IRIX® Interactive Desktop User Interface Guidelines

Document Number 007-2167-006

CONTRIBUTORS

Written by Jackie Neider, Deb Galdes, and Wendy Ferguson
Part III by Renate Kempf, Deb Galdes, and Mike Mohageg
Updated by Max Anderson
Illustrated by Dany Galgani, Delle Maxwell, and Doug O'Morain
Edited by Susan Wilkening
Production by Karen Jacobson
Principal architects of the IRIX Interactive Desktop Style: Deb Galdes, Delle Maxwell, Mike Mohageg, Michael Portuesi, Rob Myers, and Betsy Zeller.
Principal architects of the 3D Style: Rikk Carey, Deb Galdes, Paul Isaacs, Mike Mohageg, and Rob Myers.
Special thanks to Donna Davilla, Debbie Myers, Peter Sullivan, and Dave Ciemiewicz.

IRIX® Interactive Desktop User Interface Guidelines
Document Number 007-2167-006

Contents

List of Figures xv

List of Tables xix

About This Guide xxi
What This Guide Contains xxii
Part One: Integrating With the IRIX Interactive Desktop xxii
Part Two: Interface Components xxiii
Part Three: Designing 3D Applications xxiv
Appendix xxiv
What You Should Know Before Reading This Guide xxiv
Suggestions for Further Reading xxv
Conventions Used in This Guide xxvi
Style Guidelines xxvi
Font Conventions xxvii

PART I Integrating With the IRIX Interactive Desktop

1. Overview of the IRIX Interactive Desktop 3
Overview of the Desktop 4
How Users Interact with Desktop Icons 6
Mouse and Keyboard Hardware 9

2. Icons 11
Designing the Appearance of Icons 12
General Icon Design: Components, Size, and Colors 14
Application Icon Design 22
File Icon Design 24
Icon Appearance Design Guidelines 26

Defining the Behavior of Icons With FTRs 27
User and Icon Interaction 28
Icon Behavior Guidelines 29
Making Application Icons Accessible 30
Putting Icons Into the Icon Catalog 30
Naming and Locating Executables for the Find an Icon Tool 31
Application Icon Accessibility Guidelines 33

3. Windows in the IRIX Interactive Desktop Environment 35
The IRIX Interactive Desktop Look: Graphic Features and Schemes 36
Enhanced Graphics in the IRIX Interactive Desktop Look 36
Schemes for Colors and Fonts 39
IRIX Interactive Desktop Look Guidelines 40
Application Window Categories and Characteristics 41
Application Window Categories 41
Application Models 42
Window Decorations and the Window Menu 43
Window Title Bar 46
Window Size 49
Window Placement 51
Application Window Characteristic Guidelines 52
Keyboard Focus Across Windows 55
Single-Action Pointer Grab Model 56
Multiple-Action Pointer Grab Model 58
Guidelines for Keyboard Focus Across Windows 59
Minimized Windows 60
Choosing an Image for Your Minimized Window 60
Labeling a Minimized Window 63
Processing While Minimized 64
Using the Minimized Window to Show Status 64
Minimized Window Guidelines 64
Desks 66
Desks Guidelines 67

Session Management 67
Session Management Guidelines 71

4. IRIX Interactive Desktop Services 73
Software Installation 74
Software Installation Guideline 75
Online Help 76
Providing Help Using SGIHelp 76
Types of Online Help 77
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 87
Guidelines for Creating SGIHelp Content 90
Online Documentation 92
Desktop Variables 93
Scheme Setting 93
Auto Window Placement Setting 93
Language Setting 94
Mouse Double-Click Speed Setting 95
Editor Preference Setting 96
Desktop Variables Guidelines 98
File Alteration Monitor (FAM) 99
File Monitoring Guideline 99

5. Data Exchange on the IRIX Interactive 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 II Interface Components

6. Application Windows 113

Application Models 114

Window Types 114

Standard Application Models 115

Application Model Guidelines 119

Main and Co-Primary Windows 120

Menu Bars in Primary Windows 122

Scrollable Work Areas in Primary Windows 123

Control Areas in Primary Windows 123

Status Areas in Primary Windows 124

Splitting Primary Windows Into Panes 125

Popup Menus in Primary Windows 125

Primary Window Guidelines 125

Support Windows 127

General Support Window Design 127

A Specific Standard Support Window: The IRIX Interactive Desktop Color Chooser 129

Support Window Guidelines 131

Pointer Behavior in a Window 132

Pointer Behavior Guidelines 132

7. Focus, Selection, and Drag and Drop 133

Keyboard Focus and Navigation 133

Keyboard Focus Policy and Navigation Within a Window 134

Keyboard Focus and Navigation Guidelines 138

Selection 139

Selection Models—What Can Be Selected and How To Select It 139

Highlighting a Selection 142

Multiple Collections in One Application Window 142

Selection Guidelines 143

Drag and Drop 144
Two Models of Drag and Drop 144
Pointers for Drag Operations 147
Drag and Drop Guidelines 147

8. Menus 149
Types of Menus 149
Pull Down Menus 150
Popup Menus 151
Option Menus 151
Menu Traversal and Activation 152
Using the Mouse to Manipulate Menus 153
Using the Keyboard to Manipulate Menus 154
Menu Traversal and Activation Guidelines 154
The Menu Bar and Pull-Down Menus 155
Standard Menus 157
What to Put in the Pull-Down Menus 165
Choosing Mnemonics 171
Choosing Keyboard Accelerators 171
Disabling Menu Entries 173
Dynamic Menu Entries 174
Pull-Down Menu Guidelines 174
Popup Menus 177
What to Put in Popup Menus 178
Disabling Popup Menu Entries 179
Popup Menu Guidelines 179

9. Controls 181
Pushbuttons 182
Pushbutton Guidelines 182
Option Buttons 184
Option Button Guidelines 184
Checkboxes 186
Checkbox Guidelines 186

Radio Buttons 187
Radio Button Guidelines 188
LED Indicators 189
LED Button Guidelines 189
Lists 190
List Guidelines 191
Text Fields 192
Text Field Guidelines 193
Scrollbars 194
Scrollbar Guidelines 195
IRIX Interactive Desktop Scales 196
IRIX Interactive Desktop Scale Guidelines 197
Labels 198
Label Guidelines 198
File Finder 199
File Finder Guidelines 199
Thumbwheels 200
Thumbwheel Guidelines 200
Dials 201
Dial Guidelines 201

10. Dialogs 203
Types and Modes of Dialogs 203
Dialog Modes 207
Guidelines for Using the Various Types and Modes of Dialogs 208
Designing Dialogs 209
Decorations, Initial State, and Layout of Dialogs 209
Standard Dialog Actions 211
Content of Specific Types of Dialogs 214
Guidelines for Designing Dialogs 216
Invoking Dialogs 218
Invoking Dialogs When Manipulating Files 219
Other Situations for Invoking Dialogs 221
Guidelines for Invoking Dialogs 221

11. User Feedback 223
Types of Feedback 223
Providing Graphic Feedback 224
Keeping Information Up to Date 224
Providing Messages to the User 225
General User Feedback Guidelines 226
Pointer Shapes and Colors 226
Standard Pointer Shapes and Colors 226
Designing New Pointer Shapes 228
Pointer Shapes and Colors Guidelines 229

PART III 3D Style Guidelines

12. Introduction to 3D Style Guidelines 233
Making 3D Functionality Available 234
Designing Mouse Input for 3D Applications 234
Using Modifier Keys in 3D Applications 235
Basic 3D Interface Design Guidelines 236
Pointer Shapes for 3D Functions 236
Pointer Feedback Guidelines for 3D Applications 238
Resizing the 3D Viewing Window 238
Guidelines for Resizing Windows in 3D Applications 238

13. Interactive Viewing of 3D Objects 239
Introduction to 3D Viewing 240
3D Viewing Functions 241
Inspection Functions for 3D Viewing 243
Navigation Functions for 3D Viewing 251
Guidelines for 3D Viewing Functions 257
3D Viewing Interface Trade-Offs 259
Viewing and Editing in 3D Applications 259
Single-Viewport and Multi-Viewport Viewing in 3D Applications 262
3D Viewing Performance and Scene Fidelity 263
3D Viewing Trade-Offs and Related Guidelines 265

14. Selection in 3D Applications 267
3D Selection Concepts and Models 267
The Object-Action Paradigm in 3D Applications 268
Direct Selection in 3D Applications 268
Indirect Selection in 3D Applications 269
3D Selection Models 270
Selection in Hierarchies of Objects 271
3D Selection Design Guidelines 271
Selection Feedback for 3D Objects 273
Bounding Box Selection Feedback 273
Manipulator Selection Feedback 274
Highlight Selection Feedback 275
3D Selection Feedback Design Guidelines 275
Lead Objects in 3D Applications 276
Lead Object When Selecting Multiple Objects 276
Lead Object During Grouping and Ungrouping 277
Lead Object Design Guidelines for 3D Applications 279

15. Manipulating 3D Objects 281
Basic 3D Manipulation Techniques 282
Phases of 3D Manipulation 282
3D Manipulation Feedback 284
Free and Constrained 3D Manipulation 285
Manipulator Presentation and Selection 285
Basic 3D Manipulation Guidelines 286
Translating 3D Objects 287
3D Translation Basics 288
Simple (Planar) 3D Translation 289
Constrained 3D Translation 290
3D Translation User Interface Guidelines 294

Rotating 3D Objects 294
3D Rotation Basics 295
Constrained 3D Rotation 296
Free 3D Rotation 297
3D Rotation User Interface Guidelines 299
Scaling 3D Objects 300
3D Scaling Basics 300
Uniform 3D Scaling 301
Axial 3D Scaling (Stretching) 303
Scaling Around the Opposite Corner or Side 306
3D Scaling User Interface Guidelines 309
Changing the Center of Rotation and Scaling for 3D Objects 310
Guidelines for Changing the Center of 3D Rotation 313
Object Manipulation for Multiple Selected 3D Objects 313
Translation of Multiple Selected 3D Objects 314
Rotation of Multiple Selected 3D Objects 314
Scaling of Multiple Selected 3D Objects 315
Guidelines for Manipulating More Than One 3D Object 315

A. Summary of Guidelines 317
Guidelines for Integrating With the IRIX Interactive Desktop 319
Icon Appearance Design Guidelines 319
Icon Behavior Guidelines 321
Application Icon Accessibility Guidelines 322
IRIX Interactive Desktop Look Guidelines 322
Application Window Characteristic Guidelines 322
Guidelines for Keyboard Focus Across Windows 325
Minimized Window Guidelines 326
Desks Guidelines 327
Session Management Guidelines 327
Software Installation Guideline 328
Guidelines for Designing Online Help 328
Guidelines for Creating SGIHelp Content 330
Desktop Variables Guidelines 332
File Monitoring Guideline 333
Data Exchange Guidelines 333

Interface Component Guidelines 334
Application Model Guidelines 334
Primary Window Guidelines 334
Support Window Guidelines 336
Pointer Behavior Guidelines 336
Keyboard Focus and Navigation Guidelines 337
Selection Guidelines 337
Drag and Drop Guidelines 338
Menu Traversal and Activation Guidelines 338
Pull-Down Menu Guidelines 339
Popup Menu Guidelines 342
Tab Panel Guidelines 343
Pushbutton Guidelines 343
Option Button Guidelines 345
Checkbox Guidelines 346
Radio Button Guidelines 347
LED Button Guidelines 348
List Guidelines 348
Text Field Guidelines 349
Scrollbar Guidelines 350
IRIX Interactive Desktop Scale Guidelines 352
Label Guidelines 352
File Finder Guidelines 353
Thumbwheel Guidelines 354
Dial Guidelines 354
Guidelines for Using the Various Types and Modes of Dialogs 355
Guidelines for Designing Dialogs 356
Guidelines for Invoking Dialogs 358
General User Feedback Guidelines 360
Pointer Shapes and Colors Guidelines 360

3D Style Guidelines 361
Basic 3D Interface Design Guidelines 361
Pointer Feedback Guidelines for 3D Applications 362
Guidelines for Resizing Windows in 3D Applications 362
Guidelines for 3D Viewing Functions 363
3D Viewing Trade-Offs and Related Guidelines 365
3D Selection Design Guidelines 367
3D Selection Feedback Design Guidelines 368
Lead Object Design Guidelines for 3D Applications 369
Basic 3D Manipulation Guidelines 369
3D Translation User Interface Guidelines 371
3D Rotation User Interface Guidelines 372
3D Scaling User Interface Guidelines 372
Guidelines for Changing the Center of 3D Rotation 373
Guidelines for Manipulating More Than One 3D Object 374

List of Figures

Figure 1-1 IRIX Interactive Desktop 5

Figure 1-2 Icon States: User Actions and Effects 8

Figure 2-1 Web Tools Icons 12

Figure 2-2 File Icons 13

Figure 2-3 Application and File Icon Components 15

Figure 2-4 IconSmith Examples 16

Figure 2-5 IconSmith Color Palette 18

Figure 2-6 Icon States and Effects on Color 19

Figure 2-7 Potential Icon Background Areas and Colors 21

Figure 2-8 Application Icon in Running and Not Running States 22

Figure 2-9 Examples of Application Icons That Could be Improved 24

Figure 2-10 Examples of Standard File Icons 25

Figure 2-11 Examples of File Icons for Application-Specific File Formats 26

Figure 2-12 Launch Dialog Box 29

Figure 2-13 Icon Catalog 31

Figure 2-14 The Find an Icon Tool 32

Figure 3-1 Examples of Graphic Modifications in the IRIX Interactive Desktop Look 38

Figure 3-2 Scheme Setting Control Panel 39

Figure 3-3 Features of a Typical Main Primary Window 44

Figure 3-4 Title Bar Label Appearing in Desks Overview 47

Figure 3-5 Labels for Main Window Title Bars 48

Figure 3-6 Default Maximum and Minimum Window Size Examples 50

Figure 3-7 Setting Auto Window Placement 51

Figure 3-8 Single-Action Pointer Grab Example: Capture by Sweeping 57

Figure 3-9 Multiple-Action Pointer Grab Example 59

Figure 3-10 Minimized Window Example: Good User Association With Application 60
Figure 3-11 Minimized Window Examples 61
Figure 3-12 Minimized Window Example: Incorrect Design 62
Figure 3-13 Minimized Window Example: Design That's Too Literal 62
Figure 3-14 Minimized Window Example: Indicating Status With the Label 64
Figure 3-15 Desks Overview Window 66
Figure 3-16 Setting Session Management 68
Figure 3-17 Setting Auto Window Placement 69
Figure 4-1 The IRIX Interactive Desktop 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 84
Figure 4-6 Task-Oriented Help Example 89
Figure 4-7 Language Control Panel 94
Figure 4-8 Mouse Settings Control Panel 95
Figure 4-9 Desktop Settings Control Panel 96
Figure 4-10 Selecting Preferred Editor in MediaMail 97
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 118
Figure 6-5 "Multiple Document, No Visible Main" Application Model 119
Figure 6-6 Basic Primary Window 120
Figure 6-7 Primary Windows With Tool Palettes 121
Figure 6-8 Primary Window With Two Panes 122
Figure 6-9 The IRIS Showcase Align Gizmo 128
Figure 6-10 The IRIX Interactive Desktop Color Chooser 129

Figure 7-1 Location Cursor Example 134
Figure 7-2 Components and Fields 136
Figure 7-3 File Finder Component 146
Figure 8-1 Menu Bar 150
Figure 8-2 Pull-Down Menu 150
Figure 8-3 Cascading Menu 151
Figure 8-4 Popup Menu 151
Figure 8-5 Option Menu Button 151
Figure 8-6 An Open Option Menu 152
Figure 8-7 Elements of a Pull-Down Menu 155
Figure 8-8 Standard Menus for Menu Bars 157
Figure 8-9 The Standard File Menu 158
Figure 8-10 The Standard Edit Menu 161
Figure 8-11 Radio buttons 170
Figure 8-12 Checkboxes 170
Figure 8-13 Popup Menu 178
Figure 9-1 Pushbuttons 182
Figure 9-2 Option Button and Option Menu 184
Figure 9-3 Checkboxes 186
Figure 9-4 Radio Buttons 187
Figure 9-5 LED Button 189
Figure 9-6 List 190
Figure 9-7 Scrollbar 194
Figure 9-8 IRIX Interactive Desktop Scale 196
Figure 9-9 The File Finder 199
Figure 9-10 Thumbwheel 200
Figure 9-11 Dials 201
Figure 10-1 Sample Prompt, Error, Warning, Working, Question and Information Dialogs 205
Figure 10-2 The IRIX Interactive Desktop File Selection Dialog 206
Figure 10-3 Warning Dialog Layout 210
Figure 10-4 Warning Dialog With Save, Discard, and Cancel Buttons 213
Figure 10-5 Error Dialog With Specific Entity 215

Figure 10-6 Working Dialog with IRIX Interactive Desktop Scale 215
Figure 10-7 Warning Dialog for Overwriting a File 220
Figure 10-8 Warning Dialog for Reverting to Previous Version 220
Figure 10-9 Product Information Dialog 221
Figure 13-1 The Camera Analogy in 3D Viewing 240
Figure 13-2 Schematic Illustration of Tumbling (Implementation Perspective) 245
Figure 13-3 Schematic Illustration of Dollying (Implementation Perspective) 246
Figure 13-4 Schematic Illustration of Zooming (Implementation Perspective) 247
Figure 13-5 Schematic Illustration of Panning (User Drags Right) 248
Figure 13-6 Simple Example of Seeking to Door 250
Figure 13-7 Schematic illustration of Roaming (Implementation Perspective) 254
Figure 13-8 Schematic Illustration of Tilting (Implementation Perspective). 255
Figure 13-9 Schematic Illustration of Sidling (User Drags Left) 256
Figure 13-10 Application With Viewing Controls 261
Figure 14-1 Two Objects with Bounding Box Feedback 273
Figure 14-2 Object With Manipulator 274
Figure 14-3 Selection Feedback: Lead Object Has Manipulator. 277
Figure 14-4 Grouped Collection of Objects With Manipulator. 278
Figure 14-5 Ungrouped Collection of Objects (Knight Is Lead Object) 278
Figure 15-1 Object With Translation Manipulator (Bounding Box) 288
Figure 15-2 Simple Planar Translation Sequence 290
Figure 15-3 Constrained Translation Along One Axis of the Selected Plane 292
Figure 15-4 Constrained Translation Along the Normal to the Selected Plane 293
Figure 15-5 Object With Rotation Manipulator (Rotation Handles) 295
Figure 15-6 Constrained Rotation Sequence 297
Figure 15-7 Free Rotation Sequence 298
Figure 15-8 Object With Scaling Manipulator (Scaling Handles) 301
Figure 15-9 Uniform Scaling Sequence 303
Figure 15-10 Axial Scaling Sequence 305
Figure 15-11 Uniform Scaling Around a Corner 307
Figure 15-12 Axial Scaling Around a Side 308
Figure 15-13 Changing the Center of Rotation and Scaling 311
Figure 15-14 Changing the Center of Rotation and Scaling Along An Axis 312

List of Tables

Table 1-1 Mouse Button Names and Functions 9

Table 1-2 Keyboard Substitutes 10

Table 2-1 User/Icon Interactions and Expected Behavior 28

Table 3-1 Window Decorations and Window Menu Entries by Window Category 45

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

Table 7-1 Selection Actions and Results 140

Table 8-1 File Menu Entries 159

Table 8-2 Standard Edit Menu Entries 162

Table 8-3 Keyboard Accelerators 172

Table 10-1 Types of Dialogs, Their Modality, and When to Use Them 204

Table 11-1 Standard Pointer Shapes and Colors 227

Table 12-1 Use of Modifier Keys in a 3D Application 235

Table 12-2 Pointers for 3D Functionality 237

Table 13-1 3D Viewing Functions and User Interface 242

Table 13-2 Overview of Inspection Viewing Functions 244

Table 13-3 Overview of Navigation Viewing Functions 252

Table 14-1 3D Selection Actions and Results 270

Table 15-1 Overview of Manipulation Techniques 285

Table 15-2 Phases of Planar Translation 289

Table 15-3 Phases of Translation Along One Axis of the Selected Plane 291

Table 15-4 Phases of Translation Along the Normal to the Selected Plane 293

Table 15-5 Phases of Constrained Rotation 296

Table 15-6 Phases of Free Rotation 298

Table 15-7 Phases of Uniform Scaling 302

Table 15-8 Phases of Axial Scaling (Stretching) 304

Table 15-9 Phases of Uniform Scaling Around a Corner 306

Table 15-10 Phases of Axial Scaling Around a Side 307

Table 15-11 Phases of Changing the Center of Rotation and Scaling Along a Plane 310

Table 15-12 Phases of Changing the Center of Rotation and Scaling Along an Axis 311

About This Guide

This guide is written for developers of software products used on Silicon Graphics workstations, including software engineers, graphical user interface designers, usability specialists, and others involved in the design process. It contains guidelines to help you design products that are consistent with other applications and that integrate seamlessly into the IRIX Interactive 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.

Using this guide, you learn how to design user interfaces for Silicon Graphics applications. There are also specific examples of what is and isn't appropriate and why. Note that the guidelines discussed in this guide 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.

Developers using this guide are expected to be 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 IRIX Interactive Desktop 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 IRIX Interactive Desktop 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 guide will help ensure that your software product provides all of the functionality and ease of use designed into the IRIX Interactive Desktop.

One focus of this guide is how your application should look and feel on the IRIX Interactive 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 *IRIX Interactive Desktop Integration Guide*.

What This Guide Contains

This guide has three 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. This guide is also available on the World Wide Web.

Part One: Integrating With the IRIX Interactive Desktop

The first part of this guide describes how users expect to be able to interact with your application from the IRIX Interactive Desktop; it contains these chapters:

- Chapter 1, "Overview of the IRIX Interactive Desktop," sets the context for Part One; it gives you an overview of the desktop environment in which users 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 IRIX Interactive Desktop Environment," defines the various categories of windows and describes the IRIX Interactive Desktop look for your application's windows. This look is an enhanced version of IRIS IM and includes pre-packaged color and font schemes. The 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, "IRIX Interactive Desktop Services," explains how your application can take advantage of several services provided by the IRIX Interactive Desktop, such as Software Manager, SGIHelp, the IRIS InSight online documentation viewer, and global desktop settings.
- Chapter 5, "Data Exchange on the IRIX Interactive Desktop," describes the data transfer models that your application should support. It also lists the data types supported for data exchange in the IRIX Interactive Desktop 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 these 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 IRIX Interactive Desktop 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 discusses 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.

Part Three: Designing 3D Applications

The third part of this guide provides user interface guidelines specifically for 3D applications. These guidelines address design issues that are not relevant in a 2D context. In addition, 3D applications should also follow the guidelines described in the first two parts of this guide. Part Three contains these chapters:

- Chapter 12, "Introduction to 3D Style Guidelines," provides an overview of general 3D user interface design guidelines.
- Chapter 13, "Interactive Viewing of 3D Objects," first discusses inspection and navigation, which are the two approaches to viewing, and the associated viewing functions users expect. It then considers some trade-offs designers commonly have to make: Making both viewing and editing available in an intuitive way, deciding on single-viewport or multi-viewport viewing, and dealing with scene fidelity vs. speed.
- Chapter 14, "Selection in 3D Applications," provides recommendations for designing the selection interface of a 3D application. It builds on the selection models discussed in Chapter 7, "Focus, Selection, and Drag and Drop".
- Chapter 15, "Manipulating 3D Objects," provides recommendations for designing the translation, rotation, and scaling user interface of a 3D application.

Appendix

Appendix A, "Summary of Guidelines," provides a checklist that you can use to determine whether your product follows the IRIX Interactive Desktop user interface guidelines. This checklist contains all of the individual guidelines that appear throughout the guide.

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 enhances and clarifies information presented in that manual.

Suggestions for Further Reading

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

- *IRIX Interactive 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 may want to take a look at *Desktop Users Guide* (also an online manual), which explains how users interact with the IRIX Interactive Desktop.

If you're new to creating 3D applications, either in general or on Silicon Graphics systems, you'll find the following books useful:

- Foley, J.D., A. van Dam, S. Feiner, and J.F. Hughes. *Computer Graphics Principles and Practice*, second edition. Reading, Massachusetts: Addison-Wesley Publishing Co., 1990. (This book is the classic introductory text to 3D programming.)
- Kalwick, David. *3D Graphics Tips, Tricks, and Techniques*. Chestnut Hill, Massachusetts: Academic Press, Inc. 1996. (This book defines the many 3D graphics terms and explains how to create 3D graphics using existing software applications rather than writing your own code.)
- Wernecke, Josie, *The Inventor Mentor*. Reading, Massachusetts: Addison-Wesley Publishing Co., 1994. (This book provides basic information on programming with Open Inventor. It includes detailed program examples in C++ and describes key aspects of the Open Inventor toolkit, including its 3D scene database.)

Finally, you may want to refer to additional references that describe 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 interface 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. *Designing the User Interface: Strategies for Effective Human-Computer Interaction*, second edition. Reading, Massachusetts: Addison-Wesley Publishing Co., 1992. (This book presents general user interface guidelines.)

Conventions Used in This Guide

In this guide, the following conventions 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; a complete list is provided in Appendix A, "Summary of Guidelines."

Font Conventions

The typographical conventions used in this guide include:

- **Bold text** indicates that a term is a data type, a keyword, an X or IRIS IM widget name, a function, a command-line option, or an X resource.
- *Italic text* indicates that a term is a file name, a button name, a variable, an IRIX command, or a document title.
- `Screen type` indicates screen displays and code examples.
- **`Bold screen type`** indicates user input and nonprinting keyboard keys.
- "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 IRIX Interactive Desktop

Chapter 1

Overview of the IRIX Interactive Desktop

Chapter 2

Icons

Chapter 3

Windows in the IRIX Interactive Desktop Environment

Chapter 4

IRIX Interactive Desktop Services

Chapter 5

Data Exchange on the IRIX Interactive Desktop

Overview of the IRIX Interactive Desktop

The IRIX Interactive 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 IRIX Interactive 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 IRIX Interactive Desktop." For details on how users interact with the IRIX Interactive Desktop, see the online *Desktop Users Guide*.

This chapter covers the following topics:

- "Overview of the Desktop" briefly describes the major elements of the IRIX Interactive 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

Figure 1-1 shows an example of the IRIX Interactive Desktop, with the following tools running:

- Welcome page design and navigation (the center, blue Netscape window). This window can be launched from the Welcome_to_SGI icon.
- Toolchest. The Toolchest serves as the primary access point for desktop user interfaces. For example, users can access interfaces for everyday tasks such as customizing their desktop, accessing applications and Web tools, backing up and restoring their files, and finding information (help) on a variety of desktop topics. By default, the Toolchest is located in the upper left corner.
- New background patterns. The background in the new screensnap is one of the new patterns available from the Background Setting panel. The Background Setting panel itself is visible in the screensnap (lower right), previewing the new background pattern called "Citrus Citrus." The background panel can be launched from the Toolchest by selecting "Desktop > Customize > Background."
- System Manager (center window). This window has a new design but the same functionality as before. System Manager can be launched from the Toolchest by selecting "System > System Manager."
- Desks Overview window. With the Desks Overview window, users can switch from one "desk," or group of applications, to another. When your application appears in a desk other than the one currently in use, it is in a state similar to the minimized state. You need to be careful about what processes your application runs while in a minimized state. The Desks Overview window is shown in the lower left corner.
- Icon Catalog. Users can access icons from the different pages in the Icon Catalog. Some of the pages are: Applications, Demos, Desktop Tools, Media Tools, and Web Tools. Since the Icon Catalog is one of the first places users look when they need to find an application, you should add your product's icons to this catalog. By default, the Icon Catalog is below the Toolchest.
- File Manager window. From this window, you can view file contents (example shown is a model of an X29 fighter jet, partly obscured by the System Manager window). The File Manager window is shown at the left, above the Desks Overview window.

In addition, the "noiconlogin-mode" image has been updated to show the new SGI logo. The EZsetup account login and web page design have also been updated. Other elements that remain from the previous desktop design include the File QuickFind (shown in the upper right), minimized windows (five across on the left side under the Toolchest), and device icons along the right side. These are just a few examples of the kinds of things you

will need to consider to integrate your application into the Desktop Environment. This book provides style guidelines while the “IRIX Interactive Desktop Integration Guide” gives you complete and detailed instructions for integration. For the best results, use both books together.

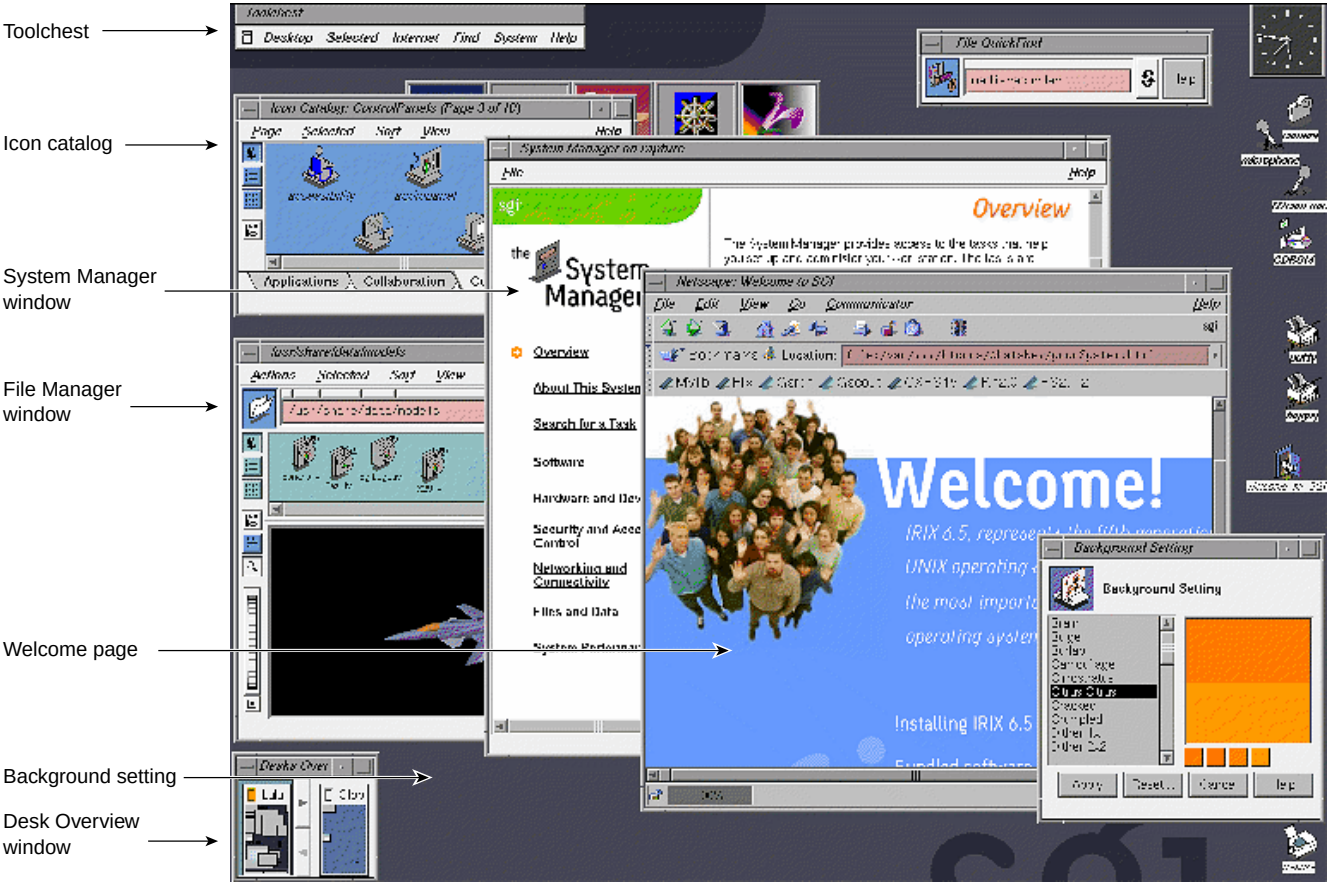


Figure 1-1 IRIX Interactive Desktop

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 *Desktop Users Guide*.

How Users Interact with Desktop Icons

Users interact with icons on the IRIX Interactive 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 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 IRIX Interactive Desktop 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 is not 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 IRIX Interactive Desktop 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.

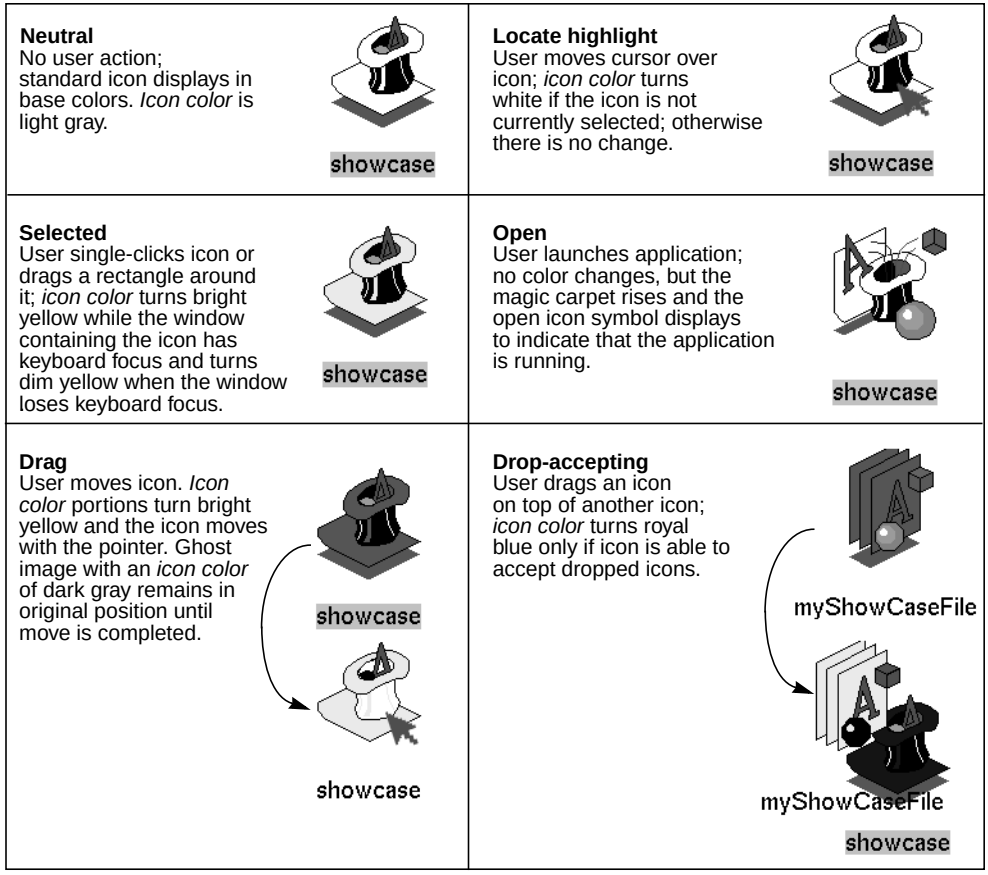


Figure 1-2 Icon States: User Actions and Effects

Mouse and Keyboard Hardware

The Silicon Graphics three-button mouse 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>

Icons

The IRIX Interactive 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.

