job you had to do
- Quality of the printing and binding

Please include the title and part number of the document you are commenting on. The part number for this document is 007-2167-001.

Thank you!

Three Ways to Reach Us

The **postcard** opposite this page has space for your comments. Write your comments on the postage-paid card for your country, then detach and mail it. If your country is not listed, either use the international card and apply the necessary postage or use electronic mail or FAX for your reply.

If **electronic mail** is available to you, write your comments in an e-mail message and mail it to either of these addresses:

- If you are on the Internet, use this address: techpubs@sgi.com
- For UUCP mail, use this address through any backbone site: *[your_site]*!sgi!techpubs

You can forward your comments (or annotated copies of manual pages) to Technical Publications at this **FAX** number:

415 965-0964