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 IRIX Interactive 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

This section discusses icon design. Topics include:

- "General Icon Design: Components, Size, and Colors"
- "Application Icon Design"
- "File Icon Design"

Your application's desktop icons can appear in any of several places on the IRIX Interactive 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. Figure 2-2 shows various data file icons that can appear on a user's desktop.

For each application, 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.)

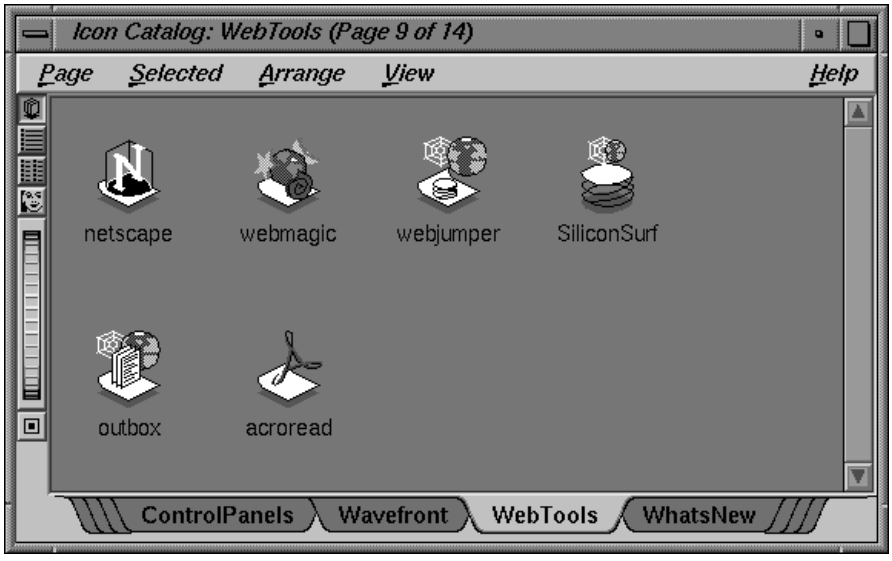


Figure 2-1 Web Tools Icons

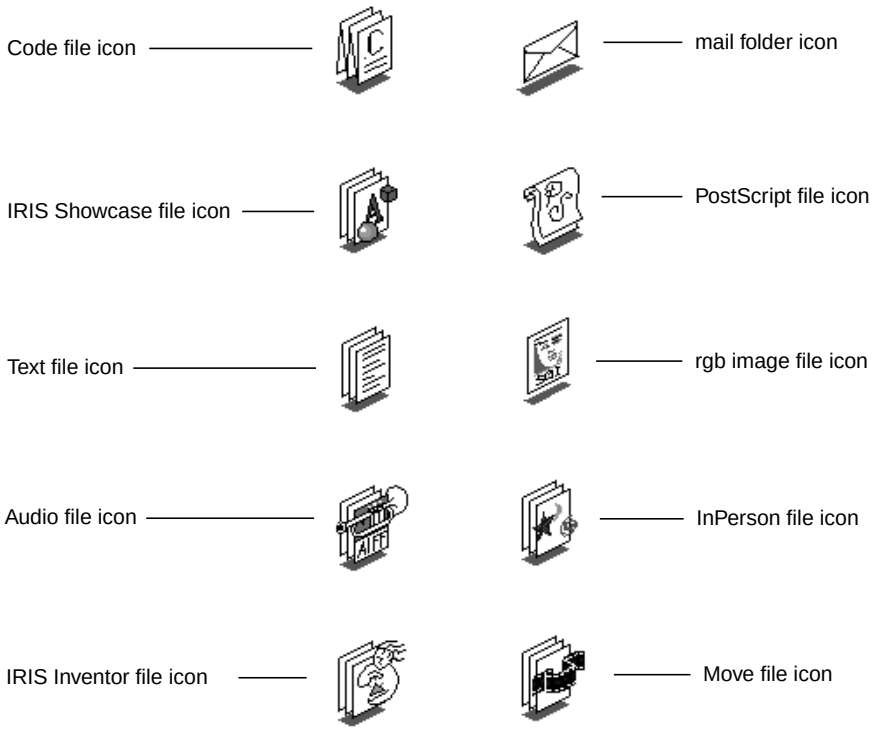


Figure 2-2 File Icons

To make your application and file icons stand out in a group of icons, be sure 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

Design your application and file icons so that they:

- convey your product identity
- have the correct components and thus are recognizable as desktop icons
- are identifiable when zoomed to the minimum and maximum sizes allowed by the IRIX Interactive Desktop
- use an effective color scheme, which allows the icon to reflect its state

Your primary goal is to differentiate your icons from those of other products and to make the design fairly simple because desktop icons are small when minimized.

Icon Components

Application and file icons consist of several components (as shown in Figure 2-3):

- 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
- 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.

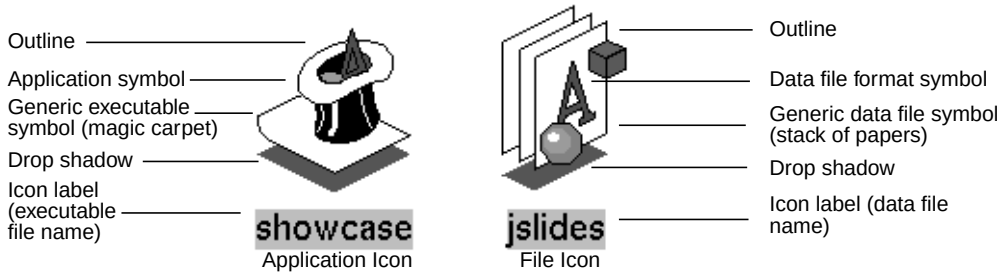
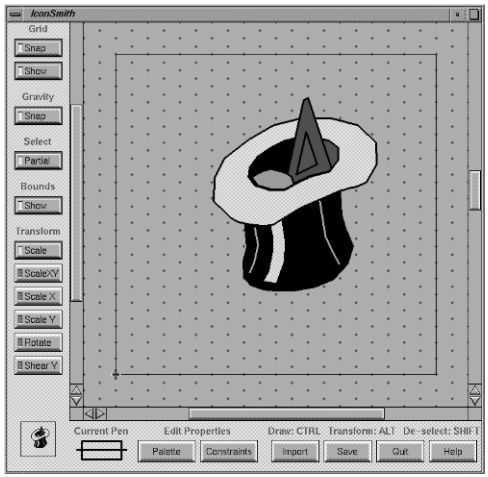


Figure 2-3 Application and File Icon Components

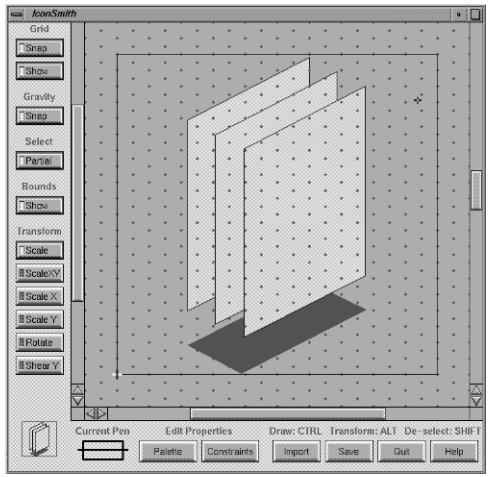
You need to create the symbols for your application icon and for any unique file types your application creates. Create the symbols with the IconSmith drawing tool, which is described in Chapter 12, "Using IconSmith," in the *IRIX Interactive Desktop Integration Guide*. IconSmith provides a drawing grid 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 can 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.

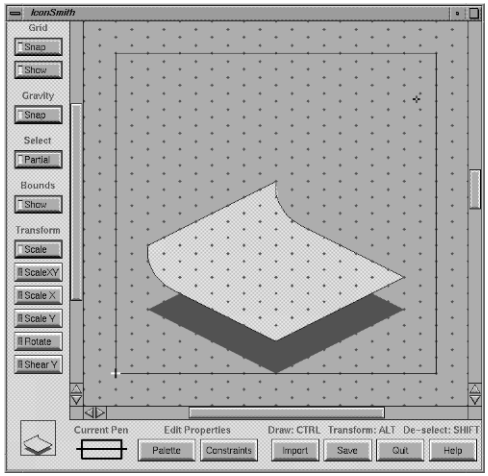
Showcase application symbol (not running)



Data file symbol



Magic carpet symbol (not running)



Magic carpet symbol (running)

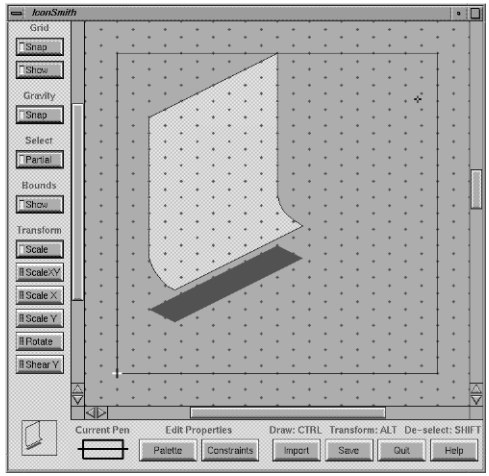


Figure 2-4 IconSmith Examples

Icon Size

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 dynamically 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 IRIX Interactive Desktop," for explanations of icon states from the user's point of view.)

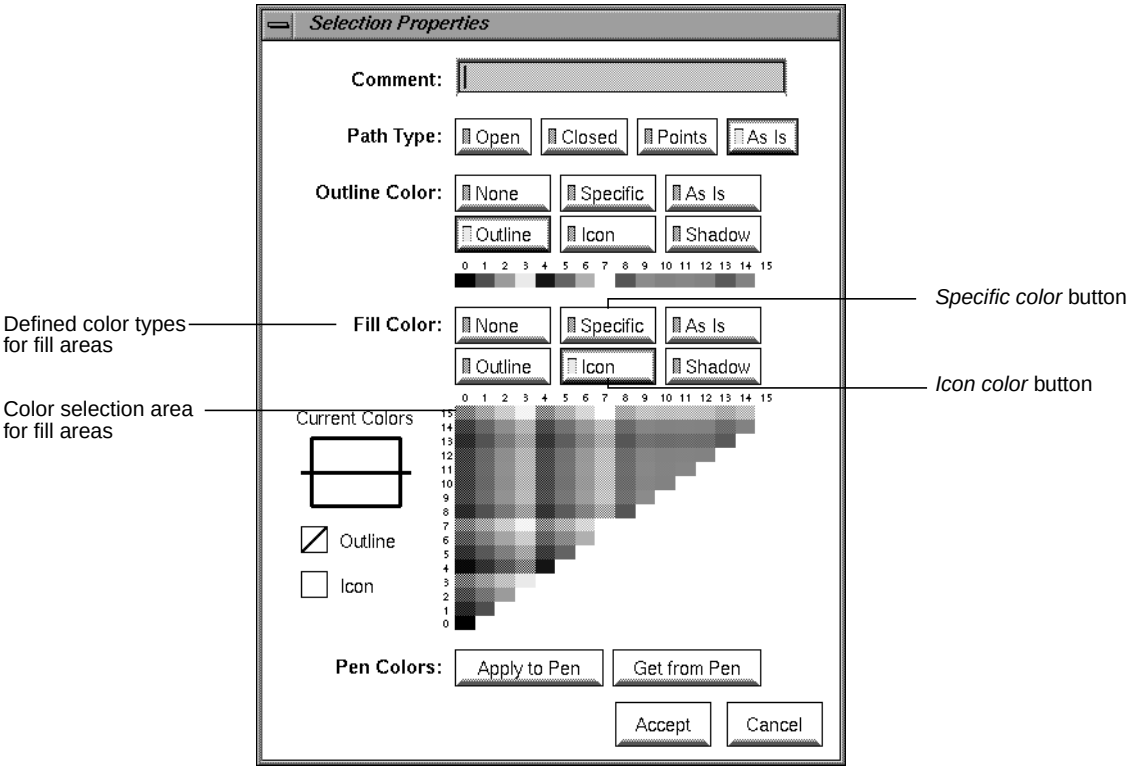


Figure 2-5 IconSmith Color Palette

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 IRIX Interactive 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 IRIX Interactive Desktop to indicate state—bright yellow, dim yellow, royal blue, pure white, and light gray.

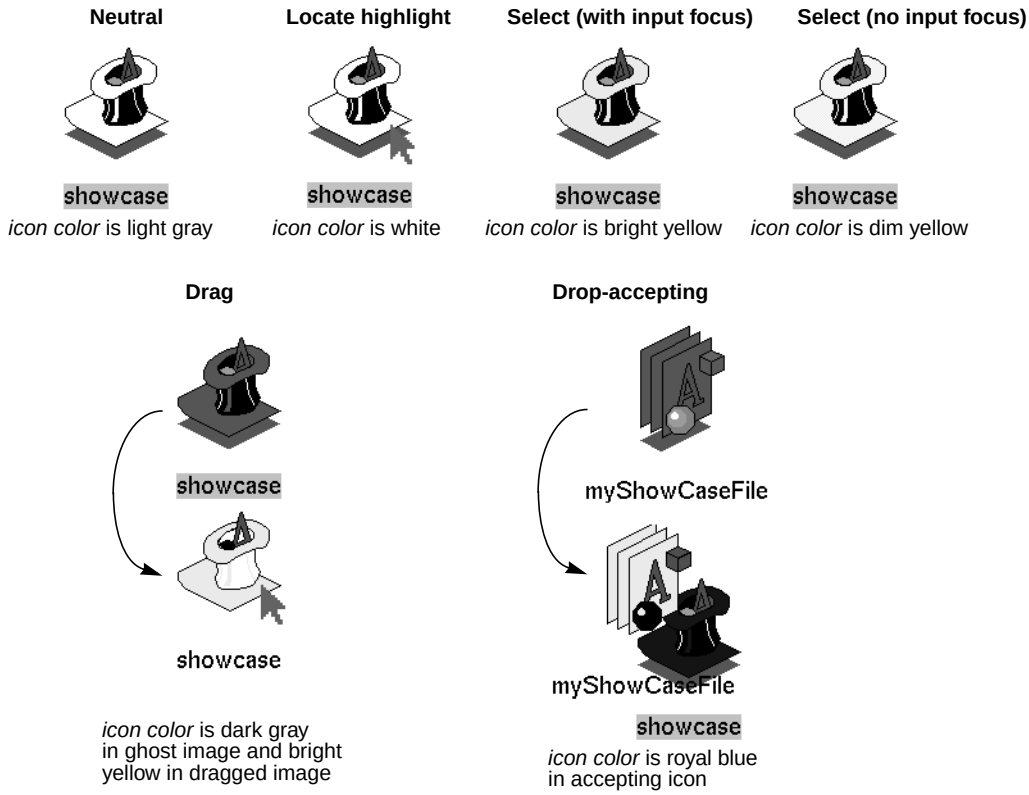


Figure 2-6 Icon States and Effects on Color

You can increase your icon's visibility 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 IRIX Interactive Desktop.") To increase visibility:

- 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 IRIX Interactive Desktop 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

  See */usr/lib/X11/schemes/Base/OzSpec* for colors used by these tools:

  ```
  *ozDirPanelColor: AlternateBackground4
  *ozCltnPanelColor: AlternateBackground5
  *ozDropPocketColor: AlternateBackground5
  ```

  See */usr/lib/X11/schemes/Base/BaseColorPalette* for definitions of these colors:

  ```
  #define AlternateBackground4 #729c9c
  #define AlternateBackground5 #5680ab
  #define AlternateBackground6 #b6b6aa
  ```

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 IRIX Interactive 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.

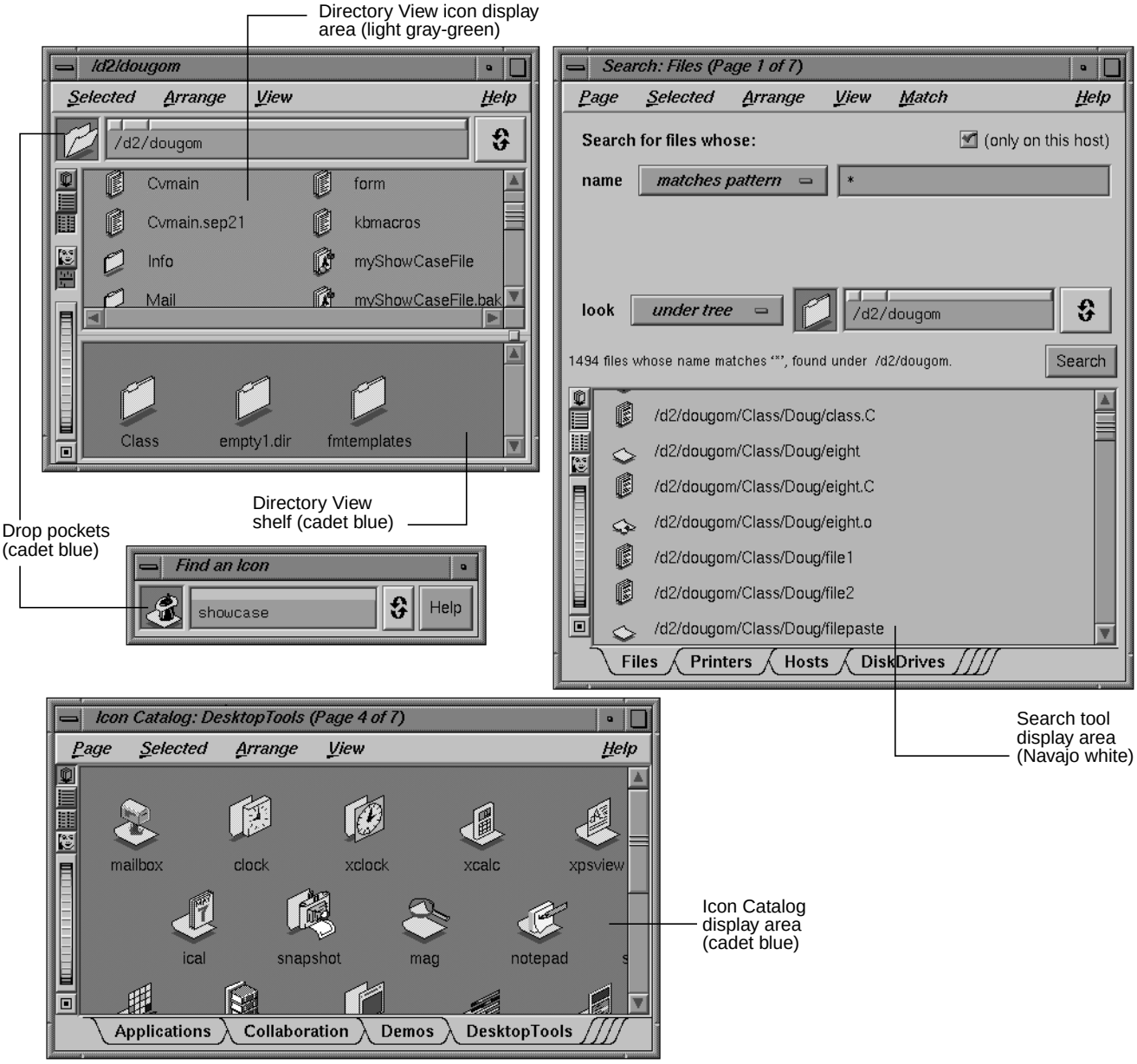


Figure 2-7 Potential Icon Background Areas and Colors

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 data 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 data 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 *IRIX Interactive Desktop Integration Guide*.

As shown in Figure 2-8, the application icon uses 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.

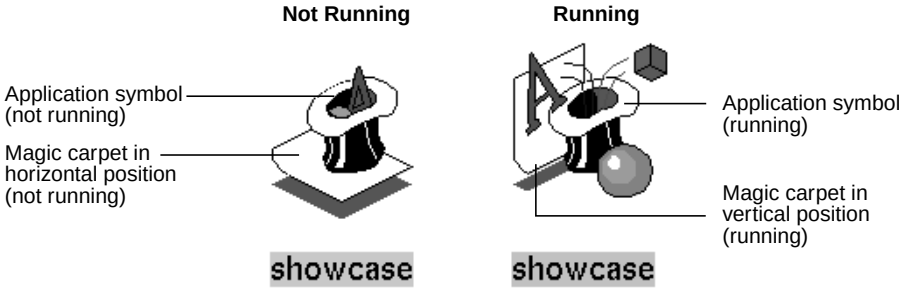


Figure 2-8 Application Icon in Running and Not Running States

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 *IRIX Interactive 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 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 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.

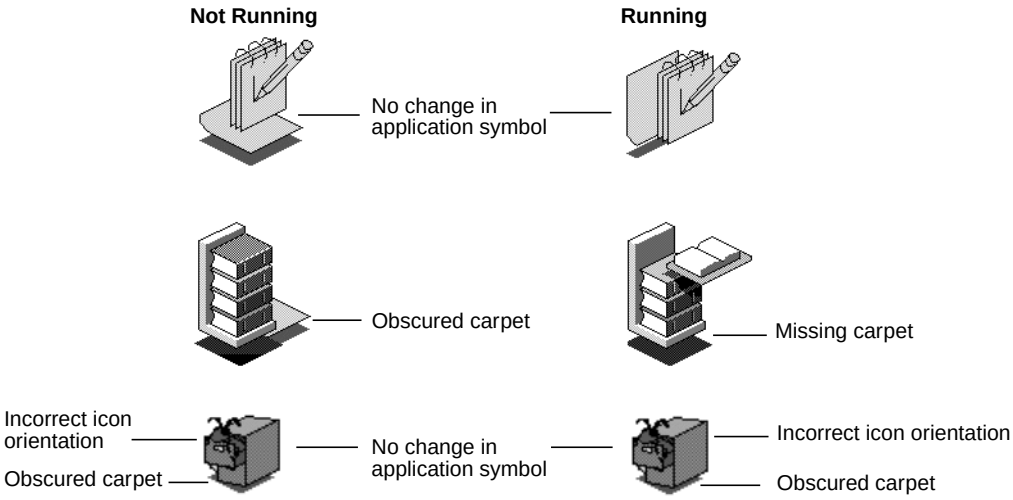


Figure 2-9 Examples of Application Icons That Could be Improved

File Icon Design

If your application creates files, those files need to be associated with *data file icons*. Silicon Graphics defines a set of standard data formats and provides associated data 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 *IRIX Interactive Desktop Integration Guide*.) If your application saves its data in any of these standard formats, it will automatically use the appropriate data 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. 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."

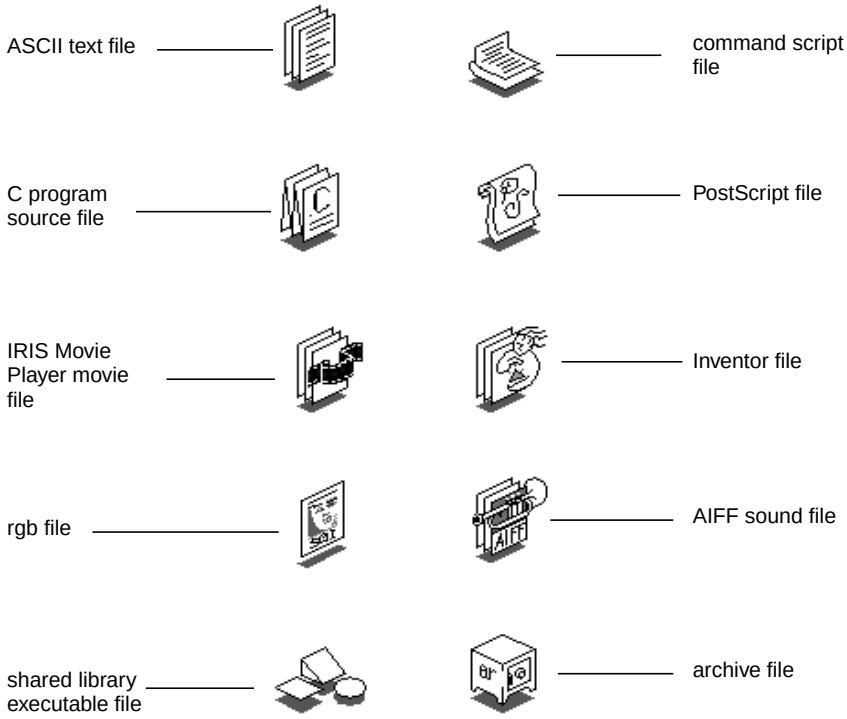


Figure 2-10 Examples of Standard File Icons

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 designs fairly simple because desktop icons can be displayed in 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.
- ❑ Use 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 IRIX Interactive 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 moving the magic carpet from a horizontal (not running) to vertical (running) position, or by providing two different application symbols and moving the magic carpet from a horizontal to vertical position. Remember that your application symbols should resemble a progressive animation when viewed in succession.
- ❑ Make sure that your application symbols do not completely obscure the magic carpet in either its horizontal or vertical position.
- ❑ Application icons that have two separate symbols for the not running and running states should make sure that the main part of the symbol remains in the same location during both states, so that the symbol appears stationary to the user.

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

Define the behavior of icons on the IRIX Interactive 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:

- "User and Icon Interaction," which explains how users can interact with icons, how an FTR supports these interactions (in brief).
- "Icon Appearance Design Guidelines," which lists the minimal set of standard behaviors your application and file icons should support.

User and Icon Interaction

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 *IRIX Interactive 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.

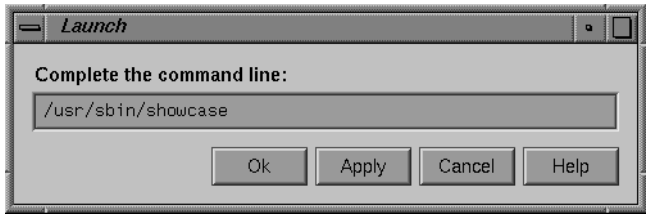


Figure 2-12 Launch Dialog Box

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. This section describes:

- "Putting Icons Into the Icon Catalog"
- "Naming and Locating Executables for 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 11, "Creating Desktop Icons: An Overview," in the *IRIX Interactive 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. Some of the Icon Catalog pages are listed below along with typical applications that can appear on them:

- Application page—multimedia presentation application (*showcase*), text editor (*jot*), electronic mail program *(MediaMail*)
- Collaboration page—desktop conferencing application (*inperson*), 3D model annotator (*annotator*)
- 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*)
- Web Tools page—Netscape Navigator (netscape), WebMagic (webmagic), WebSpace Navigator, WebJumper (webjumper), OutBox/WebServer, DynaWeb Server, What's New, TarDist

In most cases your application icon will 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.

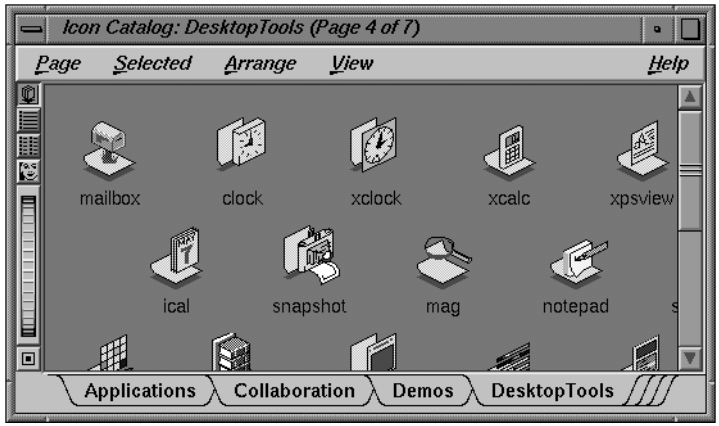


Figure 2-13 Icon Catalog

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 the icon.

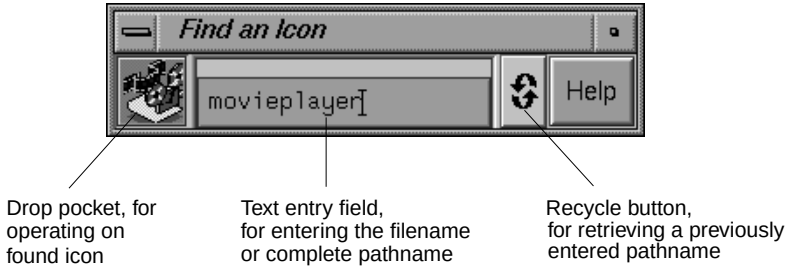


Figure 2-14 The Find an Icon Tool

Since users may be guessing at the most likely name for the application's executable, 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*. Don't use an abbreviation of your product name.
- 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, put your executable—or a link to it, which is preferable—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. Don't use an abbreviation of the product name.
- ❑ 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.

Windows in the IRIX Interactive Desktop Environment

When users run your application on the IRIX Interactive 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 IRIX Interactive Desktop Look: Graphic Features and Schemes" discusses general characteristics of windows, including the IRIX Interactive Desktop 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 IRIX Interactive 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 IRIX Interactive Desktop Look: Graphic Features and Schemes

When using the IRIS IM user interface toolkit, you can choose one of two different appearances for your application:

- the IRIX Interactive Desktop look (preferred by Silicon Graphics)
- the basic OSF/Motif look

The IRIX Interactive Desktop look provides an attractive, 3D look for your application and the default colors and fonts used by the IRIX Interactive Desktop. The two components to the IRIX Interactive Desktop look are described below:

- "Enhanced Graphics in the IRIX Interactive Desktop Look"
- "Schemes for Colors and Fonts"

Enhanced Graphics in the IRIX Interactive Desktop Look

The IRIX Interactive Desktop look contains a number of graphic modifications made to the standard IRIS IM interface. These modifications improve the appearance and ease of use of applications. They have no impact on the component layout and require minimal work on the part of the developer. Some of the differences between the IRIX Interactive Desktop look and the standard IRIS IM look appear in Figure 3-1. In comparison, the IRIX Interactive Desktop 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.)
- 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.
- 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).

- Enhances scrollbars by providing a grip on the slider used in scrollbars and a temporarily 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.
- Uses a stroked underline to indicate mnemonics in menus simply to give the look of an application some pizazz (see Figure 3-1D).
- Provides a more consolidated treatment for composite objects. For example, the IRIX Interactive Desktop 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).
- 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.
- 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 closely grouped dialog buttons that are right justified. This style makes the buttons easier to read, and the close grouping may reduce mouse motion.

For details on how to obtain the graphic enhancements of the IRIX Interactive Desktop look, see Chapter 2, "Getting the IRIX Interactive Desktop Look," in the *IRIX Interactive 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."

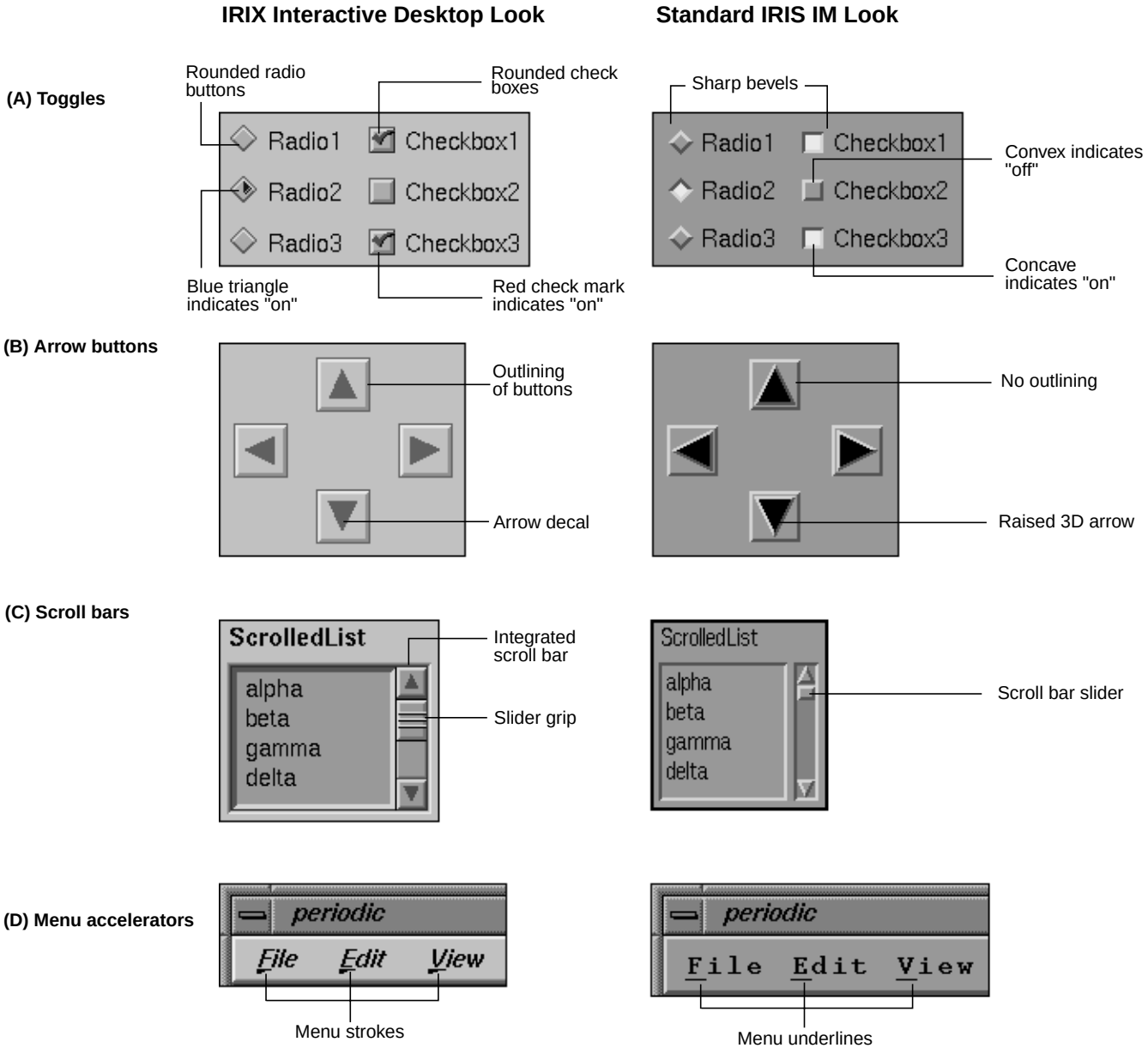


Figure 3-1 Examples of Graphic Modifications in the IRIX Interactive Desktop Look

Schemes for Colors and Fonts

In addition to the graphic modifications described in the previous section, "Enhanced Graphics in the IRIX Interactive Desktop Look," the IRIX Interactive Desktop 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.)

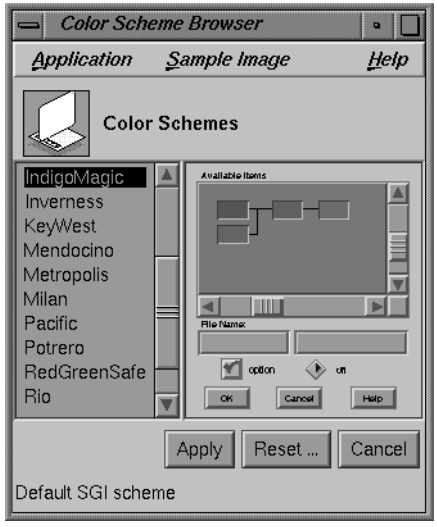


Figure 3-2 Scheme Setting Control Panel

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 IRIX Interactive Desktop comes up in the IRIX Interactive Desktop 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. Don't use schemes 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 *IRIX Interactive Desktop Integration Guide*.

IRIX Interactive Desktop Look Guidelines

When designing the look for your application . . .

- ❑ Use the IRIX Interactive Desktop 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 IRIX Interactive Desktop environment; presents four models of applications using these window types; and lists the decorations, Window menus, and behavior required for each window type. Specifically, this section covers:

- "Application Window Categories"
- "Application Models"
- "Window Decorations and the Window Menu"
- "Window Title Bar"
- "Window Size"
- "Window Placement"

Application Window Categories

The *OSF/Motif Style Guide* refers to two categories of windows: *primary windows* and *secondary windows*. The IRIX Interactive Desktop 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 *IRIX Interactive 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 IRIX Interactive 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*.

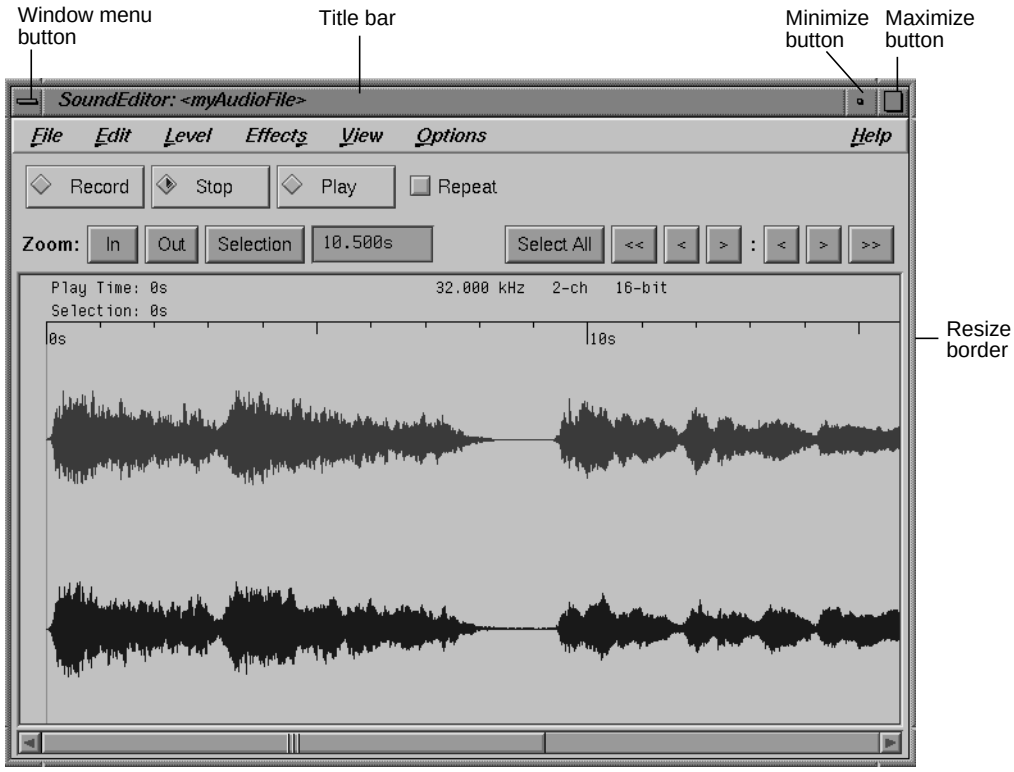


Figure 3-3 Features of a Typical Main Primary Window

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 *IRIX Interactive 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; do not assign them 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
"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.

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

Window Decorations and Window Menu Entries	Main Windows	Co-Primary Windows	Support Windows	Dialogs
"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 IRIX Interactive 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. 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 *IRIX Interactive 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.

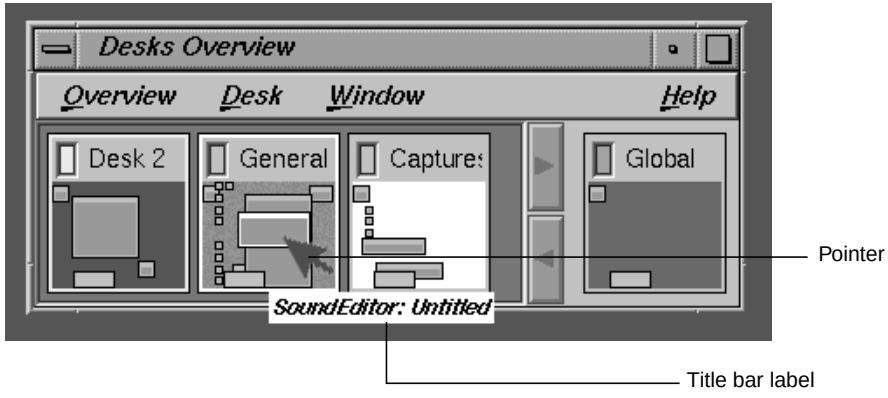


Figure 3-4 Title Bar Label Appearing in Desks Overview

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. Unless you have a real need and enough room, don't include the full pathname in the title bar. Note that if you do include a filename in the label, you need to update the label whenever the file changes.
- 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; make that information available from the "Product Information" entry in the Help menu (see "Types of Online Help" in Chapter 4 for more information).

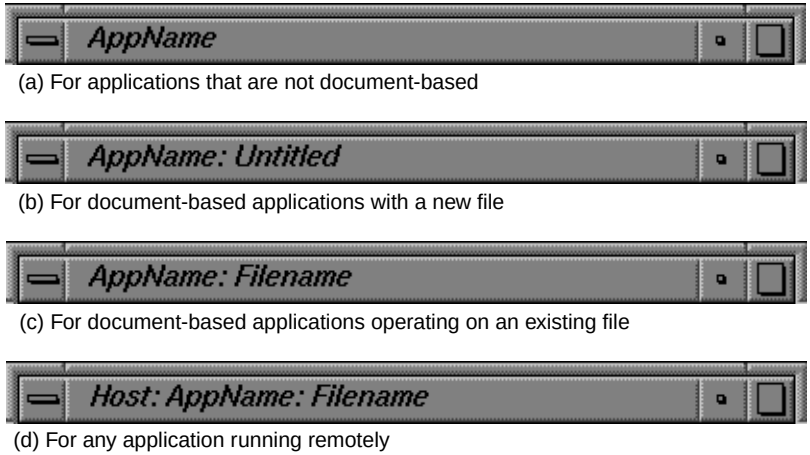


Figure 3-5 Labels for Main Window Title Bars

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). Unless you have a real need and enough room, don't include the full pathname in the title bar.

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 *IRIX Interactive 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."

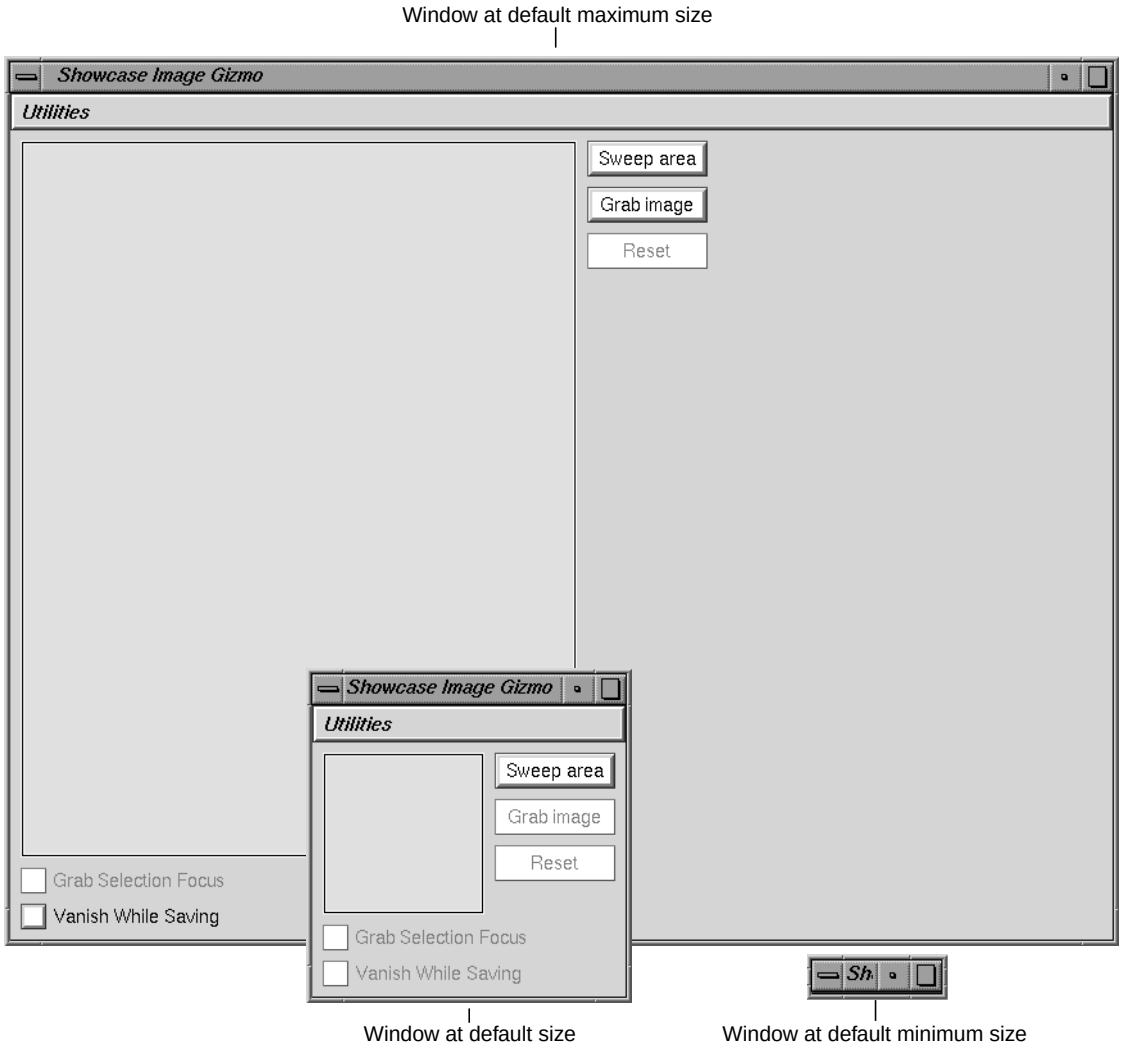


Figure 3-6 Default Maximum and Minimum Window Size Examples

Window Placement

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.

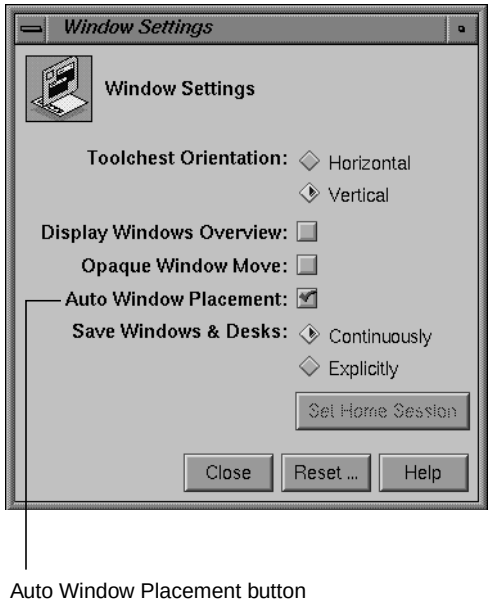


Figure 3-7 Setting Auto Window Placement

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 *IRIX Interactive 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 ...* .
- ❑ Don't include full pathnames unless that information is required by users to distinguish one window from another. For remote applications, don't include domain information.
- ❑ Don't use the version number in the title bar; make that information available from the "Product Information" entry in the Help menu.
- ❑ 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. Don't use the full pathname unless that information is required for users to distinguish one window from another.
- ❑ 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.

For windows without title bars . . .

- ❑ Display the "Exit" option with the right mouse button.
- ❑ Allow users to resize the window with the left mouse button.

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*, the two types of keyboard focus (also referred to as input focus) include:

- *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 IRIX Interactive 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:

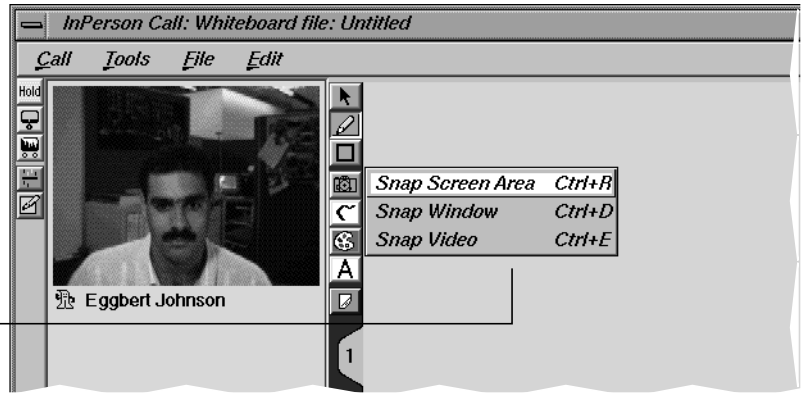
- *single-action*, which permits the user only one action to capture the data before returning to implicit focus
- *multiple-action*, which lets the user perform multiple actions while in pointer grab mode

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.)

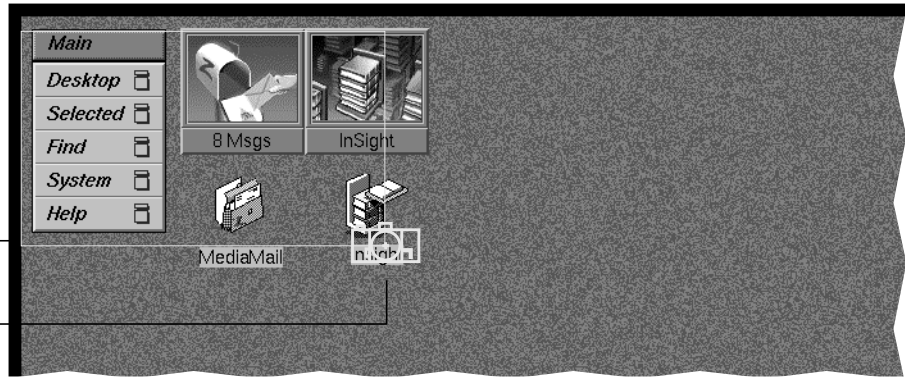
Single-Action Pointer Grab Model

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.

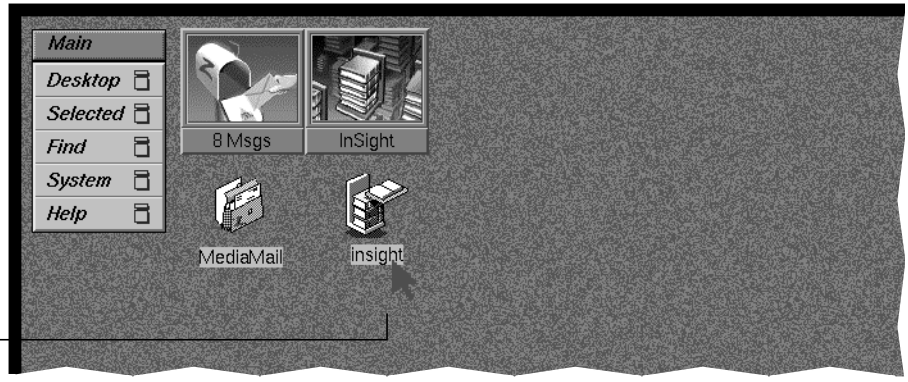


Figure 3-8 Single-Action Pointer Grab Example: Capture by Sweeping

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.

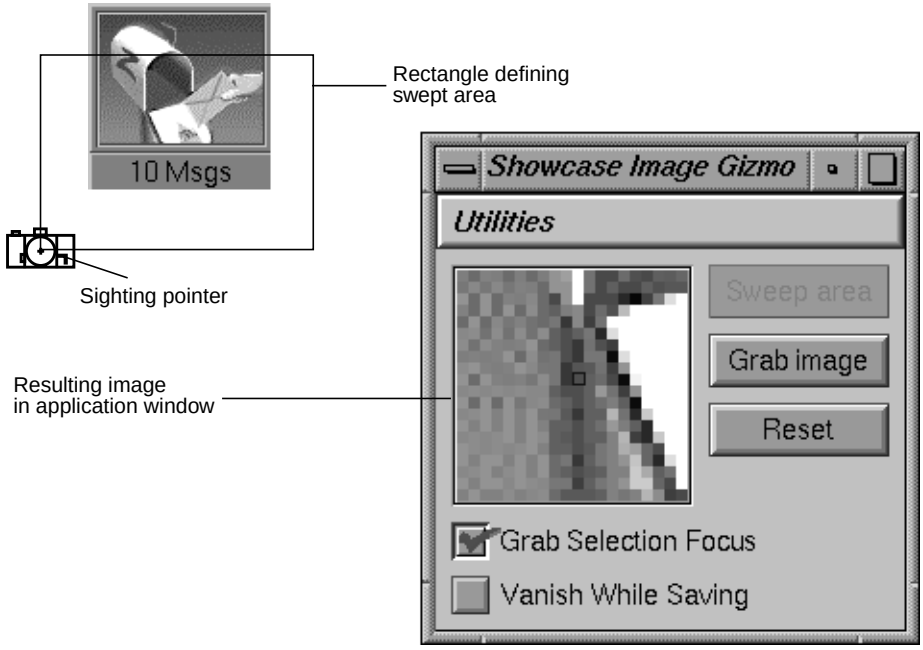


Figure 3-9 Multiple-Action Pointer Grab Example

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 area for other uses. On the IRIX Interactive 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. This section describes:

- "Choosing an Image for Your Minimized Window"
- "Labeling a Minimized Window"
- "Processing While Minimized"
- "Using the Minimized Window to Show Status"

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, provide separate images for all primary windows. If your application fits one of the multiple document models, then provide one image for the main window and a second image for the co-primary windows.

When choosing a minimized window image, use:

- 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.

Examples of minimized windows appear in Figure 3-11.

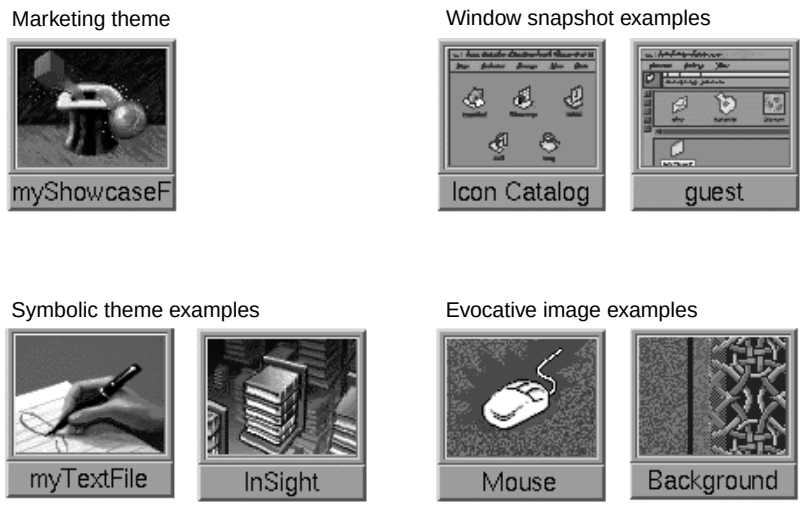


Figure 3-11 Minimized Window Examples

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 can 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 can 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.

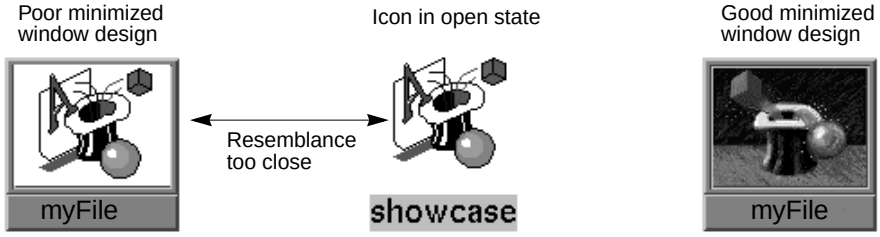


Figure 3-12 Minimized Window Example: Incorrect Design

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, with an English acronym "RIP," that represents dead bugs. This may not be readily apparent to some non-English speaking users.

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

For information on creating and implementing minimized window images, see Chapter 6, "Customizing Your Application's Minimized Windows," in the *IRIX Interactive 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 *IRIX Interactive 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 to use the minimized window label 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 *IRIX Interactive Desktop Integration Guide*.

Minimized Window Guidelines

When designing images for your minimized primary windows . . .

- ❑ Use a color image rather than the standard two-color Motif 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 (the label may be truncated).
- ❑ 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 IRIX Interactive Desktop provides users with a handy tool called Desks Overview 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.

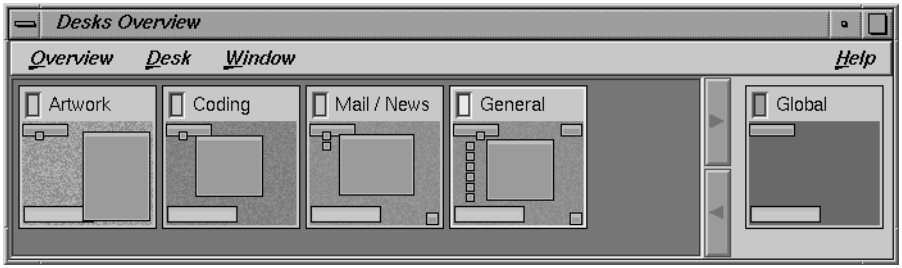


Figure 3-15 Desks Overview Window

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.)
- 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.

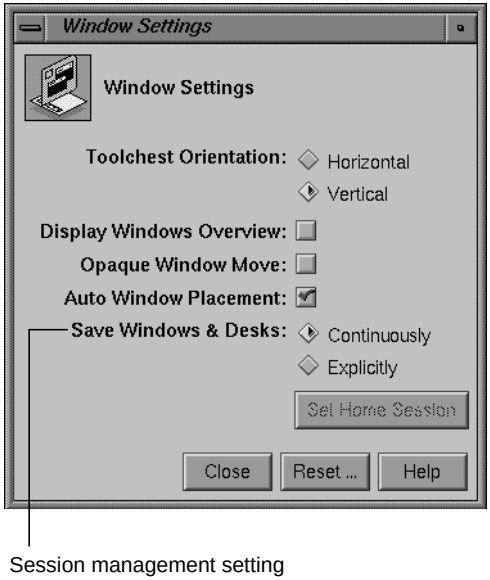


Figure 3-16 Setting Session Management

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 *IRIX Interactive Desktop Integration Guide*. The following scenario outlines the process.

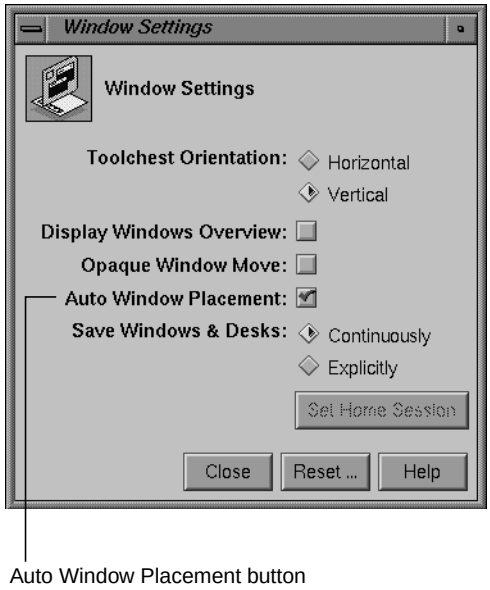


Figure 3-17 Setting Auto Window Placement

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 *IRIX Interactive 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.

IRIX Interactive Desktop Services

IRIX Interactive Desktop 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 IRIX Interactive Desktop environment. This chapter covers the following topics:

- "Software Installation" describes how users install, remove, and upgrade applications using Software Manager, an IRIX Interactive 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*.

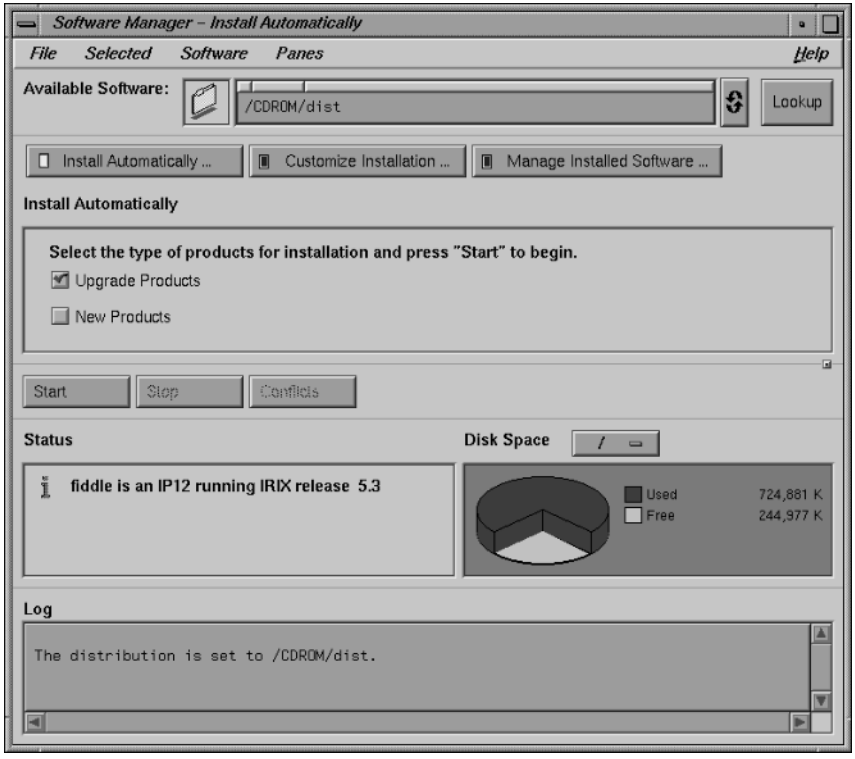


Figure 4-1 The IRIX Interactive Desktop Software Manager

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 to 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 can'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 IRIX Interactive Desktop utility.

Online Help

Your application should provide online help. In the IRIX Interactive Desktop 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). This section covers the following topics:

- "Providing Help Using SGIHelp"
- "Types of Online Help"
- "Providing Help through a Help Menu"
- "Providing Help Through a Help Button"
- "Writing Online Help Content for SGIHelp"

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 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.
- You don't have to create and maintain your own help system. All of the help capabilities that your application supports (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 *IRIX Interactive 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 consists of 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.

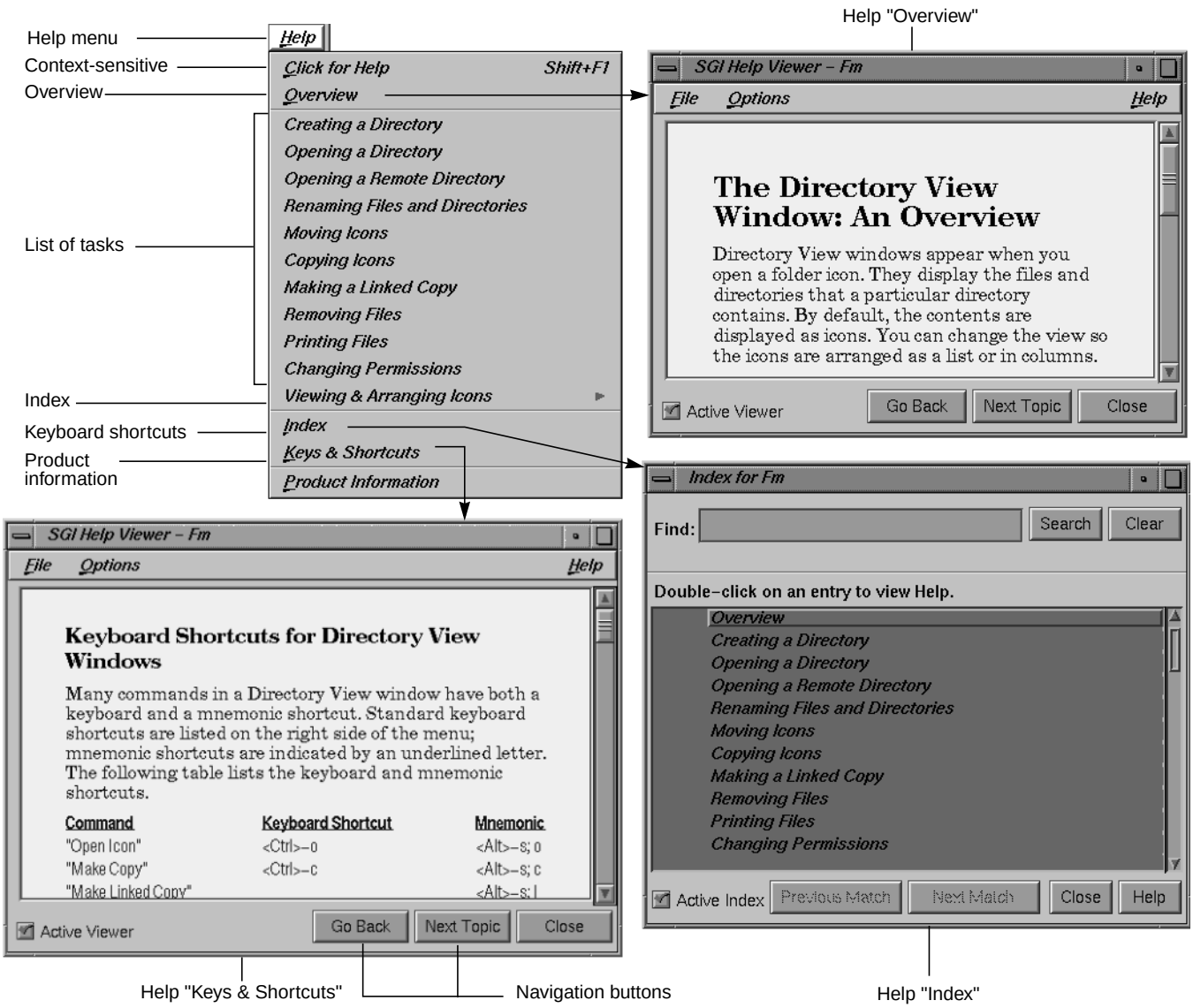


Figure 4-2 Typical Help Menu and Related Windows

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 include cross-references to task-oriented help topics which describe tasks which use this area.

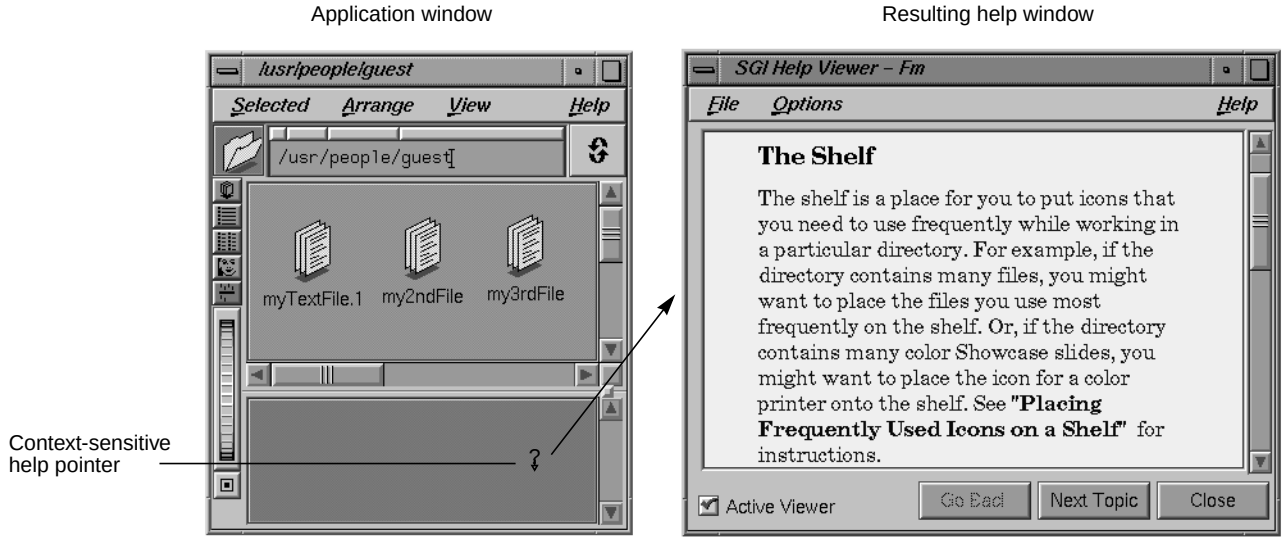


Figure 4-3 Context-Sensitive Help Example

Overview Information

Overview information answers the question "What does this application or window do?" Provide overview help information in all main windows 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.

Provide task-oriented help in all windows. 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 appear 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 identifies the application and version number, but it can also include general product information such as copyright and trademarking, licensing, and customer support access. Provide access to this information from all main windows. 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.

Always provide product information using an Information dialog rather than using 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.

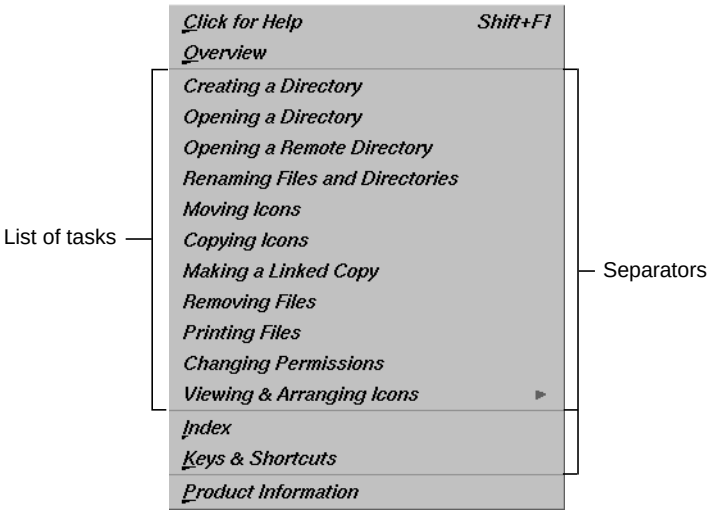


Figure 4-4 Typical Help Menu

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. Don't use mnemonics or keyboard accelerators in these entries.
- "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, 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.

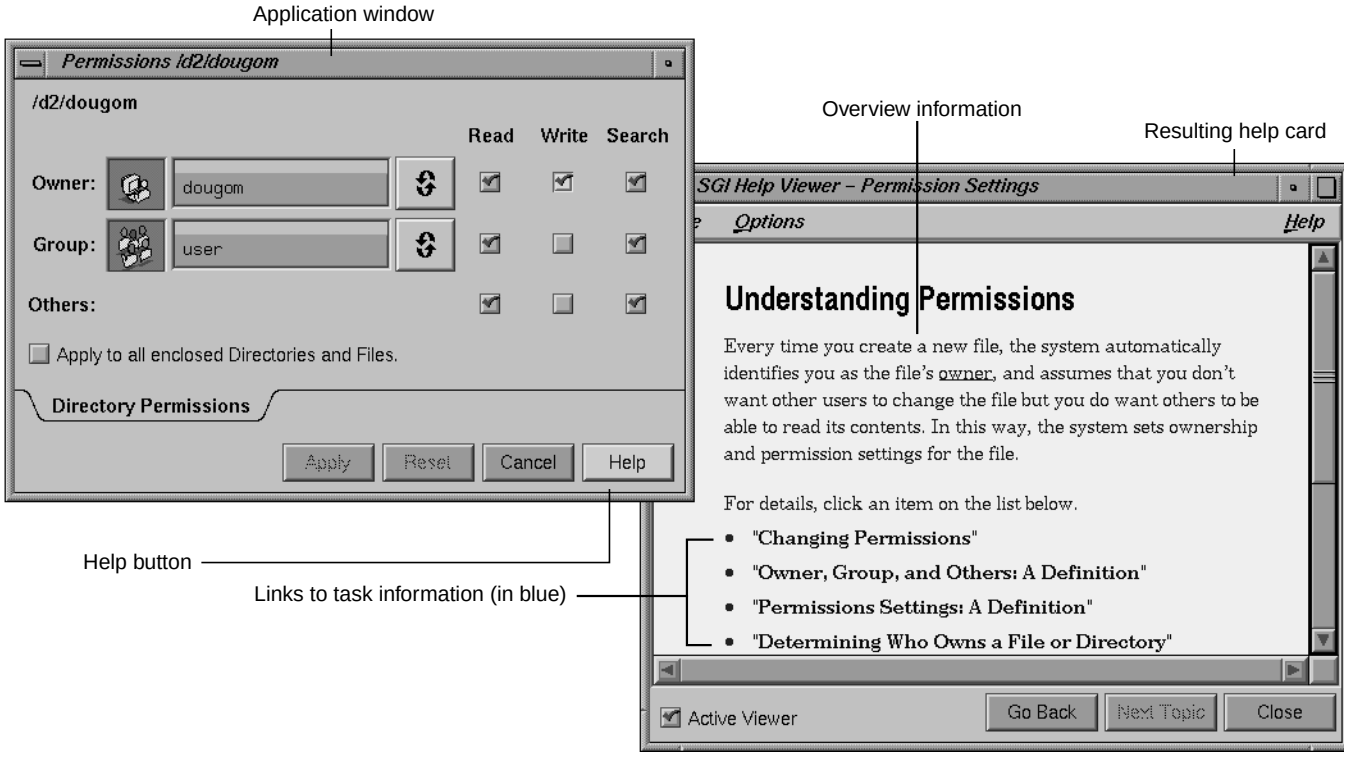


Figure 4-5 Help Button Example

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 IRIX Interactive Desktop 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 pieces (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 to minimize scrolling. The content shouldn't assume that users have read other help topics. 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. Make each title as descriptive as possible to tell the reader what's contained in the specific help card. The titles should 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.

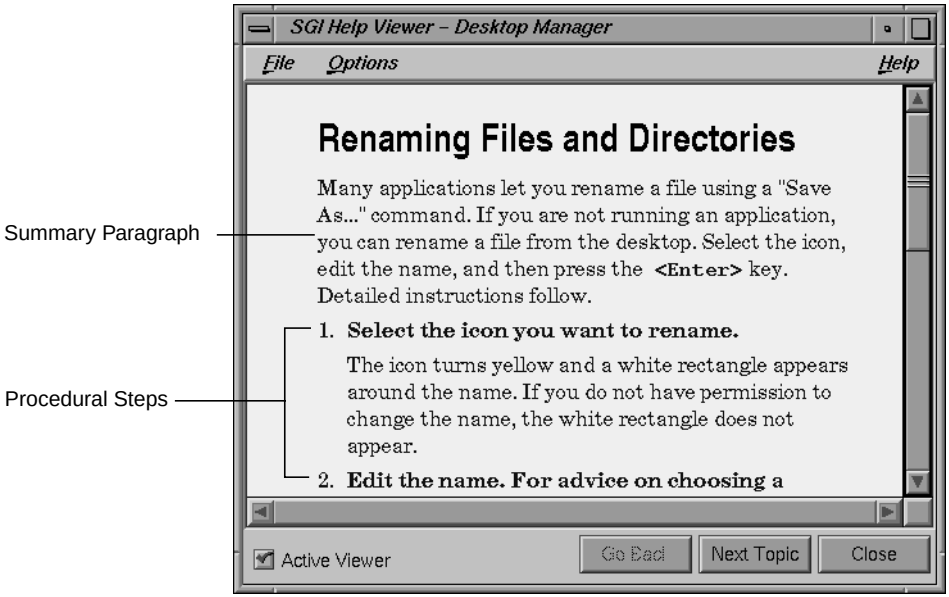


Figure 4-6 Task-Oriented Help Example

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. 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 "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 many 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 (contact Silicon Graphics for more information).

Desktop Variables

The IRIX Interactive 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.

Desktop settings described in this section include:

- "Scheme Setting"
- "Auto Window Placement Setting"
- "Language Setting"
- "Mouse Double-Click Speed Setting"
- "Editor Preference Setting"

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," in the *IRIX Interactive Desktop Integration Guide*. 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 define the mouse settings 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.

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 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*.

Figure 4-7 Language Control Panel

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). Don't reset the speed in your application.

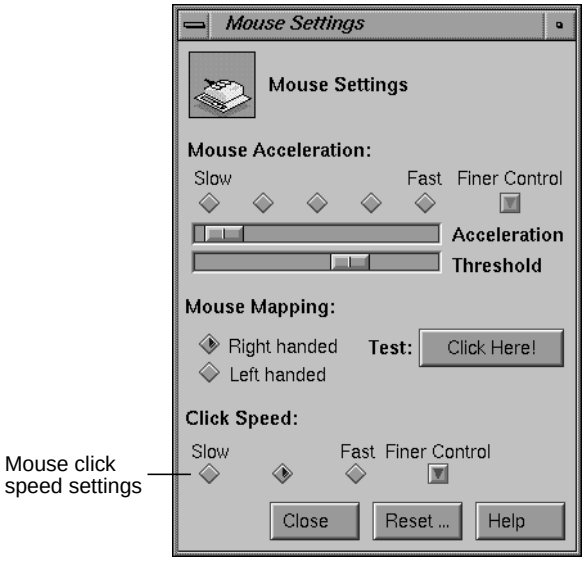


Figure 4-8 Mouse Settings Control Panel

Editor Preference Setting

When users open ASCII text files on the IRIX Interactive 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.

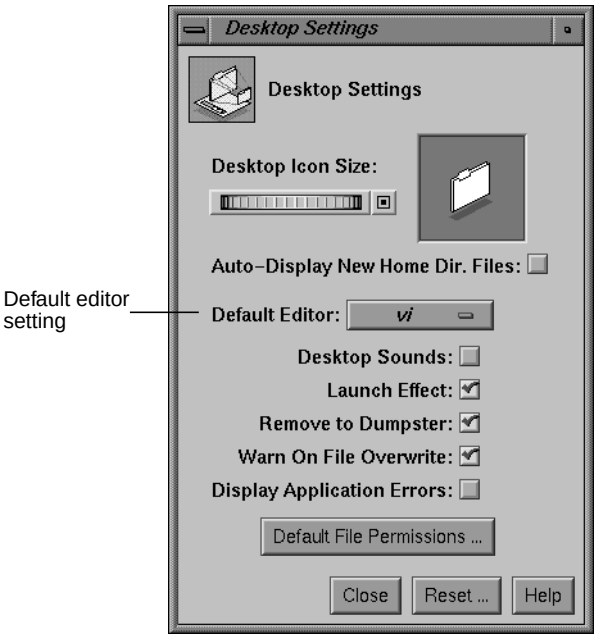


Figure 4-9 Desktop Settings Control Panel

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*.

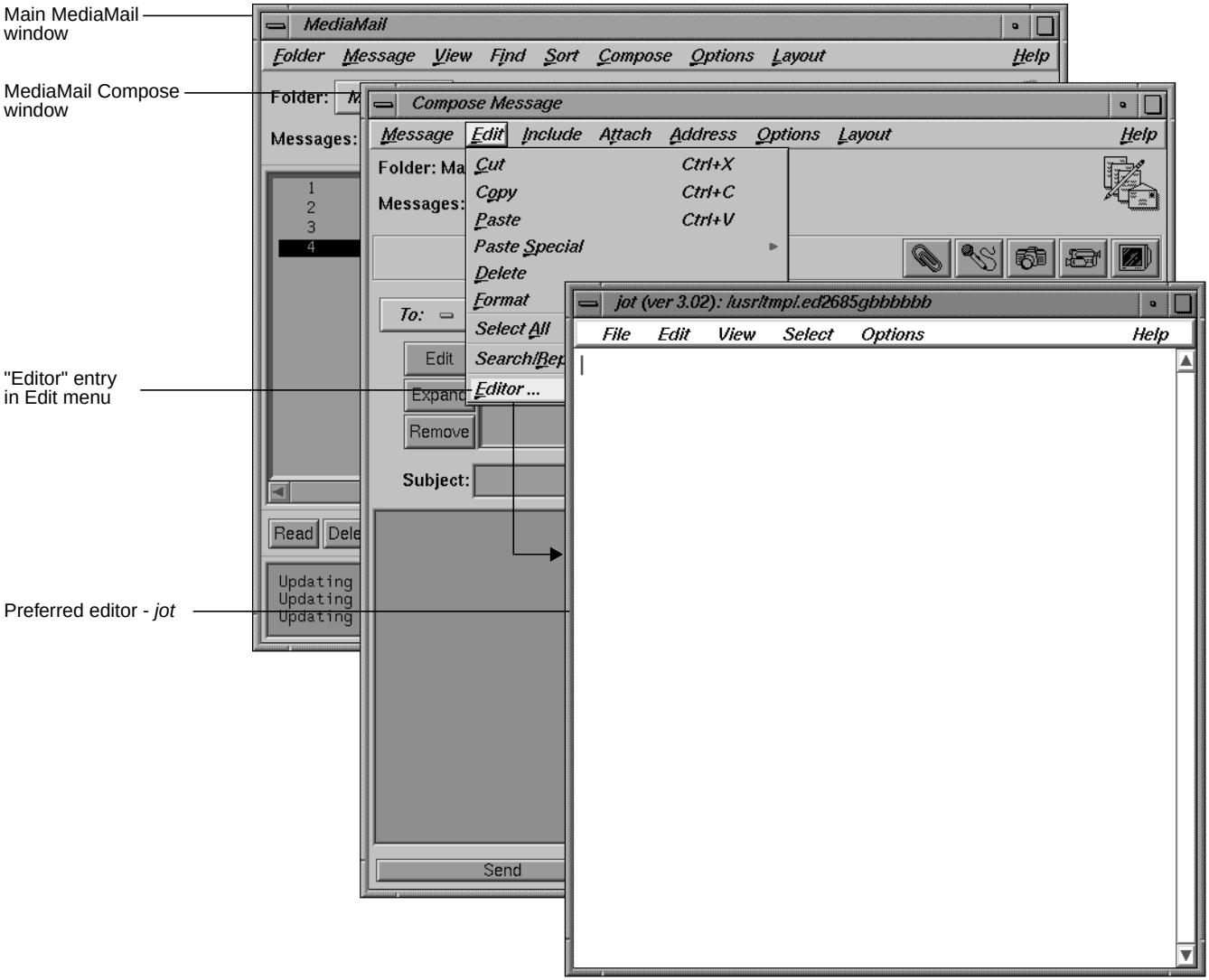


Figure 4-10 Selecting Preferred Editor in MediaMail

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 *IRIX Interactive Desktop Integration Guide* for details on using FAM in your application.)

Here are two examples of IRIX Interactive Desktop 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.

Data Exchange on the IRIX Interactive Desktop

The IRIX Interactive 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 theusing "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, “Interapplication Data Exchange,” of the *IRIX Interactive 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. For persistent media storage use the IRIX Interactive Desktop MediaWarehouse. For more information, see the *IRIX Interactive Desktop MediaWarehouse User’s Guide*.

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.

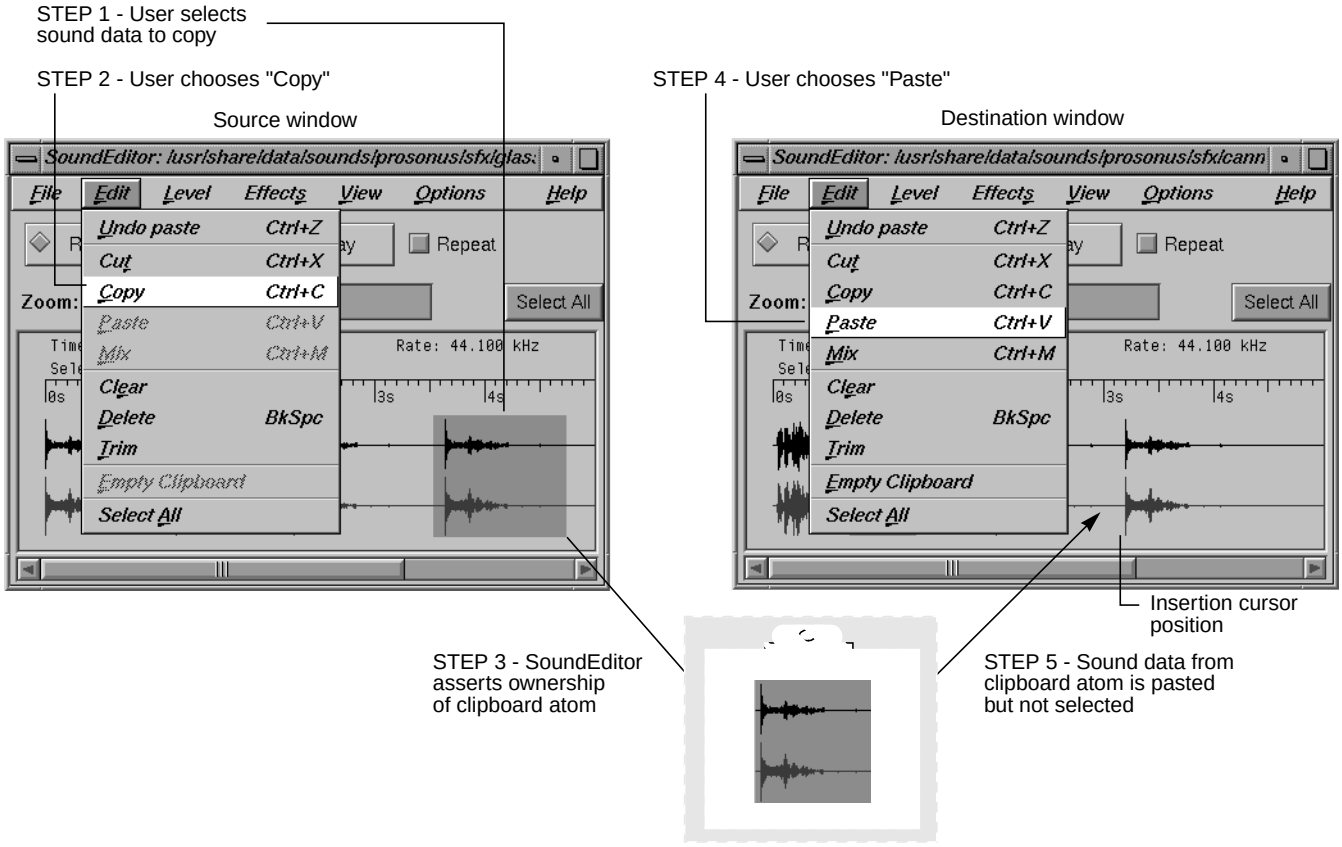


Figure 5-1 Clipboard Transfer Example

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, “Interapplication Data Exchange,” of the *IRIX Interactive 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.

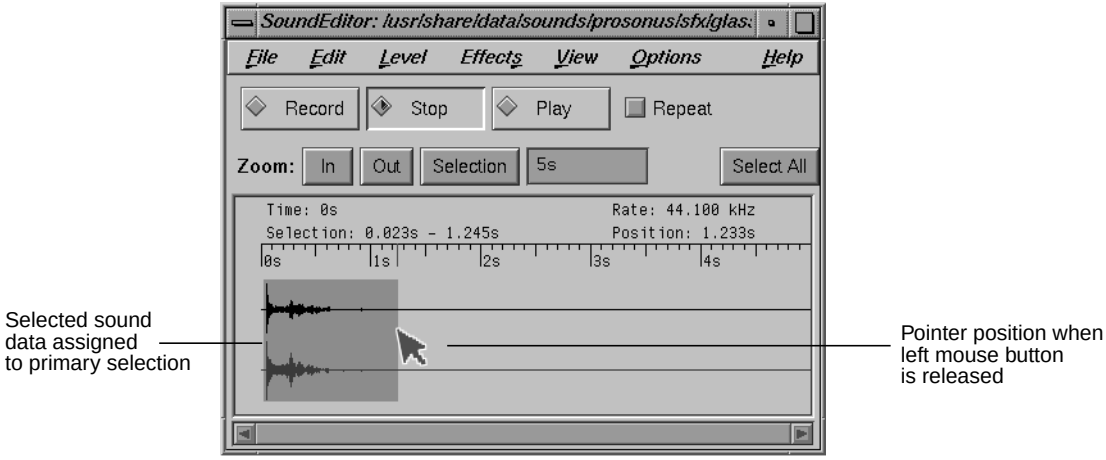


Figure 5-2 Primary Selection Example

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.

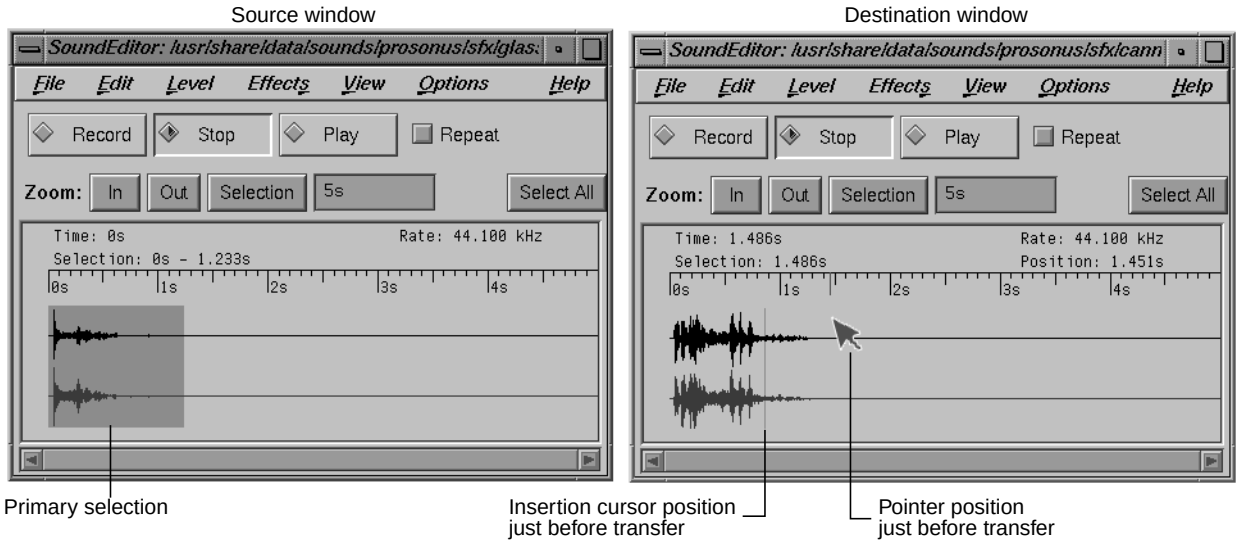


Figure 5-3 Primary Transfer Example: Before Transfer

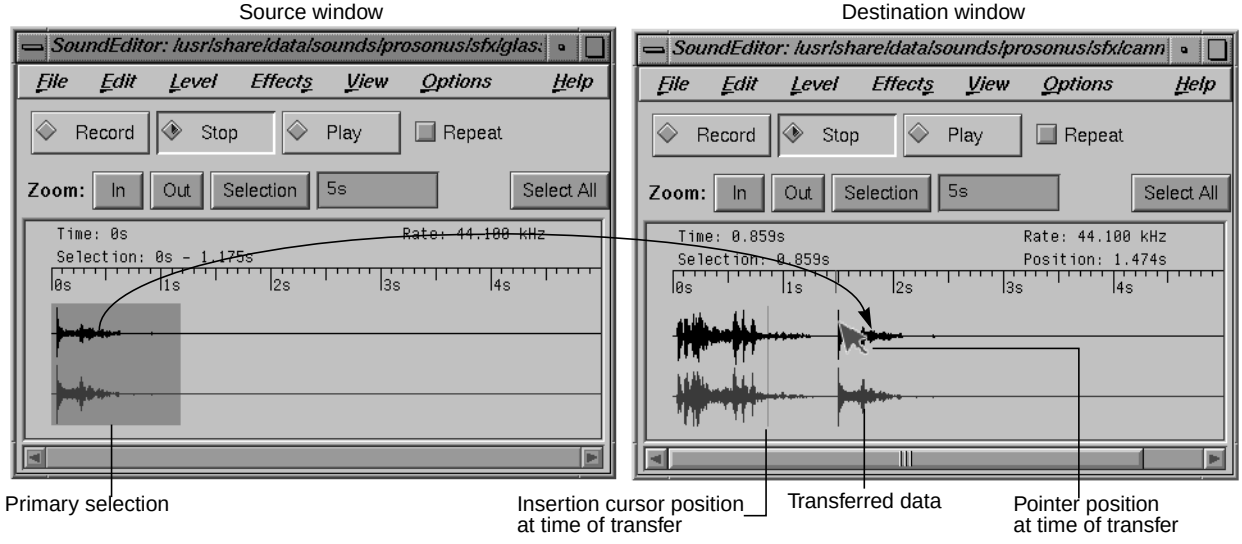


Figure 5-4 Primary Transfer Example: After Transfer

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. Make the "Promote" menu entry 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 for both clipboard and primary transfer of the corresponding types of data. A few atom names are listed in Table 5-1; the tables in Chapter 7, "Interapplication Data Exchange," in the *IRIX Interactive Desktop Integration Guide* provide a complete list. If you want users to be able to transfer data from your application to other applications, you need to be able to export the data. If your application is to receive data from other applications, it must be able to import the data. For complete details of implementing both clipboard and primary data transfer, see Chapter 7, "Interapplication Data Exchange," in the *IRIX Interactive 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

Chapter 7

Focus, Selection, and Drag and Drop

Chapter 8

Menus

Chapter 9

Controls

Chapter 10

Dialogs

Chapter 11

User Feedback

Application Windows

This chapter discusses the role of different types of windows in the IRIX Interactive Desktop 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" describes the design of primary windows.
- "Support Windows" explains 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 IRIX Interactive 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 IRIX Interactive Desktop environment can be used together in applications. Topics include:

- "Window Types"
- "Standard Application Models"

Window Types

As discussed in "Application Window Categories" in Chapter 3, windows in the IRIX Interactive Desktop 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. When choosing a model, try to keep the window hierarchy shallow so 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.

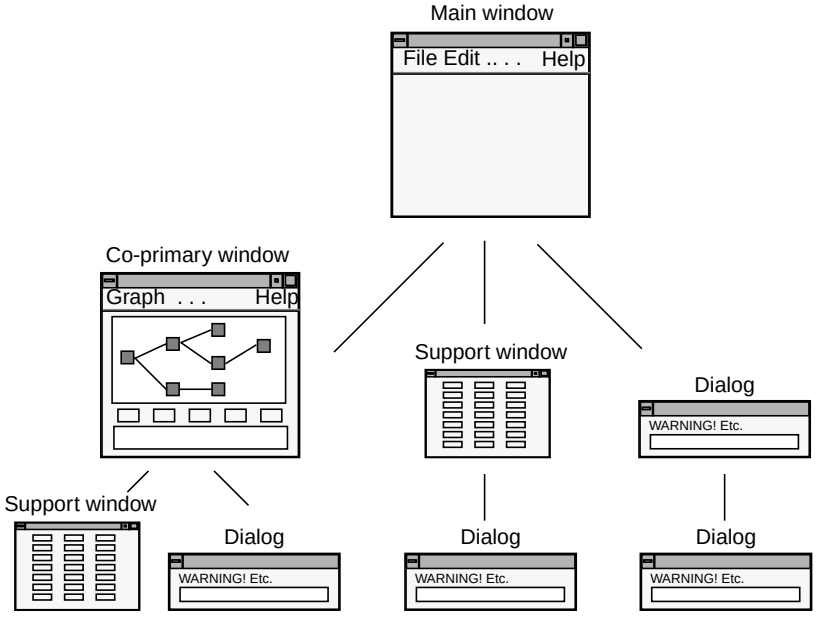


Figure 6-1 Allowable Parent-Child Window Relationships

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 Chapter 5, "Window, Session, and Desk Management," in the *IRIX Interactive 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.

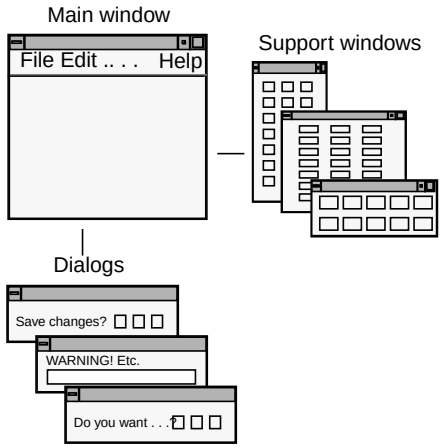


Figure 6-2 "Single Document, One Primary" Application Model

"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.

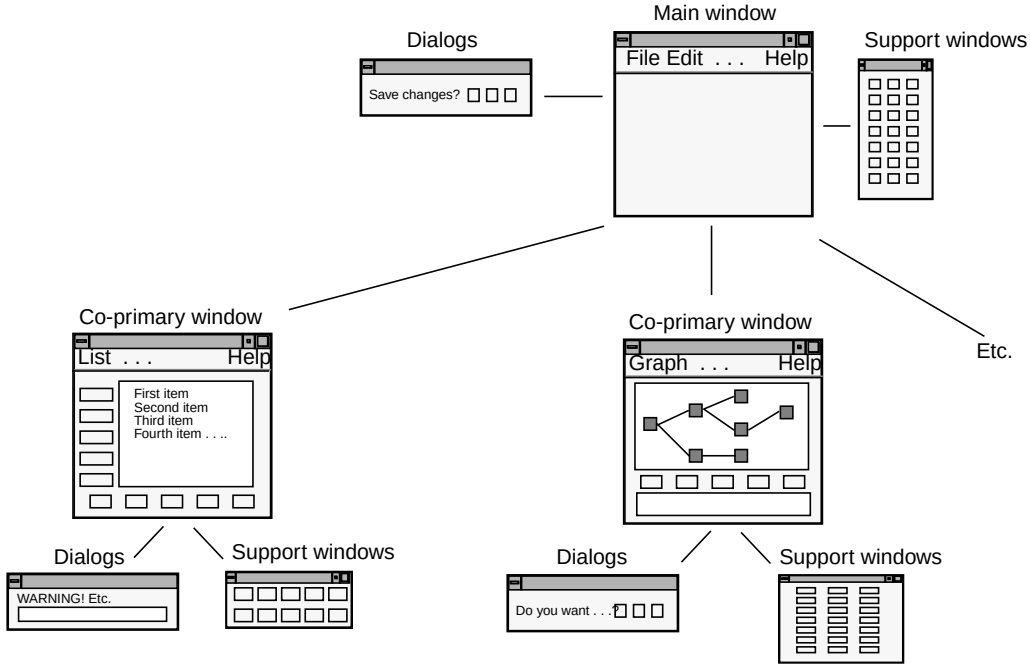


Figure 6-3 "Single Document, Multiple Primaries" Application Model

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, use the "multiple documents, no visible main" model; make the menu entries from the main window available from the pull-down menus in each of the co-primary windows, as described in the next section.

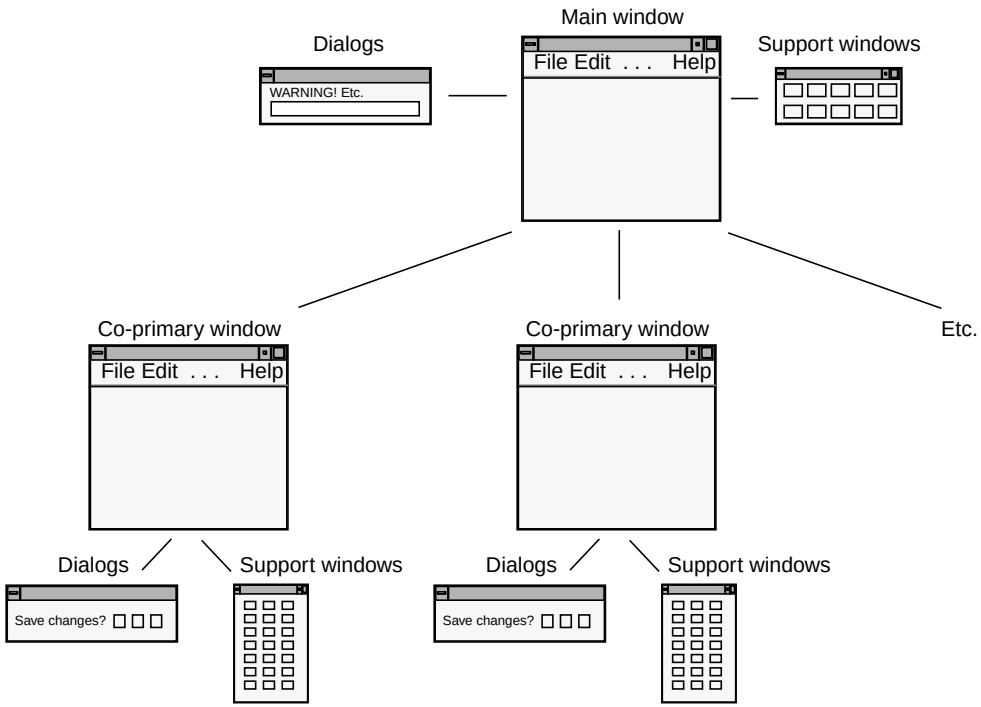


Figure 6-4 "Multiple Document, Visible Main" Application Model

"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.

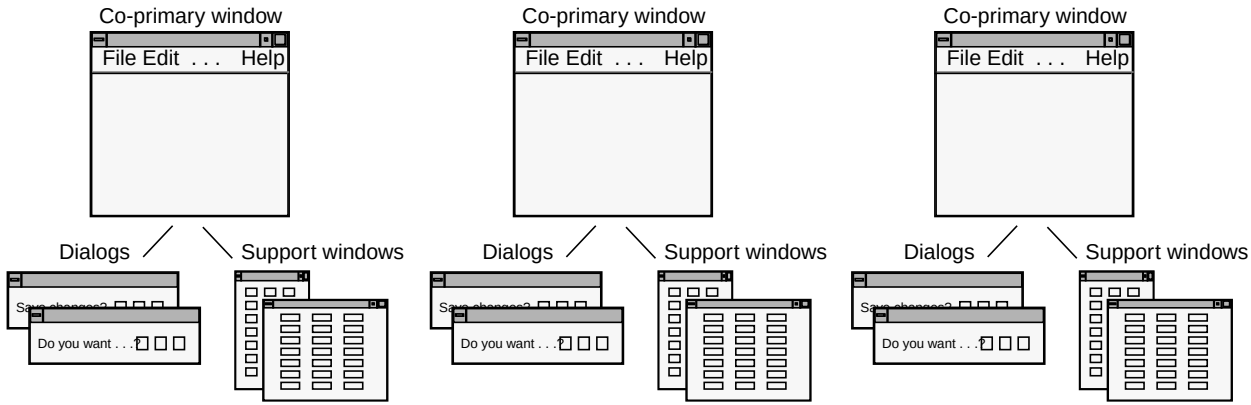


Figure 6-5 "Multiple Document, No Visible Main" Application Model

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 that 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).

This section discusses main and co-primary windows:

- "Menu Bars in Primary Windows"
- "Scrollable Work Areas in Primary Windows"
- "Control Areas in Primary Windows"
- "Status Areas in Primary Windows"
- "Splitting Primary Windows Into Panes"
- "Popup Menus in Primary Windows"

Several typical layouts for primary windows are shown in Figures 6-6 through 6-8. These layouts are discussed in the following sections.

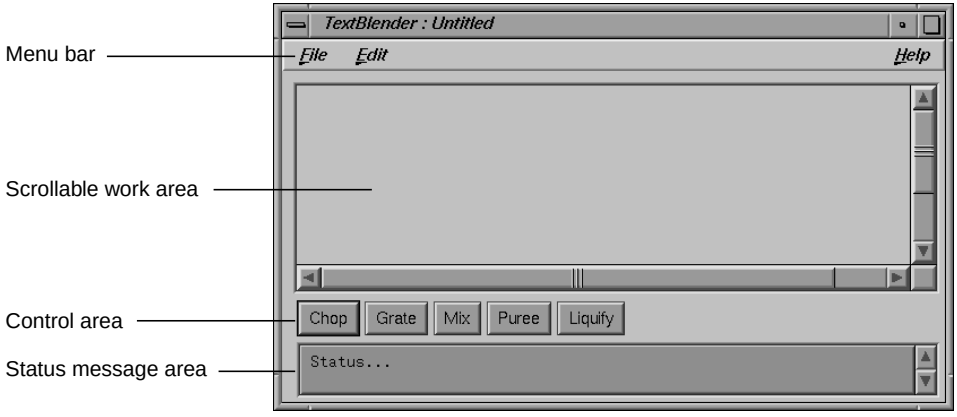


Figure 6-6 Basic Primary Window

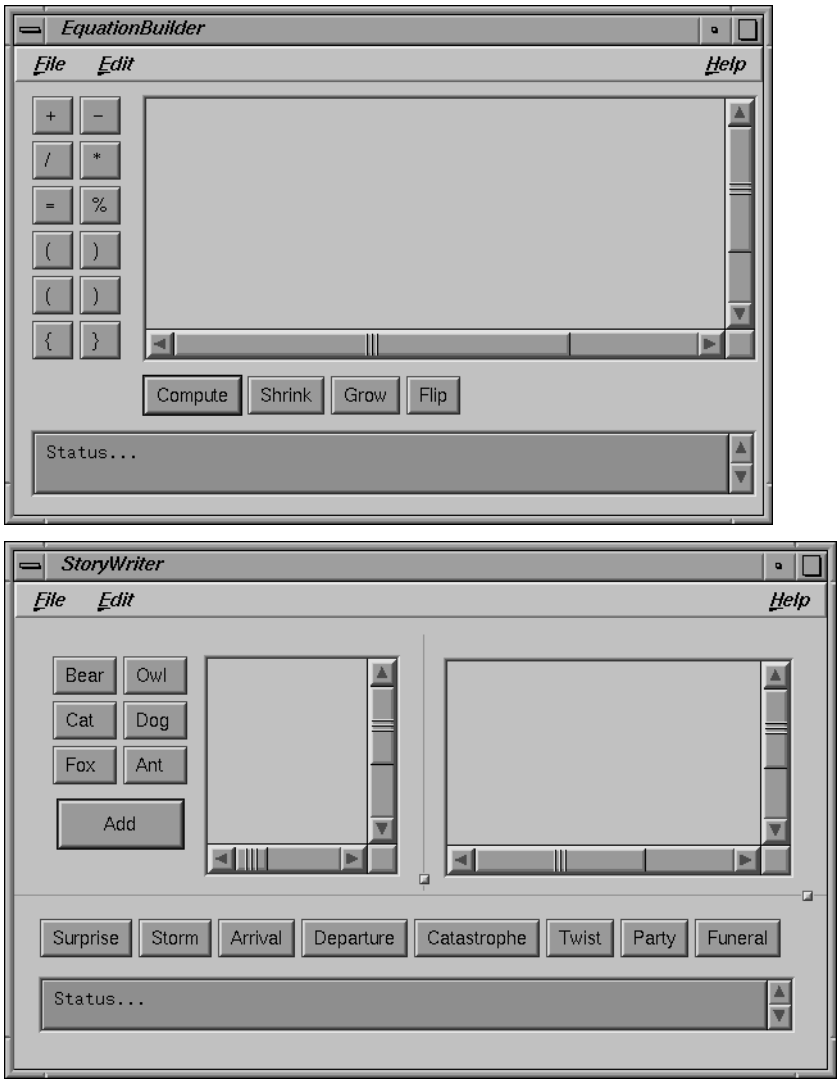


Figure 6-7 Primary Windows With Tool Palettes

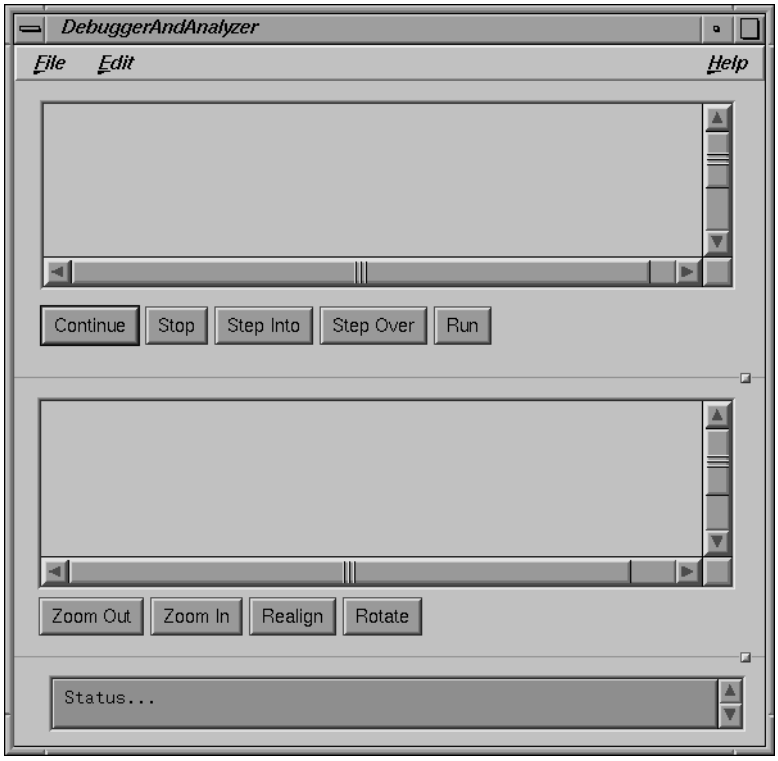


Figure 6-8 Primary Window With Two Panes

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. Don't create 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. Put controls in the control area, as described later in "Control Areas in Primary Windows." Put status information 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, don't place controls 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. Provide vertical scrollbars for this area so that users can view previously displayed messages.

Don't use this area for warnings, errors, or other kinds of messages requiring the user to respond. Instead, use dialogs to display these types of messages. (See Chapter 10, "Dialogs," for guidelines on designing dialogs.) Also, don't use it 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:

- "General Support Window Design"
- "A Specific Standard Support Window: The IRIX Interactive Desktop 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"). Launch support windows 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. Don'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.

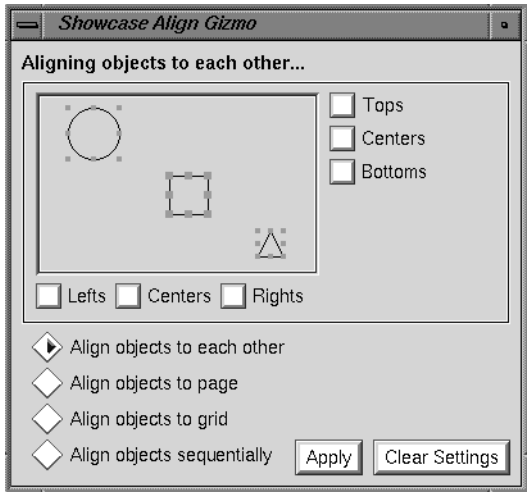


Figure 6-9 The IRIS Showcase Align Gizmo

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, use a modal dialog (see "Dialog Modes" in Chapter 10).

A Specific Standard Support Window: The IRIX Interactive Desktop Color Chooser

The IRIX Interactive Desktop color chooser is a standard support window that allows users to edit colors. 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.

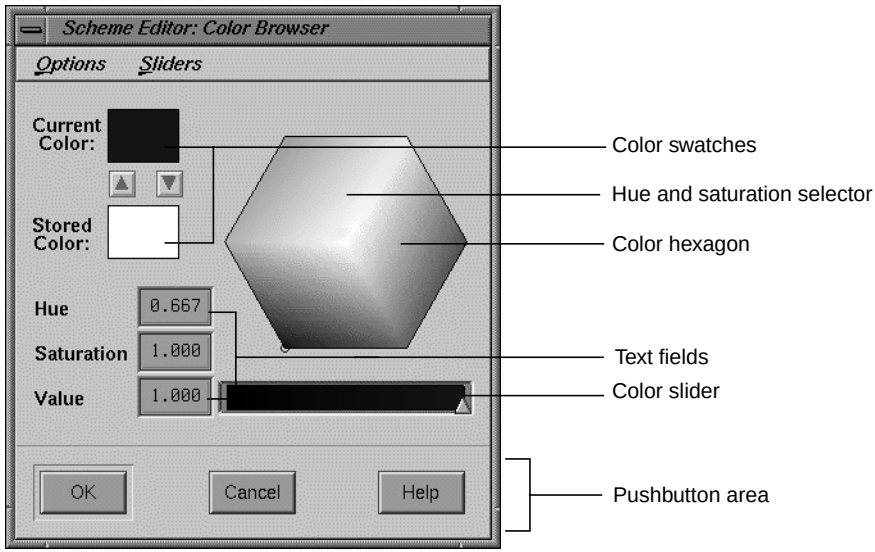


Figure 6-10 The IRIX Interactive Desktop Color Chooser

You can allow users to access the IRIX Interactive Desktop 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 IRIX Interactive Desktop color chooser in your application, see Chapter 4, "Using the Silicon Graphics Enhanced Widgets," of the *IRIX Interactive 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 IRIX Interactive Desktop 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.

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 IRIX Interactive Desktop 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 IRIX Interactive 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 and includes:

- "Keyboard Focus Policy and Navigation Within a Window"
- "Keyboard Focus and Navigation Guidelines"

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.

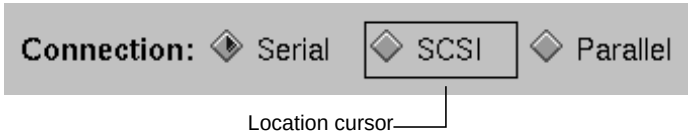


Figure 7-1 Location Cursor Example

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 include a mouse, it's not 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. They also 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 can 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.

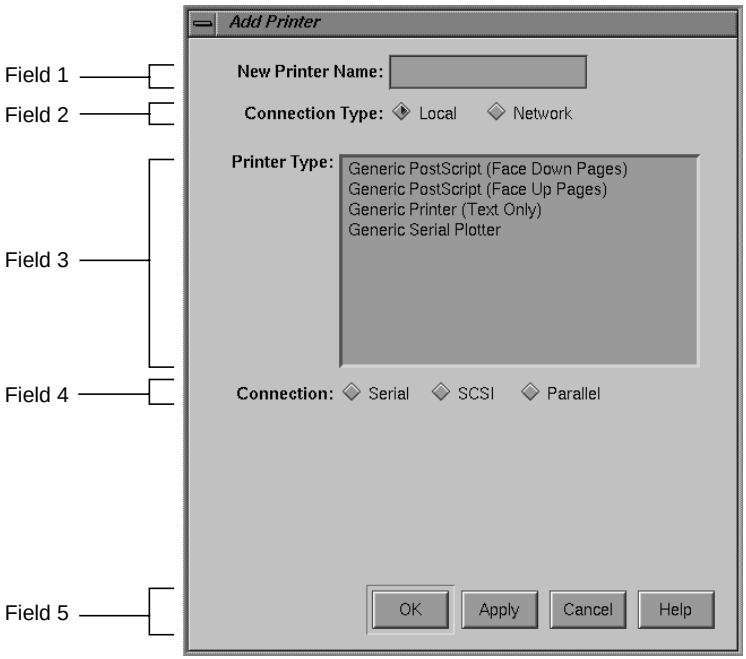


Figure 7-2 Components and Fields

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 put the pointer over a specific component and click the left mouse button. The keyboard focus moves to the selected component, if you've allowed that component to accept keyboard focus, and 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, certain components do not 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.

This section describes:

- "Selection Models—What Can Be Selected and How To Select It"
- "Highlighting a Selection"
- "Multiple Collections in One Application Window"

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.
<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.

Table 7-1 (continued) Selection Actions and Results

Action	Model	Result
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.

This section covers the following topics:

- "Two Models of Drag and Drop"
- "Pointers for Drag Operations"

Two Models of Drag and Drop

Two models for drag and drop exist, 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, IRIX Interactive Desktop 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.

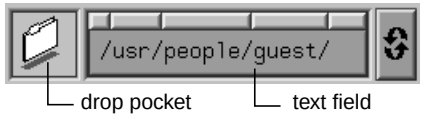


Figure 7-3 File Finder Component

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.

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

IRIX Interactive Desktop supports three types of menus, all of which are defined in the *OSF/Motif Style Guide*: pull-down menus, popup menus, and option menus. This section discusses these menu types:

- "Pull Down Menus"
- "Popup Menus"
- "Option Menus"

Pull Down Menus

Of the three types, pull-down menus are 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.

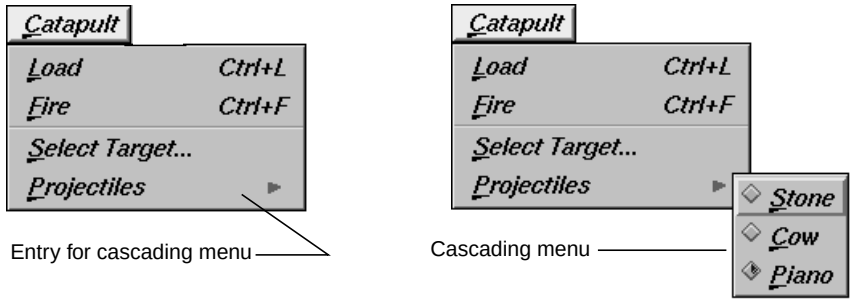


Figure 8-3 Cascading Menu

Popup Menus

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.

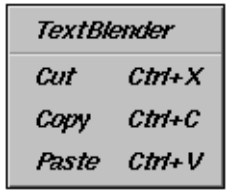


Figure 8-4 Popup Menu

Option Menus

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.

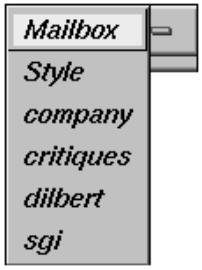


Figure 8-6 An Open Option Menu

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.

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. This section describes:

- "Using the Mouse to Manipulate Menus"
- "Using the Keyboard to Manipulate Menus"

Using the Mouse to Manipulate Menus

With the mouse, users have the additional choice of manipulating menus in either a *spring-loaded* or a *posted* manner.

Spring-Loaded 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.

Posted Manner

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.

Mouse Click

In addition to supporting this default model for manipulating spring-loaded and posted menus with the mouse, make sure your application handles 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.

Using the Keyboard to Manipulate 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, make sure the application ignores 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. This section describes the menu bar and pull-down menus:

- "Standard Menus"
- "What to Put in the Pull-Down Menus"
- "Choosing Mnemonics"
- "Choosing Keyboard Accelerators"
- "Disabling Menu Entries"
- "Dynamic Menu Entries"

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.

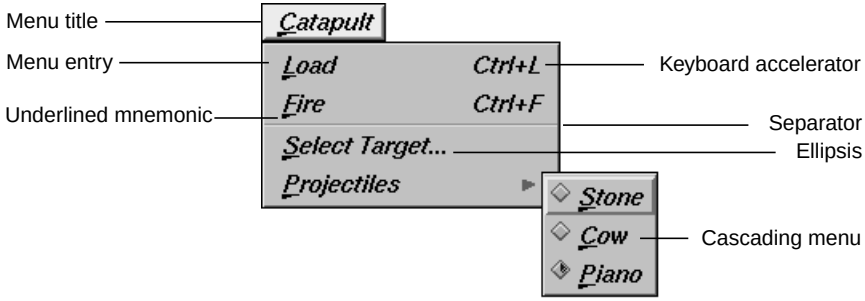


Figure 8-7 Elements of a Pull-Down Menu

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, 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; however, use the standard set as the starting point for the overall menu structure. Standard menus include:

- "File Menu"
- "Selected Menu"
- "Edit Menu"
- "View Menu"
- "Tools menu"
- "Options menu"
- "Help menu"

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 IRIX Interactive Desktop 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 (Table 8-3 lists standard keyboard accelerators). 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 lists each File menu entry, its mnemonic, OSF/Motif behavior, IRIX Interactive Desktop additions and exceptions, and keyboard accellator.

Table 8-1 File Menu Entries

Menu Entry and Mnemonic	OSF/Motif Behavior	IRIX Interactive Desktop 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 display 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).	
Import[a]	Not defined.	"Import" allows a user to read an existing data file into the current application. This entry can display the IRIX Interactive Desktop 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 IRIX Interactive Desktop 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 display the file selection dialog to allow the user to specify the specific file to import. Follow this entry with a separator.	

Table 8-1 (continued) File Menu Entries

Menu Entry and Mnemonic	OSF/Motif Behavior	IRIX Interactive Desktop Additions and Exceptions	Keyboard Accelerator
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
Exit	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 IRIX Interactive 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 should not contain editing actions such as “Cut” since these should be in the Edit menu. Use “S” as the mnemonic for the Selected menu.

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 IRIX Interactive Desktop.” Figure 8-10 shows the most common entries in the standard Edit menu; use “E” as its mnemonic.

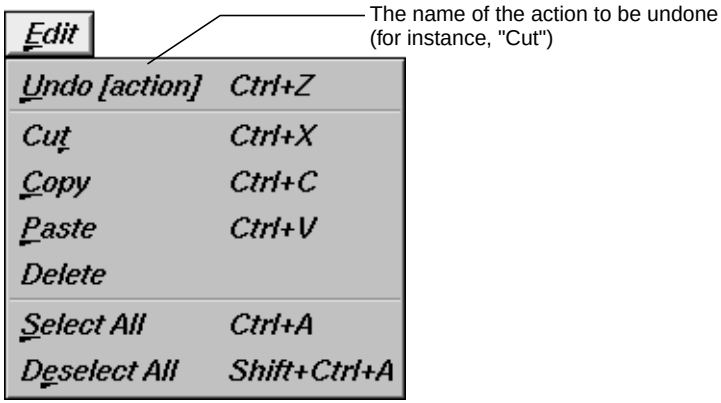


Figure 8-10 The Standard Edit Menu

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. Make sure that all of these entries 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	IRIX Interactive Desktop 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 display 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

Table 8-2 (continued) Standard Edit Menu Entries

Menu Entry and Mnemonic	OSF/Motif Behavior	IRIX Interactive Desktop Additions and Exceptions	Keyboard Accelerator
Paste	Copies the contents of the clipboard into the application window.	If there's nothing currently on the clipboard available to be pasted, display a dialog saying there's nothing available. See "Invoking Dialogs" in Chapter 10.	Ctrl+V
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" is "A."	Ctrl+A
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" is "l."	Shift+Ctrl +A
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.	
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" is "m."	Alt+ Insert
Color Editor...[a]	Not defined.	Choosing "Color Editor" invokes the IRIX Interactive Desktop color chooser, which allows the user to select colors. (See "A Specific Standard Support Window: The IRIX Interactive Desktop 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. Group together the entries representing the individual panes, and provide 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. You can find more information on menu checkboxes in "Using Radio Buttons and Checkboxes in Pull-Down Menus.")

Other entries in the View menu can 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. Use "V" as the mnemonic for the View menu.

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. Use "T" as the mnemonic for the Tools menu.

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. Use "O" as the mnemonic for the Options menu.

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

Make sure your application's pull-down menus include the standard menu entries that are relevant to your application, plus the 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 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, make all simple, frequent actions 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; make sure that all of your entries fit on the screen at one time because Motif 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.

Provide users with the option of using the keyboard for frequently used actions rather than restricting them 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, make these actions available in your application's 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.) Make sure that users can access such support windows as well as co-primary windows from the menu bar of their parent window. Make sure that these windows have an appropriate titlebar.

Naming Menus in the Menu Bar

Use one-word (capitalized) titles in all menus in the menu bar since users may interpret a second word as a separate menu title. Use entire words for menu titles rather than abbreviations. 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 are 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). Don't use abbreviations in menu entries.

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 separate the entire group 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:

- Groups where only one of the entries can be selected at any one time
- 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,” 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

If a user can select only one of a group of menu items at any one time, provide a radio button next to each item in the group, and allow only one radio button to 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, because if you ordered two cups of tea at a time the second one would get cold before you could drink it. Use separator lines to separate a set of radio buttons from other entries in the menu.

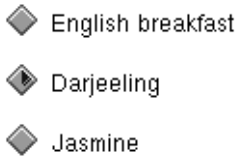


Figure 8-11 Radio buttons

If a user can select several of a group of related menu items at any one time, provide a checkbox next to each item in the group, and show the active entries 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. Use separator lines to separate a set of checkboxes from other entries in the menu.

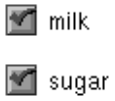


Figure 8-12 Checkboxes

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. 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." Table 8-3 lists the standard keyboard accelerators.

Table 8-3 Keyboard Accelerators

Menu Entry and Mnemonic	Keyboard Accelerator
New	Ctrl+N
Open	Ctrl+O
Save	Ctrl+S
Print	Ctrl+P
Get Info	Ctrl+I
Close	Ctrl+W
Exit	Ctrl+Q
Undo [action]	Ctrl+Z
Redo [action]	Shift+Ctrl+Z
Cut	Ctrl+X
Copy	Ctrl+C
Paste	Ctrl+V
Select All	Ctrl+A
Deselect All	Shift+Ctrl+A
Promote	Alt+Insert
Click for Help	Shift+F1

If 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," disable menu entries that aren't currently available (they become 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"), disable it if there's no current selection. If selecting the menu entry would result in no action at all (not even an error message), do not disable the menu entry. 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 never disable this menu entry.

Never disable menu entries that launch modeless dialogs. If the dialog isn't applicable when it's launched, disable the OK and Apply buttons on the dialog 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 less than four such entries exist. 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. Users might not realize that they must first get the application in a particular state before they can even see the action. Even when users work 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 entire one-word titles for menus rather than abbreviations.
- ❑ 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.
- ❑ Use entire words rather than abbreviations.
- ❑ 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. This section covers:

- "What to Put in Popup Menus"
- "Disabling Popup Menu Entries"

For example, you might provide a popup menu containing "Cut," "Copy," "Paste," and "Delete" in a text application. Never allow popup menus to 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, provide a different popup menu for each main area of your application's 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.

Provide one set of entries in the popup menu, and enable and disable each of them as appropriate, instead of following the OSF/Motif model. 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

For each popup menu, include a title followed by a separator and the individual menu entries (see 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, display titles similarly: in the same order, and with the same or very similar names as in the pull-down menu. Include ellipses and keyboard accelerators if they're included in the corresponding entry in the pull-down menu, but don't show mnemonics in popup menus.

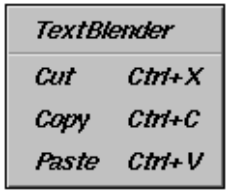


Figure 8-13 Popup Menu

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, disable that menu entry. Don't, however, remove it 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.

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 IRIX Interactive Desktop 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 IRIX Interactive Desktop environment as described in "Enhanced Graphics in the IRIX Interactive Desktop Look" in Chapter 3. To use these enhanced controls in your application, see , "Using the Silicon Graphics Enhanced Widgets," of the *IRIX Interactive 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 describes the following controls:

- "Pushbuttons"
- "Option Buttons"
- "Checkboxes"
- "Radio Buttons"
- "LED Indicators"
- "Lists"
- "Text Fields"
- "Scrollbars"
- "IRIX Interactive Desktop Scales"
- "Labels"
- "File Finder"
- "Thumbwheels"
- "Dials"

Pushbuttons

A pushbutton is a button that invokes an operation. Pushbuttons are rectangular and can be labeled 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.

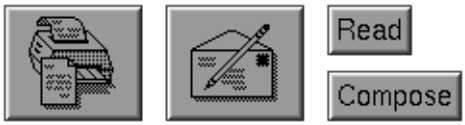


Figure 9-1 Pushbuttons

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 labeling 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. Make each text label a single, capitalized word. Don't use abbreviations in labels.
- 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.

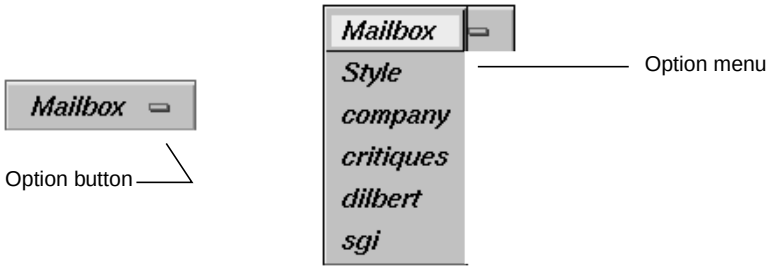


Figure 9-2 Option Button and Option Menu

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 labeling 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. Place this parameter label to the left of the option button and put a colon (:) and a space after it (see Figure 9-2). This label is typically a noun and is not abbreviated.

When labeling the entries in an option menu . . .

- ❑ Use nouns that indicate the possible values of the parameter being set.
- ❑ Use entire words for the entries rather than abbreviations.

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 IRIX Interactive Desktop 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 labeling 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.
- ❑ Don't use abbreviations for either the checkbox labels or the group label.

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 a checkbox exists for turning underlining on and off and the user selects some text, update the checkbox 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 IRIX Interactive Desktop 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.

Output Image:

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, 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 labeling 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.
- ❑ Don't use abbreviations for either the radio button labels or the group label.

When displaying radio buttons . . .

- ❑ Keep radio buttons updated. If the settings of the radio buttons depend on the current selection, update them 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.

LED Indicators

The LED Button is a toggle button with an enhanced appearance. It appears in the form of a push button, with an imbedded indicator lamp (LED) that provides state feedback. The button has two states—on or off. When the button is toggled to the ON state, it appears lighted; when OFF, it appears darkened.

You can use a LED button (as shown in Figure 9-5) instead of a radio button to indicate that one of a range of options or modes is currently in effect. Alternatively, you can use a toggle button instead of a checkbox, to indicate whether a given option or mode is currently on or off.

In either case, an LED button is most appropriate in a context where other push buttons are used as part of a control panel, since an LED button is the same size as a push button.

Figure 9-5 LED Button

For LED example code, see "Example Programs for SGI Enhanced Widgets" in the *IRIX Interactive Desktop Integration Guide*.

LED Button Guidelines

When using LED buttons . . .

- Use LED buttons 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.)
- Don't use LED buttons 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 labeling LED buttons . . .

- ❑ Label each LED button with a term that describes the attribute, state, or option it controls.
- ❑ If there is a group of LED buttons, create a group label for the group, and indent the LED buttons below the label. Use a noun that describes the function of the group.
- ❑ Don't use abbreviations for either the LED button labels or the group label.

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, follow the selection guidelines described in "Selection" in Chapter 7. Figure 9-6 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 9-6 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 labeling a list . . .

- ❑ Label the list with a noun that indicates the function of the elements in the list. Don't use abbreviations in the label.
- ❑ Place the label directly above and either left-aligned with or slightly to the left of the first element of the list.

When labeling the list entries . . .

- ❑ If the elements in the list represent operations to perform, use active verbs. Otherwise, use nouns. In either case, use entire words rather than abbreviations.

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).

IRIX Interactive Desktop 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.

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 IRIX Interactive Desktop File Finder instead.
- Use text fields for values that change over time; don't use labels.

When labeling 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. Use entire words in labels rather than abbreviations.
- 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. Vertically center the label 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, show the default or current setting (if either exists) for the text field.
- 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 IRIX Interactive Desktop Look" in Chapter 3 describes enhancements to the scrollbar's appearance.) Figure 9-7 shows a scrollbar.

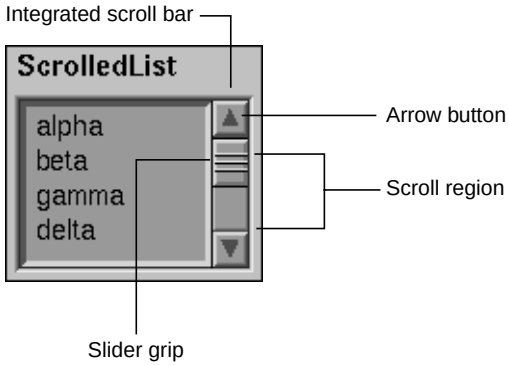


Figure 9-7 Scrollbar

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 IRIX Interactive Desktop thumbwheel instead. See "Thumbwheels" later in this chapter.
- Don't use scrollbars to choose a value in a range; use the IRIX Interactive Desktop 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 specific 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.

IRIX Interactive Desktop 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 9-8 shows the IRIX Interactive Desktop 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 IRIX Interactive Desktop scale in your application, see "The Scale (Percent Done Indicator) Widget" in Chapter 4, "Using the Silicon Graphics Enhanced Widgets," in the *IRIX Interactive Desktop Integration Guide*.

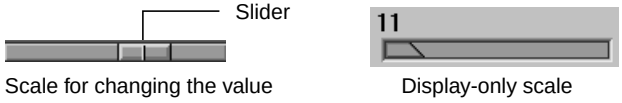


Figure 9-8 IRIX Interactive Desktop Scale

IRIX Interactive Desktop Scale Guidelines

When using the IRIX Interactive Desktop 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 labeling 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. Don't use abbreviations in this label.

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. 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, immediately update the slider 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 entire words in labels rather than abbreviations.
- ❑ 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, use a colon (:) and a space after the label, and place it 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 IRIX Interactive Desktop 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 9-9.

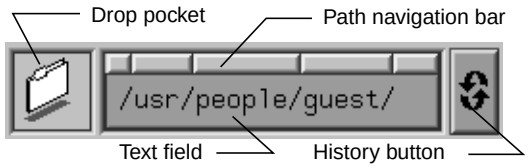


Figure 9-9 The File Finder

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. Also place this value in the history list under the history button.

Thumbwheels

The thumbwheel is an IRIX Interactive Desktop 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 9-10 shows a thumbwheel. For specific details on using the thumbwheel in your application, see the **SgThumbWheel**(3X) reference page.

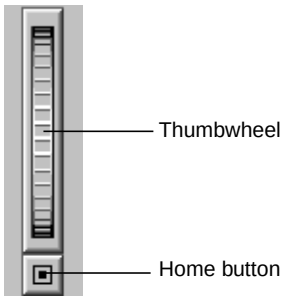


Figure 9-10 Thumbwheel

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 IRIX Interactive Desktop 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 9-11 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.

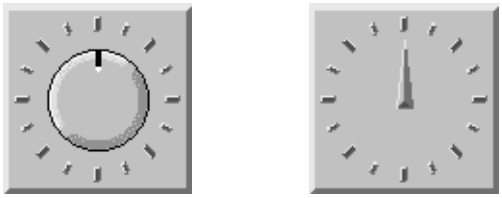


Figure 9-11 Dials

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 labeling 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 are below the dials or all the labels are above the dials).
- Use entire words in the label rather than abbreviations.

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.

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 to open). 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 the dialog window decorations, the layout of dialog information, 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 display the IRIX Interactive Desktop File Selection dialog so that the user can specify a file to open.

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 IRIX Interactive Desktop look, and Figure 10-2 shows the IRIX Interactive Desktop 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. Use this type of dialog 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 IRIX Interactive Desktop File Selection dialog, which is shown in Figure 10-2, is slightly different from the standard OSF/Motif File Selection dialog.	Modeless

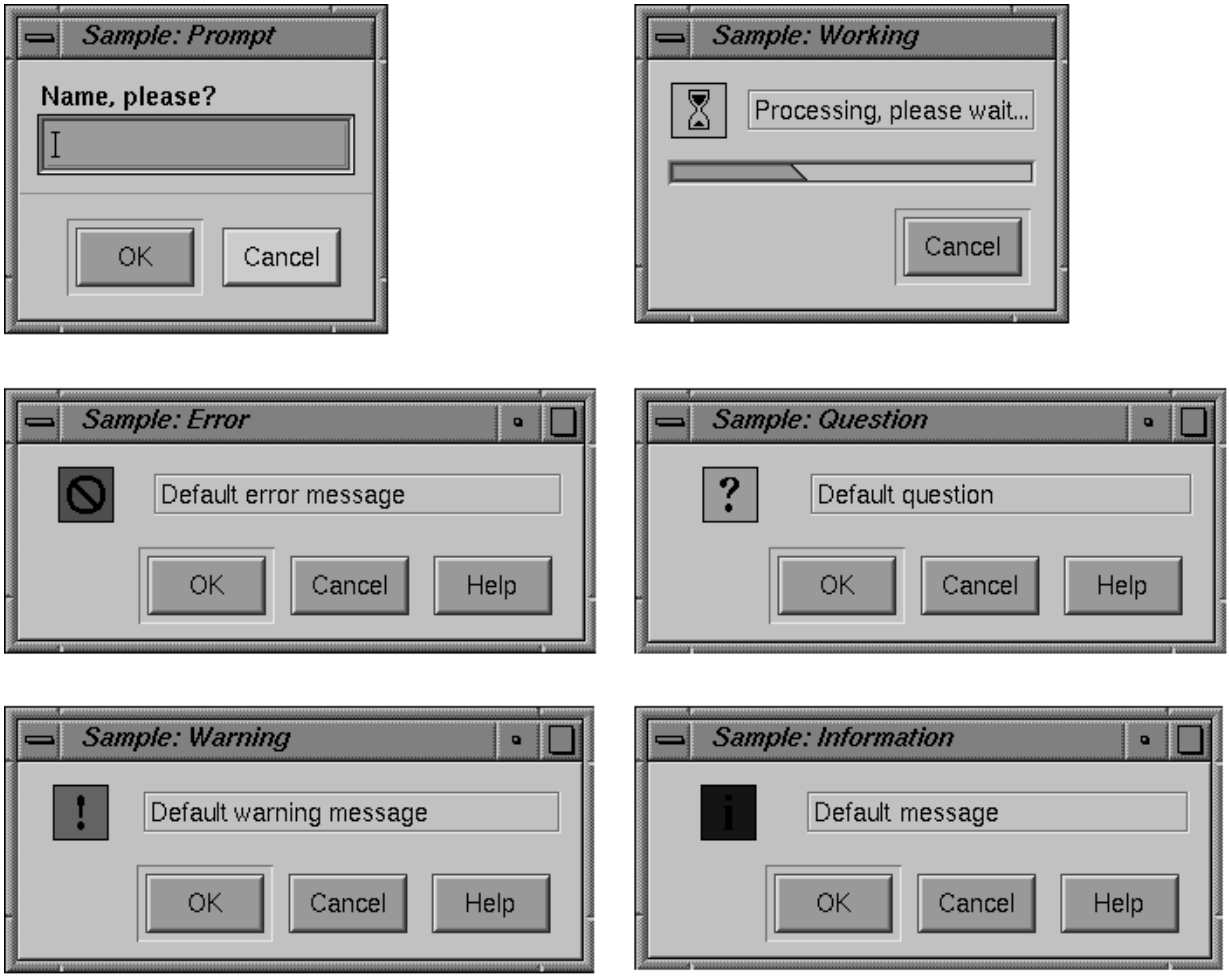


Figure 10-1 Sample Prompt, Error, Warning, Working, Question and Information Dialogs

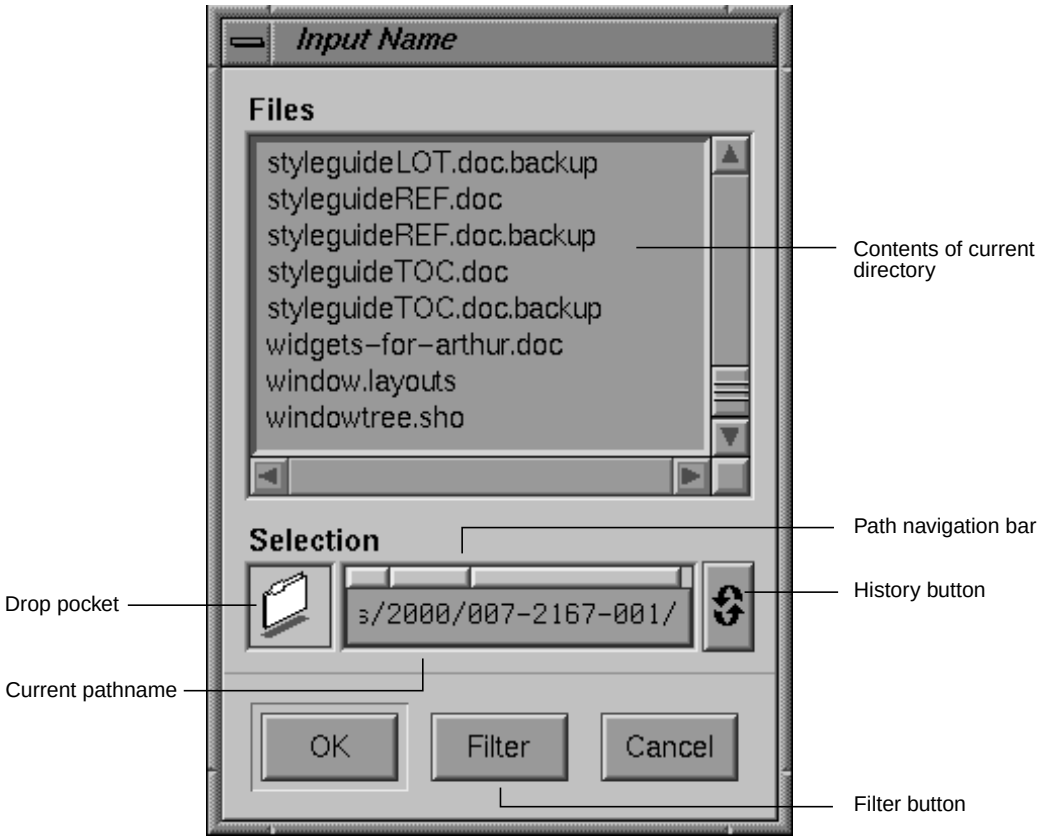


Figure 10-2 The IRIX Interactive Desktop File Selection Dialog

Both the IRIX Interactive Desktop and IRIS IM File Selection dialogs provide lists of the contents of the current directory and a text input field; the IRIX Interactive Desktop list contains both the files and the subdirectories of the current directory, while the IRIS IM list presents these in two separate lists. The IRIX Interactive Desktop 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 IRIX Interactive Desktop 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 IRIX Interactive Desktop 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 becomes useless. For example, the IRIX Interactive Desktop 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 are modeless. Error, Warning, and Question dialogs are 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 215 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 IRIX Interactive Desktop 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, dialog window decorations, size, information displayed when they first come up, and the layout of information. 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.

This section covers:

- "Decorations, Initial State, and Layout of Dialogs"
- "Standard Dialog Actions"
- "Content of Specific Types of Dialogs"

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 207.

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 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

Don't include menu bars in dialogs; 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 IRIX Interactive Desktop 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 IRIX Interactive Desktop 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. Consider using both 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, is OK.
- The default action for Question, Warning, Error, and any other dialogs that contain buttons but no text fields is 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," make sure that each button name clearly indicates 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) is 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). Don’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.

  Also don’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”), 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.

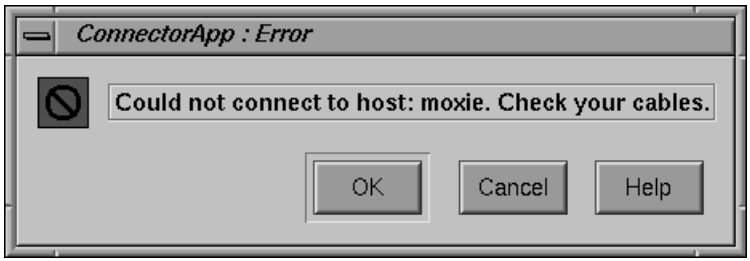


Figure 10-5 Error Dialog With Specific Entity

Warning Dialogs

Warning dialogs should clearly state what data is likely to be lost and why, and 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, use the IRIX Interactive Desktop scale to dynamically indicate the percentage of how much of the operation is complete. A Working dialog, also known as Progress dialog or Percent Done indicator, is shown in Figure 10-6. See "IRIX Interactive Desktop Scales" in Chapter 9 for more information about these indicators.

Figure 10-6 Working Dialog with IRIX Interactive Desktop 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), 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, 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, put them 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 IRIX Interactive Desktop 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 situations and gives an example of the dialog your application should provide if users can encounter this situation when interacting with your application. Topics include:

- "Invoking Dialogs When Manipulating Files"
- "Other Situations for Invoking Dialogs"

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, don't use application-generated dialogs to provide general status information; 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 IRIX Interactive Desktop 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 shows the last directory the user traversed to, and the file that the user opened the last time is selected. (See Table 10-1 and Figure 10-2 earlier in this chapter for more information about the IRIX Interactive Desktop File Selection dialog.)
- "Save As..." from the File (or leftmost) menu should display the IRIX Interactive Desktop 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, prompt them 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, prompt the user 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, prompt the user with the IRIX Interactive Desktop 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, prompt the user 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, prompt the user with the Warning dialog shown in Figure 10-8.

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, provide 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, prompt the user 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 IRIX Interactive Desktop 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 IRIX Interactive Desktop 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.

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 to follow, so they're not repeated here. Topics include:

- "Providing Graphic Feedback"
- "Keeping Information Up to Date"
- "Providing Messages to the User"

Providing 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). Design executable and data file icons to provide this sort of graphic feedback, as discussed in Chapter 2, "Icons."

The IRIX Interactive Desktop 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 IRIX Interactive Desktop look, which is discussed in "The IRIX Interactive Desktop 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 changes 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, highlight that data 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.

Keeping 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. 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.

Providing Messages to the User

In addition to providing immediate graphic feedback through your application's icons, components, and pointers, provide textual messages that describe your application's status. Keep in mind that by default the window manager for the IRIX Interactive 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 cascading 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*; therefore, use dialogs or status areas in your application instead.

In particular, use dialogs to provide warning messages, error messages, and work-in-progress feedback, as discussed in Chapter 10, "Dialogs." Also define an area on primary windows for status messages in the cases discussed in "Status Areas in Primary Windows" in Chapter 6. Finally, 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 IRIX Interactive Desktop 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 describes:

- "Standard Pointer Shapes and Colors"
- "Designing 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, 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 IRIX Interactive Desktop 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

Table 11-1 (continued) Standard Pointer Shapes and Colors

Pointer	Name	Purpose	Additions and Exceptions to OSF/Motif Style
	resize	Indicates positions when resizing an area.	none
	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 IRIX Interactive Desktop 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.

PART THREE

3D Style Guidelines

Chapter 12

Introduction to 3D Style Guidelines

Chapter 13

Interactive Viewing of 3D Objects

Chapter 14

Selection in 3D Applications

Chapter 15

Manipulating 3D Objects

Introduction to 3D Style Guidelines

A developer designing a user interface for a 3D application has to resolve some issues that don't arise in 2D user interface design. 3D applications have to provide easy and intuitive interaction in three dimensions with a 2D pointing device. Also, because users of 3D applications often need convenient access to all sides of the objects they view, many applications need to offer more than one way to view or edit scenes or objects.

This chapter provides general information that's useful for designing a 3D application that is consistent and behaves in ways that users expect. It covers these topics:

- "Making 3D Functionality Available" provides general guidelines for mapping functionality to the mouse buttons and for using modifier keys.
- "Pointer Shapes for 3D Functions" discusses the appropriate pointer shapes for some common functionality in 3D applications.
- "Resizing the 3D Viewing Window" describes how the contents of the 3D viewing window changes as the user resizes the window.

Because of their general nature, the guidelines discussed in this chapter are especially relevant if you can't find a guideline for a specific situation and therefore need to extend existing guidelines.

Making 3D Functionality Available

A 3D user interface designer typically has to make 3D functionality easily available within the constraints of a 2D input device and standard keyboard. This section provides an overview of two primary ways of doing this:

- "Designing Mouse Input for 3D Applications"
- "Using Modifier Keys in 3D Applications"

Guidelines for specific cases, for example, use of modifier keys during viewing, are discussed in detail in later chapters, where applicable.

Designing Mouse Input for 3D Applications

The section "Mouse and Keyboard Hardware" in Chapter 1 discusses the rules for mouse input that apply in 2D applications. They can be summarized as follows:

- The left mouse button provides basic functionality such as selecting and dragging objects.
- The middle mouse button is reserved for primary copy and paste operations and for advanced user shortcuts, which are also accessible through some more obvious interface.
- The right mouse button is reserved for popup menus.

The same is true in a 3D application:

- The most important functionality is mapped to the left mouse button. For example, if viewing is the user's primary task, the left mouse button provides access to viewing functionality such as navigating through a scene. If editing is the primary task, the left mouse button provides access to editing functionality such as selecting and manipulating objects.

  Modifier keys used in conjunction with the left mouse button provide additional functionality. For example, if the user is performing free translation (the default), no modifier keys are required. For constrained translation along one axis, the user has to hold down both the left mouse button and the <Shift> key. See "Free and Constrained 3D Manipulation" in Chapter 15.
- The middle mouse button is reserved for primary copy and paste operations and for advanced user shortcuts, which are also accessible through some more obvious interface.
- The right mouse button is reserved for popup menus which provide access to important functionality. See "Popup Menus" in Chapter 8.

Using Modifier Keys in 3D Applications

Using modifier keys allows you to map additional functionality to the left mouse button. Each modifier key extends the available direct manipulation functionality in a consistent way without compromising the functionality of the other mouse buttons. This consistency helps users learn when to use each modifier key. Table 12-1 lists the "3D definitions" for the <Shift>, <Ctrl>, <Alt>, and <Esc> keys.

Table 12-1 Use of Modifier Keys in a 3D Application

Key	Description
Shift	Constrains or unconstrains the default behavior for the left mouse button. For example, shifts from the default unconstrained translation in two dimensions to constrained axial translation in one dimension, or from the default constrained rotation in two dimensions to unconstrained free rotation in three dimensions. (See Chapter 15, "Manipulating 3D Objects," for how the <Shift> key modifies the default translate, rotate, and scale behaviors.)
Ctrl	Provides an alternate behavior that isn't thought of as constraining or unconstraining the default behavior for the left mouse button. For example, if users drag a translation plane using the left mouse button, the object moves along the plane (the default behavior). If users hold down the <Ctrl> key while performing the same drag operation, the object moves perpendicular to the plane. (See Chapter 15, "Manipulating 3D Objects," for how the <Ctrl> key modifies the default translate, rotate, and scale behaviors.)
Alt	Provides access to a view overlay as long as the <Alt> key is pressed. Returns the user to the previous mode when the <Alt> key is released. (See "View Overlay" in Chapter 13.)
Esc	Performs mode switches, for example from view mode to edit mode. Note that this use of the <Esc> key is consistent with the *OSF/Motif Style Guide,* which recommends using <Esc> to cancel an operation. In effect, you are canceling (exiting) one mode and entering another. (See "Separate View and Edit Modes" in Chapter 13.)

If your application uses the <Shift> key to constrain or unconstrain actions and uses the <Ctrl> key for alternate behavior, make the combination of the two available where it makes sense.

For example, the <Shift> key constrains scaling to enlarge or shrink the object in only one dimension rather than the default uniform scaling operation (enlarging or shrinking the object simultaneously in all three dimensions). The <Ctrl> key provides the alternate behavior of scaling around the opposite scaling handle rather than the default behavior of scaling around the center. Using the <Shift> and <Ctrl> keys in combination allows the user to perform constrained scaling around the opposite plane. (See "Scaling 3D Objects" in Chapter 15.)

Basic 3D Interface Design Guidelines

To provide mouse functionality that matches user's expectations in a 3D application...

- ❑ Assign primary functionality to the left mouse button.
- ❑ Use modifier keys in conjunction with the left mouse button to make additional functionality available.
- ❑ Reserve the middle mouse button for primary copy and paste operations and to provide access to advanced user shortcuts that are also accessible via a more obvious user interface.
- ❑ Reserve the right mouse button for popup menus.

When deciding on the use of modifier keys in a 3D application...

- ❑ Use the <Shift> key to constrain or unconstrain the default behavior of the left mouse button.
- ❑ Use the <Ctrl> key to allow alternate behaviors that aren't thought of as constraining or unconstraining the default behavior assigned to the left mouse button.
- ❑ Use <Shift> and <Ctrl> together where it makes sense to provide alternate behavior that's also constrained or unconstrained.
- ❑ Reserve the <Alt> key for a view overlay (see "View Overlay" in Chapter 13).
- ❑ Reserve the <Esc> key for mode switches, for example, switching from view mode to edit mode.

Pointer Shapes for 3D Functions

General guidelines for pointer feedback for all applications in the IRIX Interactive Desktop environment are discussed in "Pointer Shapes and Colors" in Chapter 11. This section discusses additional pointer shapes specifically designed for 3D applications.

Using the correct pointer feedback is crucial in a 3D application because users typically need to work with a great variety of functions, and these functions are often accessed using the left mouse button in conjunction with modifier keys. To help users recognize which function they are currently accessing and to help them learn the mapping between modifier keys and available functions, display standard pointer shapes whenever the

user accesses common 3D functions. Table 12-2 provides an overview of the common 3D functions and associated pointers; it includes references to the relevant sections in later chapters. See Table 13-1 for an overview of 3D viewing functions and the associated user interface.

Table 12-2 Pointers for 3D Functionality

Pointer	Function	Purpose
	Tumbling	User is rotating a scene in view mode. See “Tumbling” in Chapter 13.
	Dollying	User is moving a scene closer or farther away in view mode. See “Dollying” in Chapter 13.
	Panning	User is moving a scene up, down, left, or right without rotating it. User is in view mode. See “Panning” in Chapter 13.
	Roaming	User is moving through a fixed scene as if walking or flying. User can view objects but not edit them. See “Roaming” in Chapter 13.
	Tilting	User is tilting head up or down in a scene without moving through the scene. User is in view mode. See “Tilting” in Chapter 13.
	Sidling	User is sidestepping to the right or left in the scene being viewed or is moving up or down as if on an elevator. User is in view mode. See “Sidling” in Chapter 13.
	Seeking	User selects an object or target in view mode, and is then automatically moved closer to that object with each click. See “Seeking” in Chapter 13.
	Editing	User can select and manipulate objects, for example, translate, rotate, and/or scale them. See “Viewing and Editing in 3D Applications” in Chapter 13. This is also the “default” pointer shape when no special function is taking place.

Pointer Feedback Guidelines for 3D Applications

To help users stay oriented while working in your 3D application...

- ❑ Use the standard pointer shapes to indicate specific 3D functionality. Display these pointers whenever the user accesses that functionality.
- ❑ Use the standard pointers used in 2D applications as needed (for example, the watch pointer to indicate an operation is in progress).

Resizing the 3D Viewing Window

When users resize the viewing window in a 3D application, its contents changes based on whether or not the resize operation keeps the same aspect ratio:

- If the user keeps the same aspect ratio, the contents of the window stays the same but the scene in the viewing area gets proportionally bigger or smaller depending on whether the user made the window bigger or smaller.
- If the user changes the aspect ratio, the size of the objects in the scene stays the same but more of the scene becomes visible when the user makes the window bigger and less of the scene is visible when the user makes the window smaller.

Guidelines for Resizing Windows in 3D Applications

When designing the resizing operation for the viewing windows in your 3D application...

- ❑ If the user resizes the window and keeps its same aspect ratio, make the scene in the viewing area of the window proportionally bigger or smaller based on the resize action.
- ❑ If the user resizes the window and changes its aspect ratio during the resize operation, display more or less of the scene being displayed based on the resize action, but don't change the size of the objects in the scene.

Interactive Viewing of 3D Objects

Interactive viewing must be supported in the user interface of all 3D applications, even if those applications don't support editing.

Viewing 3D content is more complex than viewing a 2D image because of the added dimension. This added dimension means not only that there is more to look at, but also that there are more ways of looking at things. For example, users may want to view the sides, back, and top of a 3D model of a computer, walk through a virtual room, or fly through a 3D landscape. Interface designers have to determine the appropriate viewing functionality for their application and implement it in a consistent and intuitive way.

This chapter discusses interactive viewing of 3D objects in these sections:

- "Introduction to 3D Viewing" provides an introduction to the viewing paradigm and discusses some terminology.
- "3D Viewing Functions" introduces inspection and navigation, which are two different viewing modes, and describes the viewing functions an application needs to support in each mode.
- "3D Viewing Interface Trade-Offs" discusses 3D application design issues that developers typically must address and provides recommendations for resolving these issues.

Introduction to 3D Viewing

3D viewing can be thought of as using a camera to view the world. The following concepts are used in this document to describe the user interface to viewing functions (see Figure 13-1):

- **Eyepoint**. The eyepoint is the position of the user's eye. The camera is always positioned at the eyepoint. As the user moves the location of the camera, the location of the eyepoint also changes.
- **Viewing area.** The viewing area is what the user can currently see while looking through the camera. It's what the user sees in the application's viewport.
- **Viewing direction**. The viewing direction refers to how the camera is oriented in space. As the user turns the camera to the left or right or tilts the camera up or down, the viewing direction changes accordingly. As the user changes the viewing direction, the contents of the viewing area also changes. In effect, the user is looking through the camera at a different part of the scene.
- **Look-at point**. The look-at point is the current center of interest within the scene. The camera's viewing direction is always aimed so that the look-at point is in the center of the viewing area.

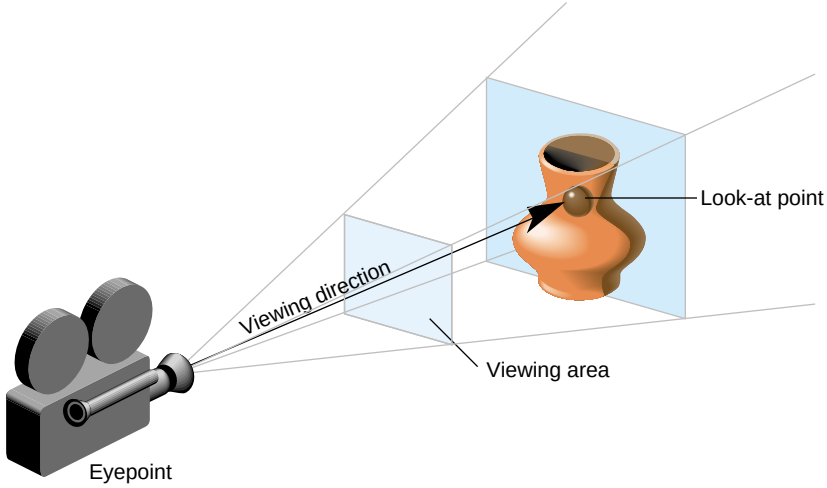


Figure 13-1 The Camera Analogy in 3D Viewing

3D Viewing Functions

In the context of this document, *viewing* refers to manipulating a camera to view the contents of a 3D application (see "Introduction to 3D Viewing"). This document distinguishes between the two basic viewing modes, inspection and navigation. Every 3D application needs to support at least one; if your application supports both, pick one as the primary mode.

Although viewing refers to manipulating a camera in a 3D application, users may base their interaction with the application on a different metaphor. These fundamental metaphors are also discussed in the following sections. For example, during inspection users interact with the scene (or object) they are viewing as if it were a single object that they are holding in their hand. They expect to be able to move this scene (or object) around in space (see "3D Viewing Trade-Offs and Related Guidelines"). Application developers, on the other hand, find it useful to implement the inspection functions in terms of a camera that moves around the scene being viewed. It's important that your application allows users to work with viewing functions using the metaphors they expect regardless of how the application implements them.

This section describes the different functions available in inspection and in navigation. Table 13-1 provides an overview of each function, the mouse and key bindings used to access it, and the pointer shape displayed when the user is accessing it. Each function is discussed in detail in the following sections.

Table 13-1 3D Viewing Functions and User Interface

Function	View Mode	Pointer	Mouse and Keyboard Binding
Tumbling	Inspection (default)		Dragging with the left mouse button.
Dollying	Inspection		Dragging while simultaneously pressing the left and middle mouse buttons.
Panning	Inspection		Dragging with the middle mouse button.
Roaming	Navigation (default)		Dragging with the left mouse button.
Tilting	Navigation		Dragging while simultaneously pressing the left and middle mouse buttons.
Sidling	Navigation		Dragging with the middle mouse button.
Seeking	Inspection and Navigation		Clicking with the left mouse button.[a]

a. Many applications need to reserve clicking with the left mouse button for a more useful function (for example, activating a link or initiating object behavior). In those applications, allow users to first activate a seek tool, then click in the scene with the left mouse button to seek.

Inspection Functions for 3D Viewing

This section first gives an overview of inspection, then describes three viewing functions that apply only to inspection (and not to navigation):

- "Tumbling"
- "Dollying"
- "Panning"

The section also discusses "Seeking," which has the same effect in both inspection and navigation.

Each function is first presented from the user's point of view, then discussed in terms of the implementation model.

Inspection Overview

Inspection is an approach to viewing where users can examine a scene as if it's a single object they are holding in their hand. For example, users may want to examine the model of a coffee mug the same way they would examine a real mug by holding it and turning it around.

The expected user model for inspection is that users are manipulating the scene, not the camera. From the users' perspective, all inspection controls appear to manipulate the scene (or object) while the camera remains stationary. For example:

- Pressing the left mouse button and dragging the pointer down (tumbling) rotates the object towards the user. To achieve this, the application actually moves the camera up over the object (see Figure 13-2).
- Pressing the middle mouse button and dragging the pointer to the left (panning) moves the scene toward the left of the viewing window. To achieve this, the application moves the camera to the right (see Figure 13-5).

Note that in both examples, users move the control (and pointer) in the direction they want the scene (or object) to move. To achieve this, the application moves the camera in the opposite direction of the control (and pointer).

Table 13-2 provides an overview of the different functions available in inspection.

Table 13-2 Overview of Inspection Viewing Functions

Function	User Model	Implementation Model
Tumbling	User holds object and rotates it to view it from all sides and angles.	Camera (eyepoint) moves around a fixed look-at point on a spherical course. Camera moves opposite to direction of user's action.
Dollying	User moves object closer or farther away.	Camera (eyepoint) moves toward a fixed look-at point to move object closer and moves away from look-at point to move object farther away. Viewing direction remains unchanged.
Panning	User moves object up, down, left, or right in viewing window.	Camera (eyepoint) moves in plane perpendicular to viewing direction. Camera moves opposite to movement of object. Viewing direction is unchanged. Look-at point moves with camera.
Seeking	User selects object (or part of object). Selected object is centered in viewing window and moved closer to user with each click.	Look-at point moves to where user clicked in the scene. Camera (eyepoint) turns so that look-at point is centered in viewing window. Camera moves closer by half the original distance between camera and object.

Tumbling

Tumbling is the default viewing function for inspection. Users rotate a model of an object or a scene as if they were holding it in their hand. Users expect to be able to tumble the object in all three dimensions around the fixed look-at point. Tumbling doesn't change the location of the object in space.

The user controls tumbling by dragging with the left mouse button. The movement follows a virtual trackball imposed on the viewing window. The initial position of the pointer on this virtual trackball influences the tumbling behavior: If the user positions the pointer in the center of the viewing window (trackball) and drags horizontally (or vertically), the object tumbles around the y axis (or x axis). If the user positions the pointer in the center of the viewing window and drags out in any direction, the object tumbles around an axis perpendicular to the drag. If the user drags in a circle around the center of the virtual trackball, the object tumbles around the z axis. If the user drags beyond the limits of the trackball, the object continues to tumble until the user releases the mouse button.

Figure 13-2 illustrates tumbling from the implementation perspective. The camera (eyepoint) moves around the scene as though the camera were placed on the surface of a sphere. The look-at point remains stationary at the center of the sphere. The camera moves opposite to the direction of the rotation. Using the original camera position, the user can look into the pot but can't see much of the pot's outside surface. After the user has tumbled the bottom of the pot upwards, it's possible to see the pot's surface. To accomplish this rotation, the eyepoint (camera) moves down along the surface of the sphere and the look-at point remains stationary.

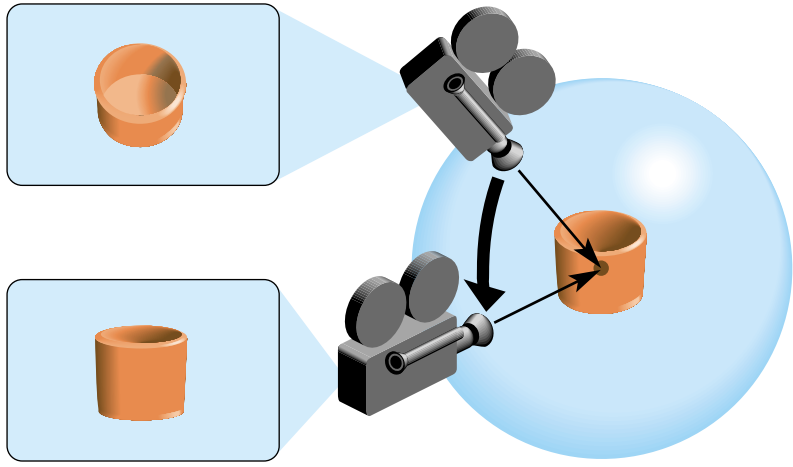


Figure 13-2 Schematic Illustration of Tumbling (Implementation Perspective)

Dollying

Dollying allows users to move a model of an object or a scene closer or farther away. Users move the object as if they were holding it in their hand. The user controls dollying by dragging while simultaneously pressing the left and middle mouse buttons during inspection. Dragging down in the viewing window moves the object closer; dragging up moves it farther away.

Figure 13-3 illustrates dollying from the implementation perspective. The look-at point and viewing direction are fixed. The camera (eyepoint) moves toward the look-at point along the viewing direction to move the object closer to the user. If the user wants to move the object further away, the camera would move away from the look-at point.

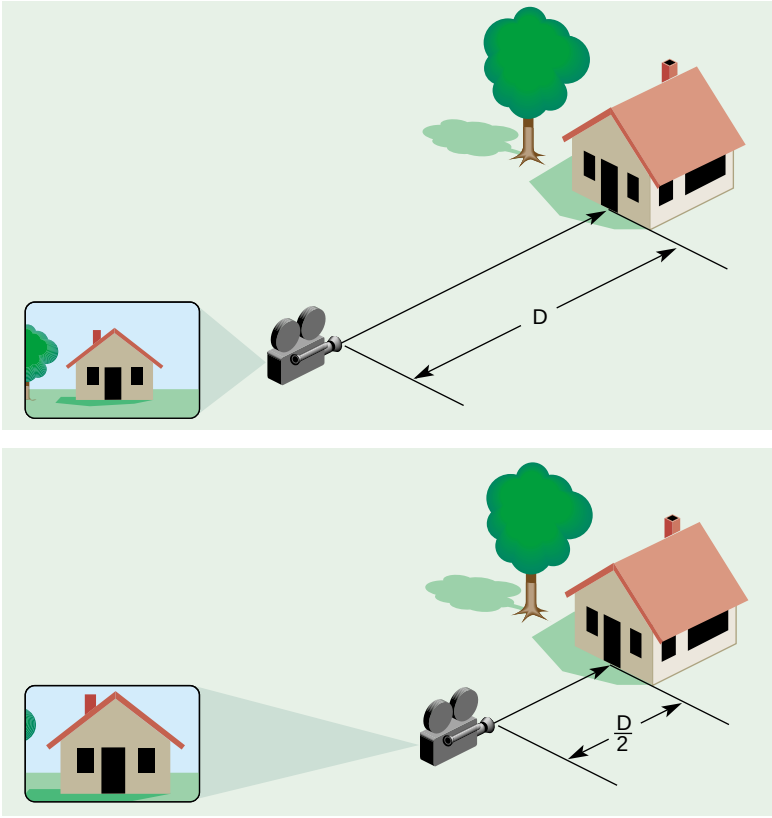


Figure 13-3 Schematic Illustration of Dollying (Implementation Perspective)

Note: When users move closer to an object, they eventually reach the point where they would touch the object. As they dolly even farther forward, the eyepoint moves past the (fixed) look-at point as if the user just moved through the object. When the user passes this point, the tumble controls are "reversed" because the user is now dragging along the inside of the virtual trackball (see "Tumbling") and along its backside.

Dollying is different from zooming. In both cases the objects change in size in the viewing window. However, in contrast to dollying, zooming doesn't move the object closer or farther away from the user. Zooming instead allows users to change the viewing angle of the camera, the same way they would use a zoom lens on an actual camera. That is, as the user zooms out the viewing angle is increased so that the viewing area becomes larger and more of the scene is visible. This new larger viewing area is then mapped to the viewing window.

As shown in Figure 13-4, objects appear larger (or smaller) after zooming in (or out) even though the location of the camera hasn't changed. This is because the viewing angle changes but the size of the viewing window hasn't changed.

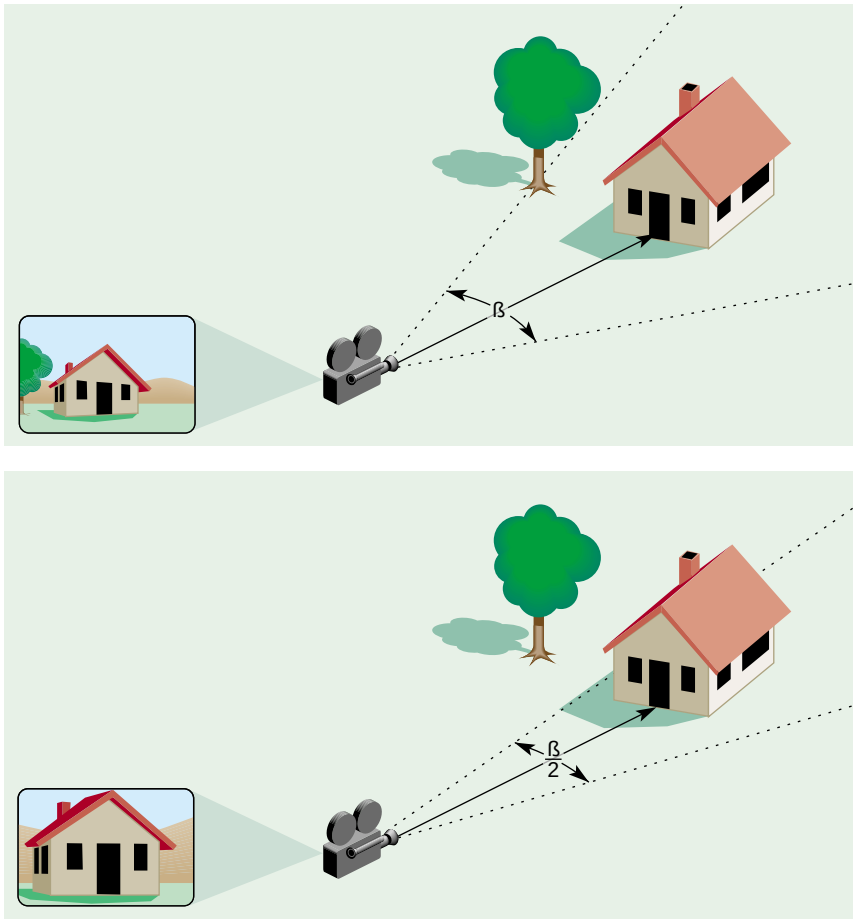


Figure 13-4 Schematic Illustration of Zooming (Implementation Perspective)

Panning

Panning allows users to move a model of an object or a scene up, down, left, or right in the viewing window. Users move the object as if they were holding it in their hand. The user controls panning by dragging while pressing the middle mouse button. The object moves in the direction of the drag; for example, dragging up in the viewing window moves the object up and dragging left moves the object left.

Figure 13-5 illustrates panning from the implementation perspective. The camera (eyepoint) moves in the plane perpendicular to the viewing direction. The camera moves opposite to the movement of the object. As shown in the figure, the camera moves right, which moves the scene to the left in the viewing window. The look-at point moves with the camera. The viewing direction is unchanged.

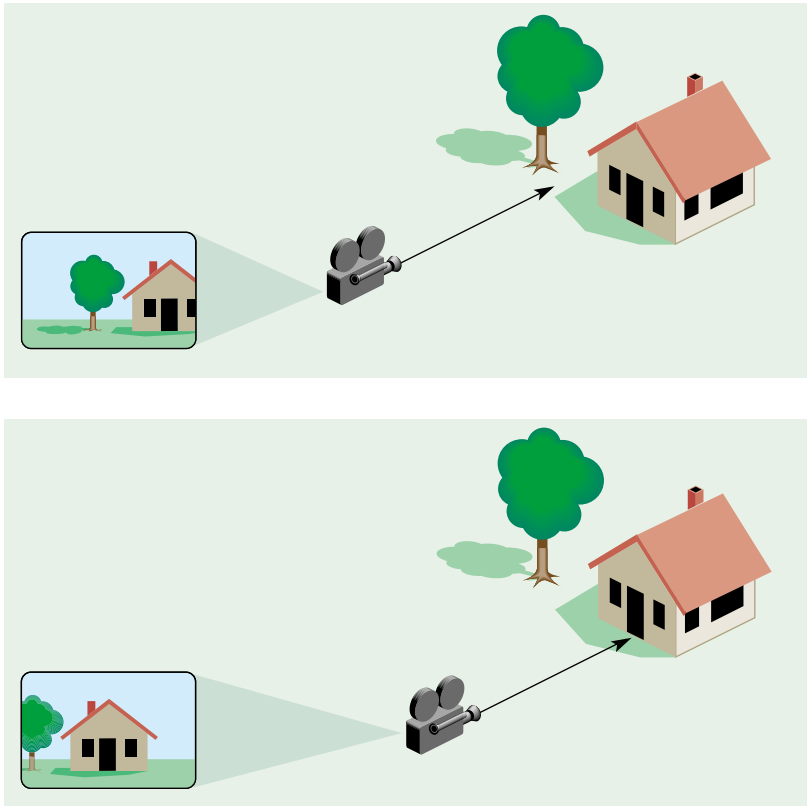


Figure 13-5 Schematic Illustration of Panning (User Drags Right)

Seeking

Seeking allows users to move an object in the scene into the center of the viewing window. In inspection, the user model is that the user is incrementally moving the object closer (see “Inspection Overview”). In navigation, the user model is that the user is incrementally moving closer to the object (see “Navigation Overview”).

For both inspection and navigation, the user controls seeking by clicking on the object (or part of the object) of interest. Clicking on the object centers the object (or part) in the viewing window and brings the object and user closer together. Each additional click on the same object (or part) brings the object and user still closer. Figure 13-6 shows a simple example of seeking to the door of a house. The first click on the door positions the door in the center of the viewing window and halves the distance between the door and the user. The second click halves the distance again.

Many applications need to reserve clicking with the left mouse button for a more critical or useful function (for example, activating a link or initiating object behavior). In those applications, allow users to first activate a seek tool, then click with the left mouse button in the scene to actually seek.

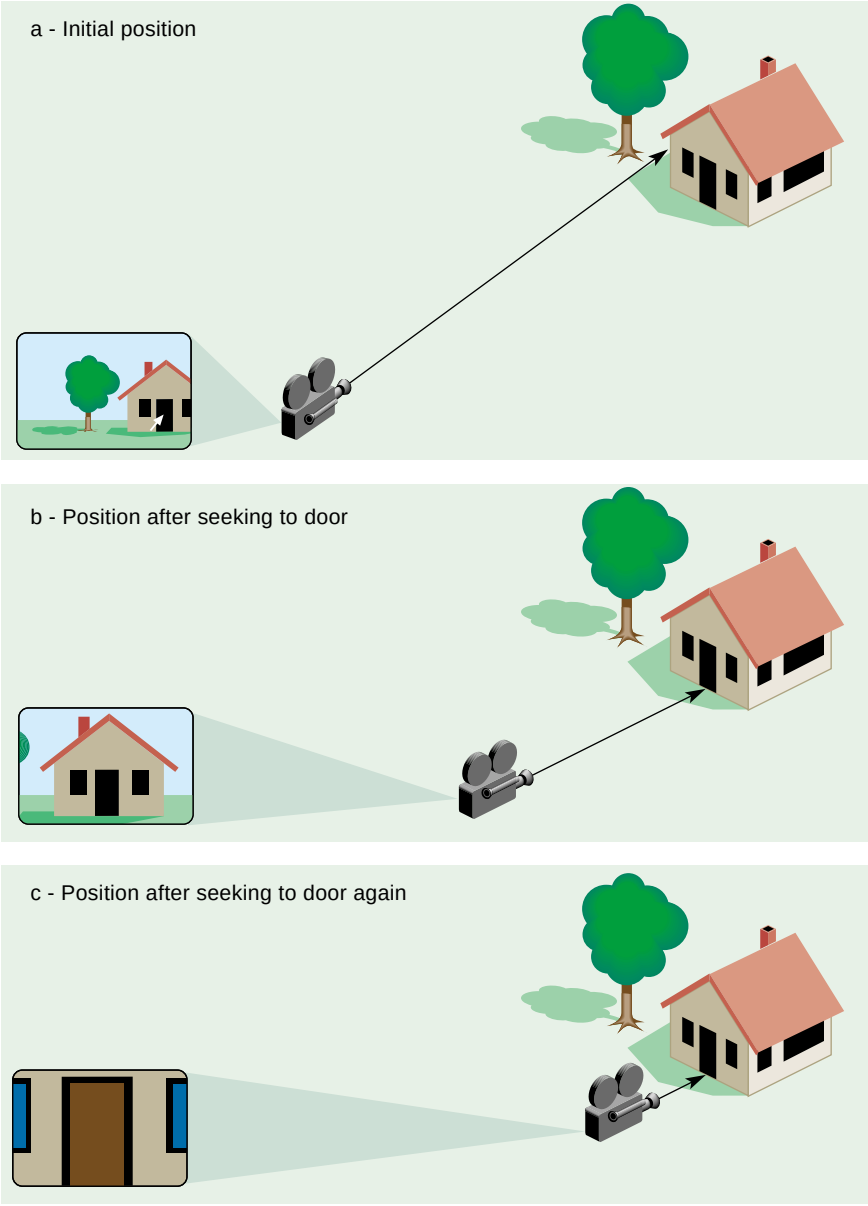


Figure 13-6 Simple Example of Seeking to Door

From the implementation perspective, seeking sets a new look-at point at the location of the user's click and moves the camera so that this new look-at point is at the center of the viewing window. The camera is also moved forward half the original distance between the camera and the object. Note that in the case of inspection, resetting the look-at point by seeking means that the camera tumbles around this new point after the seeking action.

Navigation Functions for 3D Viewing

This section first gives an overview of navigation, then describes three viewing functions that apply only to navigation (and not to inspection):

- "Roaming"
- "Tilting"
- "Sidling"

"Seeking," which has the same effect in both inspection and navigation, is discussed in the preceding section.

Each function is first presented from the user's point of view, then discussed in terms of the implementation model.

Navigation Overview

Navigation is useful when users want to move through a world, for example, walk through a 3D model of a museum or an architectural model. In navigation, the user maneuvers through a fixed, immovable world by walking, flying, or another navigation mechanism.

The expected user model for navigation is that users are manipulating the camera. From the users' perspective, all navigation controls appear to manipulate the camera while the scene remains stationary. For example:

- Pressing the left mouse button and dragging the pointer up while roaming moves the user farther forward into the scene. To achieve this, the application also moves the camera farther into the scene (see Figure 13-7).
- Pressing the middle mouse button and dragging the pointer to the left while sidling sidesteps the user towards the left of the viewing window. To achieve this, the application also moves the camera to the left (see Figure 13-9).

Note that in both examples, users move the control (and pointer) in the direction they want the camera to move.

Table 13-3 provides an overview of the different functions available in navigation.

Table 13-3 Overview of Navigation Viewing Functions

Function	User Model	Implementation Model
Roaming	User moves forward or backward in the scene. Turning changes direction of movement.	Camera (eyepoint) moves forward or backward along viewing direction in the same direction as the user action. Viewing direction moves in the direction that the user turns. Look-at point changes as the viewing direction changes.
Tilting	User looks up or down.	Viewing direction moves in the direction that the user looks (up or down). Position of camera (eyepoint) remains fixed. Look-at point changes as the viewing direction changes.
Sidling	User sidesteps left or right in the scene or "elevators" up or down in the scene.	Camera (eyepoint) moves in plane perpendicular to viewing direction. Camera moves in the same direction as user action. Viewing direction remains unchanged. Look-at point moves with camera.
Seeking	User selects object (or part of object). Selected object is centered in viewing window and moved closer to user with each click.	Look-at point moves to where user clicked in the scene. Camera (eyepoint) turns so that look-at point is centered in viewing window. Camera moves closer by half the original distance between camera and object.

Roaming

Roaming (and turning) is the default viewing function for navigation. Users move through a fixed scene as if walking through it. While users are moving they expect to be able to turn to change the direction of the movement. Users control roaming by dragging with the left mouse button while the application is in view mode. Since users may sometimes want to turn without moving, dragging on the horizontal is interpreted differently than dragging in other directions as follows:

- Dragging up in the viewing window moves the user forward into the scene; dragging down moves the user backwards out of the scene.
- Dragging directly left on the horizontal in the viewing window turns the user left without any forward or backward movement; dragging directly right turns the user right without any movement.
- Dragging in any direction above the horizontal both turns the user in that direction and moves the user forward in that direction; dragging in any direction below the horizontal both turns the user and moves the user backward in that direction.

From the implementation perspective, the camera (eyepoint) moves forward or backward along the viewing direction in the same direction as the user's action; that is, as the user moves forward, the camera moves forward (see Figure 13-7). The viewing direction moves in the same direction that the user turns; that is, as the user turns left, the viewing direction rotates left. If the user indicates a wish to turn but not move (by dragging the pointer directly left or right on the horizontal), the viewing direction changes appropriately but the camera doesn't move forward or backward. The look-at point changes as the viewing direction changes.

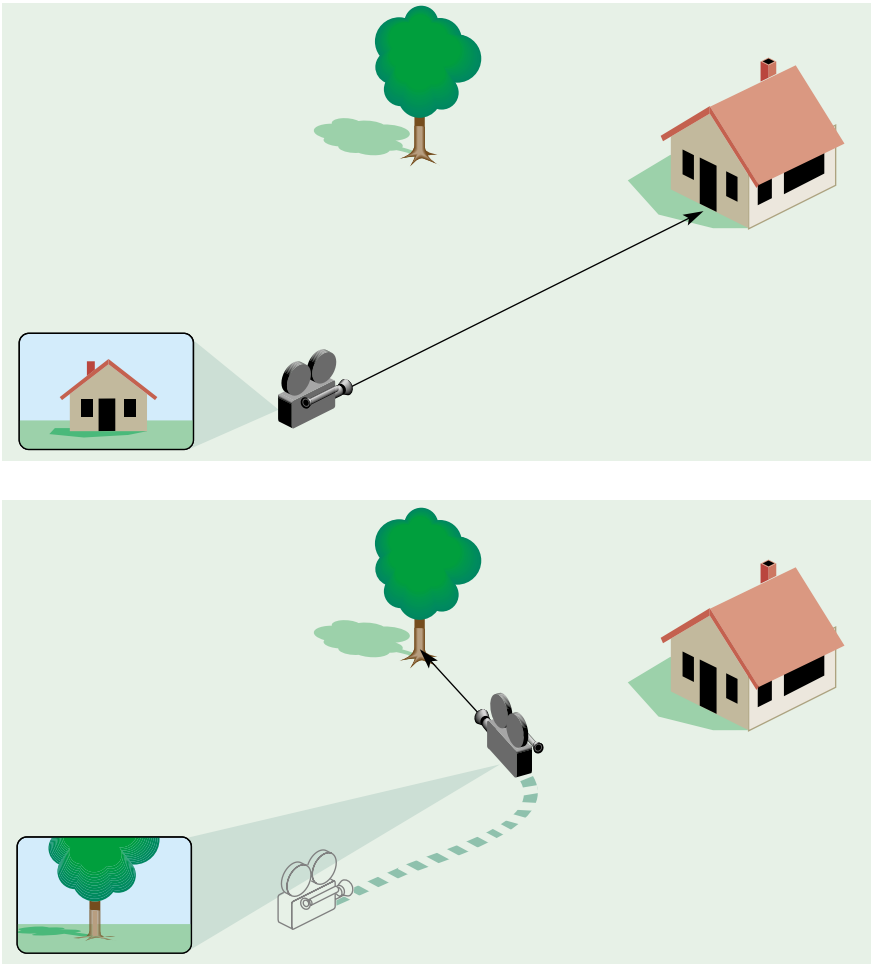


Figure 13-7 Schematic illustration of Roaming (Implementation Perspective)

Tilting

Tilting allows users to look up and down to see an object higher or lower than their current viewing direction in the scene. Tilting doesn't move the user. To move toward an object in the new view, the user has to use roaming (see "Roaming"). To control tilting, the user simultaneously presses the left and middle mouse buttons and drags. Dragging up in the viewing window tilts the user's head up to look up in the scene; dragging down allows the user to look down.

From the implementation perspective, tilting changes the viewing direction in the same direction the user's head is tilted (see Figure 13-8). As the user looks up, the viewing direction moves up; looking down moves the viewing direction down. The location of the camera (eyepoint) doesn't change. The location of the look-at point changes as the viewing direction changes.

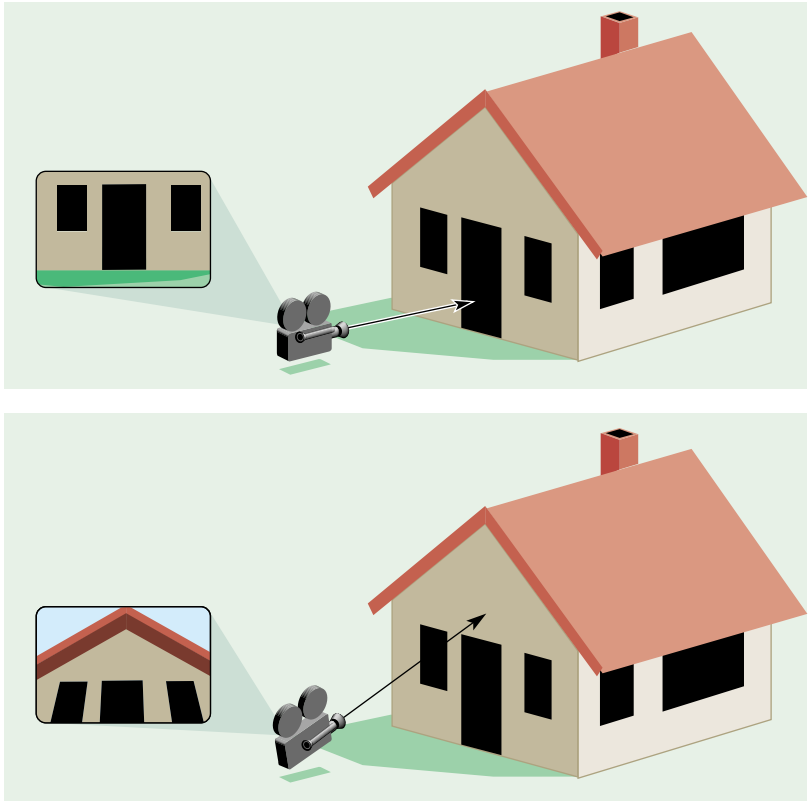


Figure 13-8 Schematic Illustration of Tilting (Implementation Perspective).

Sidling

Sidling allows users to sidestep left and right in the scene or to "elevator" up and down in the scene. Sidling moves the user left, right, up and down in the plane perpendicular to the viewing direction; it doesn't move the user forward or back in the scene. The user controls sidling by dragging while pressing the middle mouse button. The user moves in the direction of the drag; for example, the user drags left in the viewing window to sidestep to the left. Dragging up moves the user up as if riding on an elevator.

Figure 13-9 illustrates sidling from the implementation perspective. The camera (eyepoint) moves in the plane perpendicular to the viewing direction. The camera moves in the same direction that the user wants to move. As the user sidesteps left, the camera moves left. If the user moves up, the camera also moves up. The orientation of the camera remains unchanged. The look-at point moves with the camera.

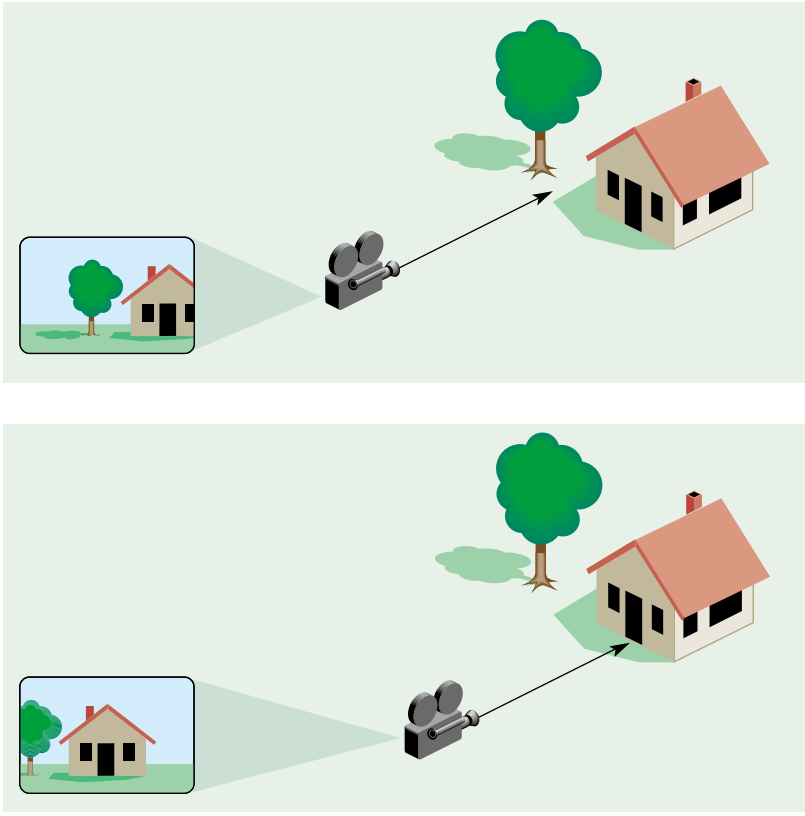


Figure 13-9 Schematic Illustration of Sidling (User Drags Left)

Guidelines for 3D Viewing Functions

When designing the user interface for a 3D application...

- ❑ Provide a viewing interface regardless of other capabilities of the application (for example, editing).

When designing the user interface for 3D viewing...

- ❑ Decide whether your application will support inspection, navigation, or both, then provide the appropriate viewing functions. If your application supports both inspection and navigation, choose one as the primary mode for viewing.
- ❑ Use standard pointer shapes to indicate the current 3D viewing function.

When designing the user interface for INSPECTION in a 3D application...

- ❑ Support the user model that users are manipulating a scene as though it were a single object they are holding in their hand (not the user model that users are manipulating a camera). From the user's perspective, all controls appear to manipulate the object or scene while the camera remains stationary.
- ❑ Support tumbling as the default inspection function to allow users to view all sides of the scene.
- ❑ Assign tumbling to dragging with the left mouse button.
- ❑ Display the tumble pointer while the user accesses the tumble function.
- ❑ Support dollying to allow users to move the scene closer or farther away.
- ❑ Assign dollying to dragging with the left and middle mouse buttons pressed simultaneously.
- ❑ Display the dolly pointer while the user accesses the dolly function.
- ❑ Support panning to allow users to move the scene left, right, up, or down.
- ❑ Assign panning to dragging with the middle mouse button.
- ❑ Display the pan pointer while the user accesses the panning function.
- ❑ Support seeking to allow users to change the look-at point and center the object of interest and to bring the object incrementally closer.

❑ Support seeking as follows:

- If your application needs to reserve clicking with the left mouse button for a more critical or useful function, allow users to seek by first activating a seek tool, then clicking with the left mouse button in the scene. Otherwise, support seeking without the use of a tool.
- In either case, the user seeks by clicking on a part of the scene with the left mouse button. The application centers that part of the scene in the viewing window and moves the scene closer by half the distance between the camera and the object.
- With each subsequent click on the same part of the scene, the scene again moves closer.

❑ Display the seek pointer while the user accesses the seek function.

When designing the user interface for NAVIGATION in a 3D application...

❑ Support the user model that the scene is stationary and the user is moving through this fixed, immovable world. From the user's perspective, all navigation controls appear to manipulate the camera (user's view into the world) while the scene remains stationary.

❑ Support roaming as the default navigation function. In roaming, the user can move forward and backward, turn left and right, and turn while moving.

❑ Assign roaming to dragging with the left mouse button.

❑ Display the roam pointer while the user accesses the roaming function.

❑ Support tilting to allow users to change their view of the scene by tilting their head up and down. Tilting doesn't move the user forward or backward.

❑ Assign tilting to dragging with the left and middle mouse buttons pressed simultaneously.

❑ Display the tilt pointer while the user accesses the tilting function.

❑ Support sidling to allows users to sidestep left and right and to move up and down as if on an elevator.

❑ Assign sidling to dragging with the middle mouse button.

❑ Display the sidle pointer while the user accesses the sidling function,

❑ Support seeking to allow users to move closer to an object in the scene.

❑ Support seeking as follows:

- If your application needs to reserve clicking with the left mouse button for a more critical or useful function, allow users to seek by first activating a seek tool, then clicking with the left mouse button in the scene. Otherwise, support seeking without the use of a tool.
- In either case, the user seeks by clicking on a part of the scene with the left mouse button. The application centers that part of the scene in the viewing window and moves the scene closer by half the distance between the camera and the object.
- With each subsequent click on the same part of the scene, the scene again moves closer.

❑ Display the seek pointer while the user accesses the seek function.

3D Viewing Interface Trade-Offs

When designing a user interface for viewing in a 3D application, developers often need to address the design issues discussed in this section:

- "Viewing and Editing in 3D Applications"
- "Single-Viewport and Multi-Viewport Viewing in 3D Applications"
- "3D Viewing Performance and Scene Fidelity"

Viewing and Editing in 3D Applications

Only a limited number of mouse and keyboard key combinations is available for interacting with an application. Users therefore can't easily have access to all necessary editing and viewing functions at the same time. Instead, they need to switch contexts between editing and viewing so that they can use the same mouse and keyboard combinations in the different contexts to access different functions.

This context switch is best done by splitting editing and viewing functionality into two separate explicit modes. Using explicit modes avoids a potentially confusing interface that may result if the user doesn't know whether the next action will change the view of the object or the object itself.

In general, when users work with an application that allows editing, they like to be offered several ways to access the viewing functions, and they like to always have quick access to these functions.

The following sections discuss several techniques for providing both viewing and editing capabilities to the user:

- "Separate View and Edit Modes"
- "View Overlay"
- "Viewing Controls"
- "Dedicated Viewing Peripheral Devices"

No matter how an application allows users access to viewing and editing, it's important to always display the correct pointer shape to let users know which function they are currently performing. See "Pointer Shapes for 3D Functions" in Chapter 12.

Separate View and Edit Modes

If an application supports editing, separate and explicit view and edit modes are highly recommended. This allows more flexibility in assigning functions to mouse and keyboard key combinations. In edit mode, mouse and keyboard input perform editing functions on selected objects and on the scene; in view mode, mouse and keyboard input perform viewing functions.

Users expect an obvious mechanism to switch modes, for example an item in a pull-down menu or a button on a tool palette that provides a variety of possible modes. In addition, users also expect to be able to switch modes using the <Esc> key (see "Using Modifier Keys in 3D Applications" in Chapter 12). Pressing this key takes the user to the next mode.

View Overlay

When they are editing, users expect to always have quick access to viewing with a view overlay. A view overlay is a temporary view mode that's available while the user holds down the <Alt> key (see "Using Modifier Keys in 3D Applications" in Chapter 12). As long as the <Alt> key remains pressed, mouse and keyboard input is temporarily interpreted as providing viewing input rather than editing input. Releasing the <Alt> key returns the application to standard editing operations. If the application is already in view mode when the user presses the <Alt> key, the <Alt> key is ignored.

A view overlay offers users quick access to temporary viewing but allows them to stay focused on the editing tasks at hand. This avoids forcing the user to make a heavyweight switch between edit and view modes. Although the view overlay is temporary, users still need to see the correct pointer shape feedback while accessing the viewing functions (for example, the roam pointer or tilt pointer). See "Pointer Shapes for 3D Functions" in Chapter 12.

Viewing Controls

Applications can optionally provide separate user interface controls to access viewing functions. In this approach, all mouse input is interpreted as editing input unless the user is using the mouse pointer to manipulate a viewing control.

Figure 13-10 shows an application window with viewing controls around the sides and the bottom of the window. Manipulating the thumbwheels or sliders with the mouse affects viewing: For example, dragging the thumbwheel in the lower right hand corner of the window dollies the camera, which changes the view but doesn't edit it. Using the mouse in the viewing area of the window performs editing actions: for example, clicking on the star selects that object for editing.

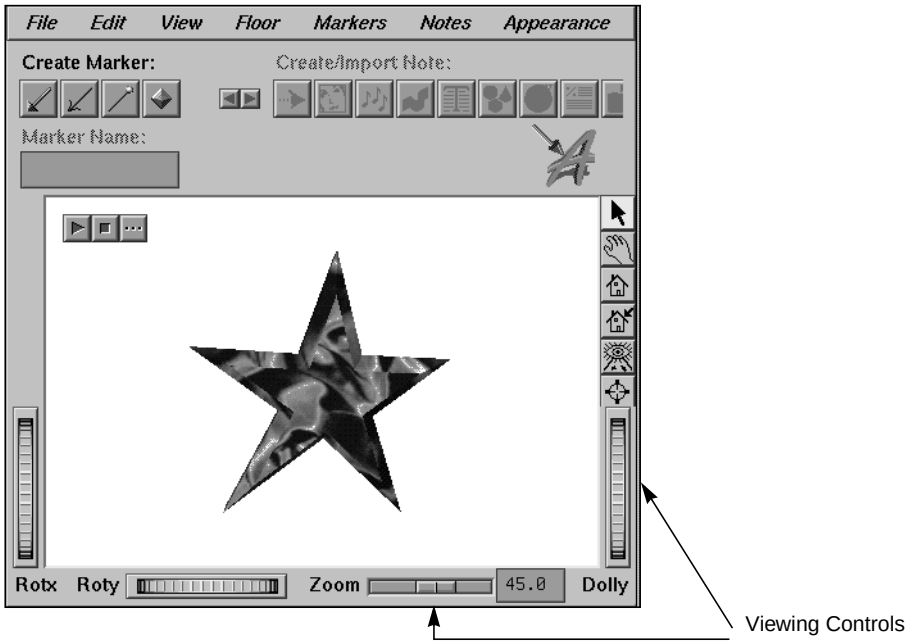


Figure 13-10 Application With Viewing Controls

Dedicated Viewing Peripheral Devices

Another optional method of addressing the conflict between viewing and editing input is to assign all viewing actions to one dedicated input device, such as a spaceball. All input from the dedicated input device performs viewing functions; input from other devices performs editing functions. This approach provides more input bandwidth: Context switching between viewing and editing is handled by the choice of input device.

Single-Viewport and Multi-Viewport Viewing in 3D Applications

When designing a viewing interface, you must decide whether to offer users only one view of the scene (single-viewport) or multiple views simultaneously (multi-viewport). Multiple views may be, for example, one close-up and one distance view or one view from the top and one from each side. This section presents "Single-Viewport Viewing" and "Multi-Viewport Viewing," discussing their advantages and disadvantages.

Single-Viewport Viewing

In the single-viewport model, only one view of the scene can be projected to the single viewport at any given time, even if there are multiple cameras in the scene. This is a serially multiplexed approach; different views are presented one after another in the same viewport and the user can switch among them.

By default, the viewport provides a perspective view of the scene. The view updates as the user selects different cameras.

Single-viewport viewing has these advantages:

- **Performance**—Updating the contents of one view is less computationally expensive than updating two or more views. Application performance deteriorates as the number of views increases, so single-viewport viewing is faster than multi-viewport viewing.
- **Space**—The view doesn't need to share space with other views in the application window. The total viewing area is dedicated to a single view; this allows the largest possible representation of the 3D data.
- **Simpler user model**—Users have to deal only with one view and one window. In contrast, a multi-viewport model requires that users determine the relationship among the different views or decide how changes in one view influence the other views.

Multi-Viewport Viewing

In the multi-viewport model, two or more views of a scene are simultaneously available. Typically, there are four views: front, top, one side (typically the right), and perspective.

A view isn't necessarily bound to a particular camera. For each view, the user can choose which camera to use and what each camera views. For example, to view an object from the bottom that's currently visible from the front, the user can either find a camera that displays it from the bottom or tumble or roam to get that view.

Multi-viewport viewing has the advantage that it allows simultaneous views of different representations of data. Users can examine and edit data from different perspectives simultaneously and can edit and examine data across multiple views without having to switch views. This is important for editing complex objects or during scene composition. While performance can be worse with multiple views (because more windows must be updated during viewing operations), experienced users find multiple views useful because they can coordinate operations across multiple viewports to get more accurate feedback on the actions they are performing.

3D Viewing Performance and Scene Fidelity

Viewing is critical to interacting with 3D environments and applications. The more responsive the application is during viewing, the more realistic and compelling the user's experience.

To achieve realistic user interaction, an application has to maintain at least 8 fps while the user interacts with the view. The frame rate—number of frames per second (fps)—is a good gauge of acceptable viewing performance:

- If the frame rate drops below 8 fps, users typically find interacting with the application cumbersome.
- In an editing context, 10-12 fps can be sufficient.
- 15 fps is the minimum frame rate to give the user a fluid, in-control experience. Action games or immersive experiences may require a greater frame rate to achieve that goal.

Some 3D scenes are so complex that just rotating the view becomes computationally expensive. In that case, the 3D scene can't be rendered at an acceptable frame rate. In such situations, applications must provide automatic adaptive rendering, user-controlled adaptive rendering, or both:

- In automatic adaptive rendering, the application always maintains viewing responsiveness at the expense of scene fidelity.
- In user-controlled adaptive rendering, users explicitly choose between adaptive rendering (that is, maintaining viewing responsiveness at the expense of scene fidelity) and fully rendering the contents of the scene (but taking a performance hit during viewing). This choice is important if users sometimes need fully rendered, high-fidelity scenes and, therefore, need to turn off adaptive rendering.

Note: It isn't acceptable to let the frame rate drop below 8 fps without explicit user confirmation.

Adaptive rendering maintains viewing performance by changing the rendering characteristics of objects and elements during viewing operations. Typically, some detail is omitted from the display to reduce the computational requirements. As a result, a higher frame rate is achieved at a somewhat lower level of fidelity. Once viewing stops, the scene is returned to its original fidelity. Most users are satisfied with such a trade-off. Without adaptive rendering, users complain of poor performance or sluggishness. Adaptive rendering maintains responsive behavior without reducing functionality or impeding user tasks.

To implement adaptive rendering, an application can use techniques such as turning off texturing when an object is being moved, or using wireframe models. If an application has multiple views, adaptive rendering can be implemented by updating only one of the views. Then, when the view is no longer changing, the other views can be updated.

Note that if an application uses only automatic adaptive rendering, it needs to provide users easy access to fully rendered scenes. At a minimum, this should occur when the user stops interacting with the view.

3D Viewing Trade-Offs and Related Guidelines

To make viewing quickly and easily accessible in 3D applications...

❑ Always provide ready access to viewing no matter what the user is doing (for example editing).

When designing a viewing interface for a 3D application that also supports editing...

❑ Display the appropriate pointer depending on the task the user is performing:

- While the user is accessing editing functions, display the edit pointer.
- While the user is accessing viewing functions, display the appropriate view pointer based on the user's current viewing function (for example, the roaming pointer if the user is currently navigating a scene).

❑ Provide a modal interface to viewing and editing whenever possible.

❑ Provide an obvious mechanism for changing between the view and edit modes, such as buttons in a tool palette or entries in a pull-down menu.

❑ Reserve the <Esc> key for switching between the view and edit modes.

❑ Always provide a view overlay for quick access to viewing. That is, when the primary task is editing, the user can at any time temporarily enter a view mode by pressing and holding the <Alt> key. The user can release the <Alt> key to return the application to edit mode.

❑ Reserve the <Alt> key for providing access to a view overlay. If the user is already in view mode, the <Alt> key has no effect.

❑ Display the appropriate pointer for the current viewing function (for example, the tumble pointer or the roaming pointer) while the user is accessing a view overlay.

❑ Optionally provide additional ways to access viewing, for example, offer viewing fixtures or split viewing and editing input across separate dedicated input devices.

When deciding between a single viewport and multiple viewports...

- ❑ Use a single viewport if the user doesn't need to do much editing, performance or screen real estate is critical, you need a simple user model, or if several of these conditions are met.
- ❑ Support multiple viewports if the user needs two or more views of the data simultaneously (such as when editing complex objects or working on scene composition) and performance isn't a critical issue.

When designing a viewing interface for a single viewport...

- ❑ Use the perspective view of the scene as the default view.
- ❑ Update the single-viewport view with a new view as the user selects different cameras.

When making viewing performance design decisions...

- ❑ Support a minimum frame rate of 8 fps when the user is interacting with the view.
- ❑ Ideally, support a minimum rate of 10-12 fps for editing and a minimum frame rate of 15 fps for a realistic interactive experience.
- ❑ If the frame rate drops below 8 fps, provide at least one of the following solutions:

- Automatic adaptive rendering, where the application always maintains an acceptable frame rate at the expense of scene fidelity.
- User-controlled adaptive rendering, where the user explicitly chooses between adaptive rendering (acceptable frame rate but loss of detail) and fully rendering the contents of the scene (at a possibly unacceptably low frame rate).

- ❑ If users sometimes need fully rendered, high-fidelity scenes and the frame rate is likely to drop below 8 fps, provide user-controlled adaptive rendering.
- ❑ If you application only provides automatic adaptive rendering, provide users ready access to fully rendered scenes. At a minimum, this should happen when the user stops interacting with the view.

Selection in 3D Applications

Any 3D application that offers more than simple viewing requires a selection mechanism. Just like 2D applications, 3D applications need to allow users to select one or more objects. This chapter discusses object selection in 3D applications and extends and complements the guidelines for selection in 2D applications (see "Selection" in Chapter 7). It includes these topics:

- "3D Selection Concepts and Models" briefly summarizes the object-action paradigm, explains two techniques for selecting objects (direct and indirect selection), and discusses how the OSF/Motif selection models apply to 3D applications.
- "Selection Feedback for 3D Objects" discusses user feedback to indicate an object can be selected and to indicate an object is currently selected.
- "Lead Objects in 3D Applications" explains how to use the concept of a lead object to determine how to apply specific actions when multiple objects are selected.

3D Selection Concepts and Models

This section first discusses the object-action paradigm and then contrasts two techniques for selecting objects: direct selection and indirect selection. It then presents the selection models and minimum selection actions users expect in a 3D application. These techniques extend the 2D-oriented OSF/Motif guidelines for selection (see "Selection" in Chapter 7).

The Object-Action Paradigm in 3D Applications

In the object-action paradigm, the user first selects an object or a group of objects and then applies an action to the objects in the selection (see "Selection" in Chapter 7 for details). Object selection is the prerequisite to other operations. As a shortcut, selecting and manipulating the object can sometimes be combined (see "Drag and Drop for Non-Text Objects" in Chapter 7, "Focus, Selection, and Drag and Drop"). For example, users may move the pointer over an object, press and hold the mouse button, and thereby select and drag the object at the same time.

Each 3D application must have a current selection to which the action is applied at any time. The current selection may be empty (that is, no selection). Note that if your application supports several viewports, each window can maintain a separate selection, but there is only one current selection.

Some applications use an action-object paradigm instead: The user selects an action, then chooses the object to which the action should apply, and exits by returning to the selection rule (neutral state). This paradigm can be useful for tools that attach to the pointing device. In most cases, however, a user interface using an action-object paradigm tends to be excessively modal and cumbersome to use, and this paradigm isn't recommended.

Direct Selection in 3D Applications

Direct selection means that the user selects one or more objects by selecting the objects themselves, not an indirect representation of them (for example, the name of an object in a list). Because users of 2D applications are accustomed to direct selection, using direct selection in 3D applications makes these applications easier to learn. Direct selection is therefore the preferred primary selection mechanism.

Direct selection can be used to select one or more objects, as follows:

- To select a single object, the user moves the pointer over an object and presses the left mouse button. As the user clicks, the application determines which object is selected by following a ray projected into the scene. The first object the ray intersects is selected.
- To add objects to the current selection (or remove them from the current selection), the user holds the <Ctrl> key while clicking on individual objects. Objects which are not currently selected become selected, while currently selected objects are deselected.

- To select several objects at the same time, the user presses the left mouse button and drags around a group of objects. When the user releases the mouse button, all objects inside the drag area are selected. This is referred to as sweep selection. There are two variants of this technique:
  - **Rectangle-drag**. The user drags the pointer to define the diagonal of a rectangular area on the screen. All objects inside this rectangle are selected.
  - **Lasso-drag**. The user draws out an irregularly shaped area with the pointer. All objects inside this area are selected.

  In both cases, the initial mouse-down action doesn't make a selection but sets the point of origin for the area to be selected. When the user releases the mouse button to complete the drag operation, the selected region is defined by the boundaries of the drag and extends from the user's current z position back to the full length of the viewing frustum. All objects, completely visible or partly obscured, that fall in this region are considered to be in the selected region.

Note that some applications may need to provide sub-object selection methods to allow users to select vertices, edges, and faces. This isn't discussed in this document.

Indirect Selection in 3D Applications

In indirect selection, the user selects an object by selecting an indirect representation of it. For example, assume that the names of all the objects in a 3D scene appear in a scrolling list and the user can click on a name in the list to select the object. Other examples are using a "Select All" menu command, which allows users to select all objects in a 3D scene, and selecting up and down a text representation of the object hierarchy.

Indirect selection is a useful secondary selection mechanism. It can be especially useful in 3D applications because:

- Users may have to choose among a large number of objects.
- An object may be obscured or too far away in the scene to be easily selected using direct selection.

Don't make indirect selection the only selection paradigm. Instead, use it to augment direct selection.

3D Selection Models

The four OSF/Motif selection models recommended for 2D applications are listed in Table 7-1. For 3D applications, users expect that one of the following two OSF/Motif selection models is supported:

- The *single selection model* in applications that allow users to select only one object at a time
- The *discontiguous selection model* in applications that allow users to select more than one object at a time (these objects don't have to be next to each other).

Table 14-1 describes the selection actions in the single selection and discontiguous selection models. It also lists which of these actions users minimally expect in a 3D application and which are optional but ideally supported.

Table 14-1 3D Selection Actions and Results

Action	Model	Required	Result
Click an object.	Single and discontiguous	Yes	Object is selected. Any previously selected objects are deselected.
<Ctrl>-click an object.	Discontiguous	Yes	Selection state of the object is toggled. That is, a previously selected object is removed from the current selection and a previously unselected object is added to the current selection.
<Shift>-click an object.	Discontiguous	No[a]	Same as <Ctrl>-clicking an object.
Click outside the selection in an empty area.[b]	Single and discontiguous	Yes	All objects are deselected.
Sweep out a selection area.[c]	Discontiguous	No	Only objects included in the sweep area are selected. All other objects are deselected.
<Ctrl>-sweep an area.[c]	Discontiguous	No	Selection state of each object inside the sweep area is toggled. Selection state of each object outside the sweep area is unchanged.
<Shift>-sweep an area.[c]	Discontiguous	No[d]	Same as <Ctrl>-sweep area.

a. Recommended if users are accustomed to using the <Shift> key in their other applications to toggle the selection state of an object.

b. An application may additionally include a "Deselect All" menu item. This is especially useful for applications that support densely populated scenes or objects that fill the entire background.

c. Can support sweep selection using either rectangle-drag or lasso-drag as described in "Direct Selection in 3D Applications."

d. Recommended if your application supports the selection action <Ctrl>-sweep area and users are accustomed to using the <Shift> key to toggle the selection state of an object.

Selection in Hierarchies of Objects

Some applications support hierarchies of objects and allow users to select more than one object in these hierarchies. If your application lets users select more than one object in a hierarchy of objects, provide at a minimum a method for selecting the highest and lowest object in the hierarchy and a method for adjusting the selection up and down the hierarchy.

3D Selection Design Guidelines

When designing the selection user interface for your 3D application...

- ❑ Follow the object-action paradigm of direct manipulation: The user first selects an object (or group of objects), then chooses an action to perform on it.
- ❑ For actions that apply to objects, apply the action to all the objects in the current selection and only to those objects.
- ❑ Provide one current selection for each application at any time. The current selection may be empty (that is, no selection). Note that each window of your application can maintain a separate selection, but there is only one current selection.
- ❑ Support direct selection as the primary selection mechanism. Using the left mouse button, the user either clicks directly on an object to select it or sweeps out an area to select multiple objects.
- ❑ Support indirect selection if your users need it. For example, allow users to select an indirect representation of an object such as an item in a list, or provide a “Select All” menu item.
- ❑ If your application lets users select more than one object in a hierarchy of objects, provide at a minimum a method for selecting the highest and lowest object in the hierarchy and a method for adjusting the selection up and down the hierarchy.

When deciding on a selection model...

❑ If your application allows users to select only one object at a time, support the OSF/Motif single selection model as follows:

- Users directly select an object by moving the pointer over it and pressing the left mouse button. Any previously selected object is deselected.
- Users deselect an object by clicking outside the selection in an empty area.

❑ If your application allows users to select more than one object at a time, support the OSF/Motif discontiguous selection model as follows. Ideally, support the entire OSF/Motif discontiguous selection model.

- Users directly select an object by moving the pointer over it and pressing the left mouse button. Any previously selected objects are deselected.
- Users <Ctrl>-click an unselected object to add it to the current selection, and <Ctrl>-click an already selected object to remove it from the current selection. That is, <Ctrl>-click toggles the selection state of the object.
- If users are accustomed to using the <Shift> key in other applications to toggle the selection state of an object, allow them to add and remove objects by <Shift>-clicking an object in addition to <Ctrl>-clicking an object. In that case, <Shift>-clicking an object performs the same selection actions as <Ctrl>-clicking an object.
- Users deselect all objects by clicking outside the selection in an empty area. In addition, a "Deselect All" menu item may be useful for some applications.
- Optionally, allow users to use sweep selection to select multiple objects, allowing either rectangle-drag or lasso-drag. At the end of a sweep action, the only objects selected are those inside the sweep area.
- Optionally allow users to <Ctrl>-sweep through a collection of objects to toggle the selection state of all objects inside the sweep area. That is, objects inside the sweep area that were previously selected are deselected, and objects inside the sweep area that were previously deselected are selected. The selection state of objects outside the sweep area doesn't change.
- If your application supports the optional <Ctrl>-sweep selection action and users are accustomed to using the <Shift> key in their other applications to toggle the selection state of an object, allow users to use <Shift>-sweep in addition to <Ctrl>-sweep to toggle the selection state of all objects inside the sweep area.

Selection Feedback for 3D Objects

When an object is selected, applications need to provide selection feedback on the object. This section discusses three techniques for indicating that an object is selected—bounding box, manipulator, and highlighting—and provides guidelines for when to use each technique.

This section covers these topics:

- "Bounding Box Selection Feedback"
- "Manipulator Selection Feedback"
- "Highlight Selection Feedback"

Bounding Box Selection Feedback

Figure 14-1 shows two objects with bounding box feedback: A box is placed around the object. The bounding box needs to be differently shaped or larger than the object itself so that it's readily visible. Using a distinct color for the box is also highly recommended.

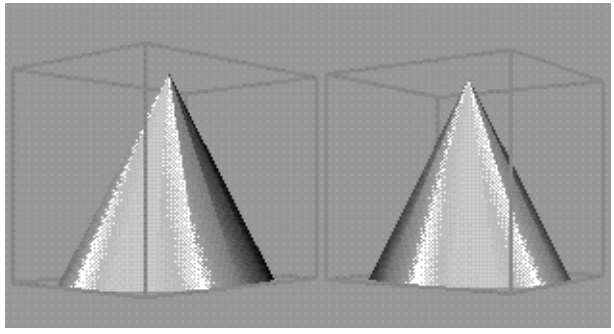


Figure 14-1 Two Objects with Bounding Box Feedback

Note that for some applications a bounding box may not be the most intuitive or useful shape. For example, some applications might use a bounding volume that's a pyramid or cone. Other applications might use a shape that uses planes that don't form a closed bounding volume. In most cases, however, users find the bounding box the most intuitive shape. The term bounding box is used in this document to mean a bounding box or a bounding volume.

Bounding box selection feedback is most appropriate for applications where users can select objects but either don't typically manipulate them (translate, rotate, scale) or can't manipulate them. This type of feedback is appropriate for both applications that support the single selection model and applications that support the discontiguous selection model (see "3D Selection Models").

Manipulator Selection Feedback

Figure 14-2 shows an application that uses a manipulator to indicate that an object is selected. A manipulator is a control that allows users to change the position, orientation, or scale of objects. Specific manipulators are discussed in Chapter 15, "Manipulating 3D Objects."

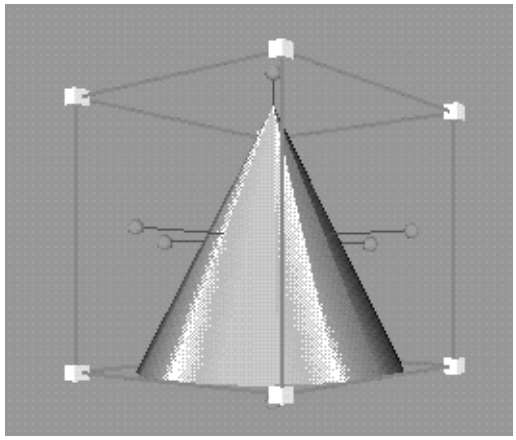


Figure 14-2 Object With Manipulator

Manipulator selection feedback is most appropriate for applications where users typically select objects to perform manipulations on them. For these applications, placing the manipulator on an object as soon as the object is selected has two advantages:

- The manipulator feedback clearly indicates that the object is selected.
- The immediate display of the manipulator makes the manipulation functions readily available.

If your application allows users to select more than one object at a time (see the description of the discontiguous selection model in "3D Selection Models"), you need to decide whether to use manipulators as selection feedback on all or just some of the

selected objects. Displaying a manipulator on all selected objects may make it more difficult for users to select and interact with a specific object's manipulator and may impede performance.

As an alternative, some applications support the concept of a lead object (see "Lead Objects in 3D Applications"). The lead object has the manipulator as the selection feedback and other selected objects have bounding boxes as the selection feedback (see "Lead Object When Selecting Multiple Objects").

Highlight Selection Feedback

In highlight selection feedback, selected objects are highlighted in some way to make them stand out, for example, by brightening the colors of the selected objects or using the same distinct color for all selected objects.

This technique is most appropriate for experiential consumer applications that allow users, for example, to explore an architectural model or a museum. These types of applications typically want to always present a realistic representation of objects in the scene to make the experience immediate. Cluttering the scene with bounding boxes would greatly reduce the realism.

3D Selection Feedback Design Guidelines

When designing selection feedback for your 3D application...

- ❑ Provide clear feedback on each object as it is selected.
- ❑ When using a bounding box for selection feedback, make sure that it's differently shaped or larger than the object itself so that it's readily visible. Using a distinct color for the bounding box is also highly recommended.
- ❑ If users don't typically select objects in your application to manipulate them (translate, rotate, scale) or can't manipulate the selected objects, use bounding boxes to indicate the selected objects.
- ❑ If users typically select objects in your application to manipulate them, use the manipulator as selection feedback. If your application allows more than one object to be selected at a time, consider displaying the manipulator only on a lead object and bounding boxes on the other selected objects.
- ❑ If your application needs to always present a realistic "experience-oriented" representation of objects in the scene, highlight selected objects in some way rather than cluttering the scene with bounding boxes or manipulators.

Lead Objects in 3D Applications

If your application allows users to select more than one object at a time, consider identifying one of those objects as the lead object (also sometimes called the master object). The lead object is necessary for performing certain object manipulations if more than one object is selected. For example, if you have several objects selected and rotate the lead object, all selected objects rotate around the lead object's center of rotation.

The lead object is clearly distinguished from other objects in the selection. For example as shown in Figure 14-3, if users typically select objects to manipulate them, the manipulator can be used to distinguish the lead object and bounding boxes can be used as selection feedback on other selected objects (see "Manipulator Selection Feedback").

The rest of this section discusses how to choose the lead object in specific situations.

- "Lead Object When Selecting Multiple Objects"
- "Lead Object During Grouping and Ungrouping"

For more detailed information on the role of the lead object during specific manipulations, see "Object Manipulation for Multiple Selected 3D Objects" in Chapter 15.

Lead Object When Selecting Multiple Objects

If the user clicks to define a selection or <Ctrl>-clicks to add to a selection (see "3D Selection Models"), the object just added to the selection becomes the lead object. Having the last object selected be the lead object differs from the standard OSF/Motif guidelines of having the lead or "anchor" be the first object selected regardless of how many objects are later added to the selection.

This difference between the 2D-oriented OSF/Motif guidelines and these 3D guidelines is necessary because it allows users to quickly change the lead object in a 3D application. For example, to change the lead object to one that's currently not selected, a user can use click or <Ctrl>-click to select that object. To change the lead object to one that's already selected, the user can click or <Ctrl>-click twice on the object: the first time to deselect it and the second time to make it the most recently selected object.

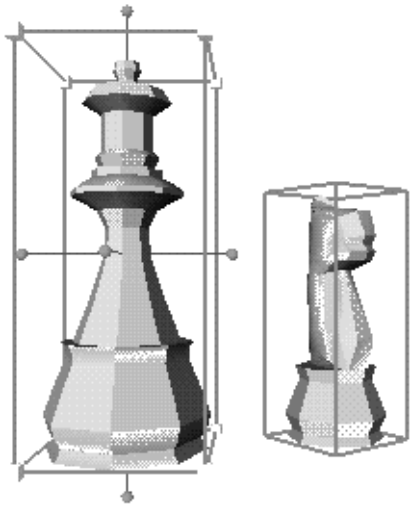


Figure 14-3 Selection Feedback: Lead Object Has Manipulator.

In some situations, users may add multiple objects to the current selection without actually clicking on the individual objects. For example, a user may select several objects using sweep selection (see "3D Selection Models") or choose a "Select All" menu command to select all objects in a scene. Alternatively, a user may import a grouped object, place it in the scene, then ungroup the collection, leading to the selection of all objects that were previously grouped (see "Lead Object During Grouping and Ungrouping"). In these situations, make the lead object the one that's closest to the camera and closest to the middle of the viewing window.

When the user deselects the lead object, move back through the previous lead objects making the most recent lead object that's still selected the new lead object.

Lead Object During Grouping and Ungrouping

If the user groups a collection of objects, the group becomes a single selected object. Since the group is also considered to be the last selected object, it becomes the lead object. All selection feedback is removed from the individual objects in the group and the selection feedback is displayed only for the group.

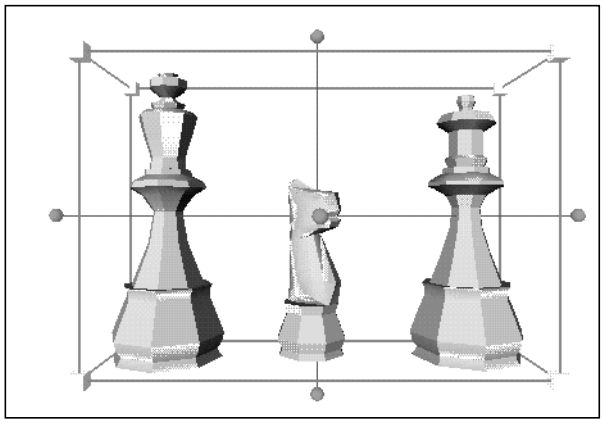


Figure 14-4 Grouped Collection of Objects With Manipulator.

When the user ungroups a grouped collection of objects, for example, by selecting the group and choosing an Ungroup command, each object in that group becomes selected. The object from the group that's closest to the camera and closest to the middle of the viewing window becomes the lead object. Figure 14-4 shows a grouped collection of objects and Figure 14-5 shows the same group ungrouped. After ungrouping, the knight is the lead object because it's closest to the camera and closest to the middle of the viewing window. In this example, the lead object uses the manipulator to distinguish it from other selected objects.

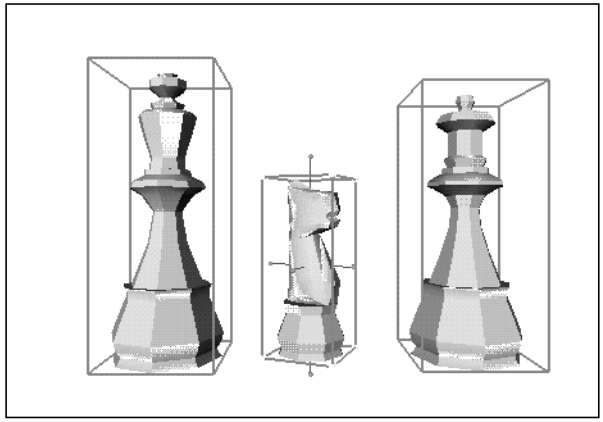


Figure 14-5 Ungrouped Collection of Objects (Knight Is Lead Object)

Lead Object Design Guidelines for 3D Applications

When designing the selection user interface for your 3D application...

- ❑ If your application allows users to select more than one object at a time, consider identifying one of those objects as the lead object.
- ❑ Clearly distinguish the lead object from other selected objects:
  - If users typically select objects to manipulate them, consider making this distinction by displaying the manipulator only on the lead object and a bounding box on all other selected objects.
  - Otherwise, if there are manipulators or bounding boxes on all selected objects, distinguish the lead object another way (color, full manipulator instead of partial, and so on.).
- ❑ If the user clicks to define a selection or <Ctrl>-clicks to add to a selection, make the lead object the last object selected. This allows users to change the lead object using click or <Ctrl>-click on an object that's currently not selected, or using <Ctrl>-click twice on a currently selected object.
- ❑ If the user employs a single action such as a sweep selection to select multiple objects at the same time, make the lead object the one that's closest to the camera and closest to the middle of the viewing window.
- ❑ When the user deselects the lead object, move back through the previous lead objects making the most recent lead object that's still selected the new lead object.
- ❑ If the user groups a collection of objects, make the group the new lead object.
- ❑ When the user ungroups a grouped collection of objects, each object that was in the group becomes selected, and the object from the group that's closest to the camera and closest to the middle of the viewing window becomes the new lead object.

Manipulating 3D Objects

Users of 3D applications typically need to change the position, orientation, and scale of objects. Users of 3D editing applications are especially in need of fine control and power in manipulating objects. Applications such as games that don't allow editing may need less control.

This chapter provides guidelines for three basic manipulation techniques: translation, rotation, and scaling, and their variations. It also discusses the special case of moving the center of rotation and scaling, which applications can optionally add to the basic techniques. It includes these topics:

- "Basic 3D Manipulation Techniques"
- "Rotating 3D Objects"
- "Scaling 3D Objects"
- "Changing the Center of Rotation and Scaling for 3D Objects"
- "Object Manipulation for Multiple Selected 3D Objects"

Note that the focus of this chapter is manipulating complete objects; it doesn't address manipulating vertices, faces, or edges.

Basic 3D Manipulation Techniques

This section introduces the general paradigm for 3D object manipulation. It first describes the different phases of each manipulation process (for example, approach phase and drag phase) and then lists the decisions you have to make to provide a consistent user interface. It discusses these topics:

- "Phases of 3D Manipulation"
- "Free and Constrained 3D Manipulation"
- "Basic 3D Manipulation Guidelines"
- "Manipulator Presentation and Selection"

Note that because users are accustomed to manipulating objects directly, direct manipulation of 3D objects is usually preferable. In some cases, manipulating objects with the help of an intermediary like a separate dialog box can be useful, either as a secondary method or to make more precise manipulation possible. An example is a dialog box that lets users type in percentages for scaling or precise degree values for rotation.

Phases of 3D Manipulation

The general paradigm for 3D object manipulation consists of several phases: manipulator display, approach, grab, drag, and release. In each of these phases, the manipulator uses certain feedback styles—neutral, locate highlight, selected, choice, and guide—to indicate the actions currently available to the user. These feedback styles are discussed in detail in "3D Manipulation Feedback".

The phases of 3D manipulation are used throughout the rest of this chapter to illustrate how users perform the different manipulation operations. The phases can be summarized as follows:

1. **Manipulator Display Phase.** The user performs an action and the manipulator is displayed in its neutral state on the selected object. As discussed in "Selection Feedback for 3D Objects" in Chapter 14, manipulators usually appear when users select an object. Some applications may, however, require an additional step to have the manipulators appear. Silicon Graphics provides standard manipulators for translation, rotation, and scaling.

2. **Approach phase**. As the user moves the pointer over a control on the manipulator, the control changes to locate highlight style (for example, by changing color) to indicate that it's an active control that can be selected.
3. **Grab phase**. The user selects the control on the manipulator by positioning the pointer over it and pressing the left mouse button.
    - If the selection can result in only one action, the application provides selected feedback and, if necessary, guide feedback. For example, if the user selects a scaling handle (see Figure 15-9), the handle is displayed in the selected style and lines in the guide style appear to indicate the direction of dragging for the drag phase.
    - If more than one action is possible, the application provides selected feedback, choice feedback and, if necessary, guide feedback. For example, if the user selects a rotation handle (see Figure 15-6), the handle is displayed in the selected style, and two arrows in choice style are displayed to indicate that the user has to clarify the direction of the rotation. In addition, two rings appear in the guide style to indicate the direction of dragging for the drag phase.
4. **Drag phase**. The user drags the selected control to manipulate the object. During the drag, the object changes in response to the user's actions. If choice feedback was provided in the grab phase, the user has to first resolve the choice by dragging the pointer in one of the choice directions. Once the choice is made, the application replaces the choice style with the selected style to indicate the choice was made.
5. **Release phase**. The user releases the mouse button. The manipulator is returned to its original displayed state.

For translation, rotation, and scaling, use the standard manipulators provided by Silicon Graphics. If you design your own manipulator and controls for actions other than translation, rotation, and scaling, be sure to include in your design:

- nonambiguous controls
- precise control
- nonambiguous gestures

3D Manipulation Feedback

Provide immediate feedback about available functionality of the manipulator at all times. Applications need to use consistent design schemes across manipulators and controls to provide this feedback and indicate the current state of the manipulator. Experience has shown that the following feedback styles, which use a design scheme based on colors, are useful:

- **Neutral**. Indicates that the user isn't interacting with the controls on the manipulator. Manipulators in neutral style (often green or white) stand out from the other elements in the scene.
- **Locate highlight**. Indicates that the pointer is over a control on the manipulator that can be selected. The control brightens (often to an orange color) as the pointer passes over it to indicate that it's a live, functional control.
- **Selected feedback**. Indicates that the control has been selected. Controls in selected style (often rendered in yellow) stand out strongly.
- **Choice feedback**. Indicates the user has selected a control but has to make an additional selection. For example, if the user wants to constrain translation to just one axis of a translation plane, the user first selects the plane for translation, then two arrows prompt the user to choose which of the two axes to translate along. Feedback in choice style (often rendered in orange) stands out but not as strongly as the selected style.
- **Guide feedback**. Provides supplementary information to guide the user during object manipulation. For example, when the user chooses to rotate an object, two (or three) rings indicating the rotation sphere appear to guide the user. Feedback in the guide style (often rendered in purple) recedes but is distinct from the object and the manipulator.

In addition to the immediate feedback, continuous feedback on the state of the object is important while the user is manipulating objects. For example, as the user translates an object, the object moves in the scene in a smooth and continuous fashion so the user always knows the location of the object and can decide exactly where to position it. If the user manipulates objects in a complex scene, it may be difficult to maintain an acceptable frame rate during the manipulation, so adaptive rendering may be necessary. (See "3D Viewing Performance and Scene Fidelity" in Chapter 13 for recommended frame rates and information on adaptive rendering.)

Free and Constrained 3D Manipulation

Depending on how complex a given operation is and how often it's used, the default manipulation is either free (in all three dimensions) or constrained. Users always perform the default operations by directly interacting with the manipulator. Users perform nondefault operations by holding down a modifier key while interacting with the manipulator. Choice of a modifier key follows the guidelines outlined in Table 12-1, "Use of Modifier Keys in a 3D Application," on page 235.

Table 15-1 provides an overview of the available manipulations and their associated modifier keys.

Table 15-1 Overview of Manipulation Techniques

Manipulation	Default?	Modifier Key
Simple (Planar) 3D Translation	Yes	none
Translation Constrained to One Axis of the Plane		<Shift>
Translation Constrained to the Normal of the Selected Plane		<Ctrl>
Constrained 3D Rotation (rotation around one axis)	Yes	none
Free 3D Rotation (rotation around a point, by default center of object)		<Shift>
Uniform 3D Scaling	Yes	none
Uniform Scaling Around a Corner		<Ctrl>
Axial 3D Scaling (Stretching)		<Shift>
Axial Scaling Around a Side		<Shift>-<Ctrl>
Changing the Center of Rotation and Scaling Along a Plane		<Ctrl>
Changing the Center of Rotation and Scaling Along an Axis		<Shift>-<Ctrl>

Manipulator Presentation and Selection

Your application needs to provide default manipulators that are appropriate for the functional requirements of the users and for the tasks they most commonly perform. For example, if users commonly move objects through a scene but rarely scale or rotate them, only a translation manipulator is expected. If users commonly perform all three basic manipulation techniques, display translation, rotation, and scaling manipulators by default.

Once you've defined default manipulators, consider allowing users to change the set of manipulators that's displayed. At a minimum, allow users to make this change by choosing commands in the View menu. For example, if rotation is not a default manipulation but rarely used, allow users to add the rotation manipulator when they need it.

Basic 3D Manipulation Guidelines

When designing a user interface for object manipulation in a 3D application...

- ❑ Let users manipulate objects directly whenever possible.
- ❑ If an intermediary, such as a dialog box, yields more precise results for an action and your users need precise manipulations, provide an intermediary method in addition to direct manipulation.
- ❑ Provide manipulators to allow users to use direct manipulation when editing objects.
- ❑ Use the standard manipulators provided by Silicon Graphics for translation, rotation, and scaling.
- ❑ Make sure users can readily identify a manipulator's controls and how to interact with those controls. Also, make sure the controls allow users to precisely manipulate an object.
- ❑ Provide immediate feedback on available actions during the different stages of manipulation as follows:

- Display phase—Display the manipulator in its neutral state.
- Approach phase—As the pointer passes over a control on the manipulator, locate highlight the control to show that it's a live, functional control.
- Grab phase—Provide selected feedback if there is no additional choice to make; otherwise provide choice feedback. Provide guide feedback as appropriate to facilitate user interaction.
- Drag phase—If the user was presented with a choice in the grab phase, resolve it when the user begins the drag, and replace the choice feedback with selected feedback.
- Release phase—Return the manipulator to its neutral state.

❑ Use clearly distinguishable feedback styles as follows:

- Provide neutral feedback using a style that stands out from the scene; for example, use the color green or white.
- Provide locate highlight for manipulator controls. That is, the controls brighten (often to an orange color) as the pointer passes over them.
- Provide selected feedback using a style that stands out strongly; for example, use the color yellow.
- Provide choice feedback using a style that stands out, but not as strongly as selection feedback; for example, use the color orange.
- Provide guide feedback using a style that recedes but is distinct from the other styles; for example, use the color purple.

❑ Provide continuous feedback on the status of an object while the user is manipulating it. For example, as the user translates an object, the object should move in the scene in a smooth and continuous fashion to keep the user updated on the location of the object. Use adaptive rendering if necessary.

❑ Make commonly used and critical manipulation techniques immediately available on the manipulator via the left mouse button. Make less commonly used techniques available on a secondary level, for example, through modifier keys used in conjunction with the left mouse button.

When displaying manipulators in your 3D application...

❑ Decide which manipulators to display by default based on the functional requirements of your users and their most common tasks.

❑ Consider allowing users to change which subset of manipulators are currently displayed. At a minimum, allow users to specify this subset using entries in the View menu.

Translating 3D Objects

Translating an object means moving it and positioning it at a desired location within the scene. This section discusses and provides guidelines for the basic translation behavior users expect in 3D applications. It includes the following topics:

- “3D Translation Basics”
- “Simple (Planar) 3D Translation”
- “Constrained 3D Translation”

3D Translation Basics

In most cases, the intuitive manipulator for translation is a bounding volume (see Figure 15-1), usually a bounding box. The bounding box defines a local coordinate space for the object and lets the user perform translation in the object's coordinate space, which may or may not be aligned with world (scene) coordinate space. See "Bounding Box Selection Feedback" for more detail on bounding boxes.

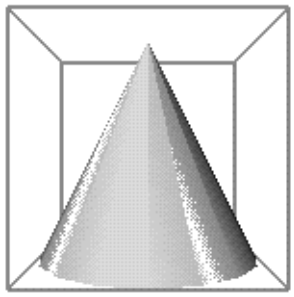


Figure 15-1 Object With Translation Manipulator (Bounding Box)

Provide access to both simple (planar) and constrained (axial) translation:

- Planar translation moves objects along a 2D plane. It's discussed in detail in "Simple (Planar) 3D Translation."
- Constrained translation moves objects either:
  - along only one of the axes that define the plane of translation or
  - along the normal to these axes

  Constrained translation techniques are discussed in detail in "Constrained 3D Translation."

Note that translation in all three dimensions isn't a recommended technique and therefore isn't discussed in this guide. This is because even for experienced users, 3D translation is difficult to perform.

In the process of translating, the user could push an object so far away from the camera (along the z-axis) that it's no longer visible. While users may intend to position the object as far away as possible, they almost never want it to disappear from view. Your application should check whether users move objects beyond the vanishing point and either warn them or prevent them from doing so.

Simple (Planar) 3D Translation

Planar translation moves objects along a 2D plane. It's the most common type of translation and is therefore the default translation behavior.

Table 15-2 illustrates user input and the resulting application behavior for planar translation:

Table 15-2 Phases of Planar Translation

Phase[a]	User Action	Application Response[b]
Approach	User moves pointer over plane of interest.	Plane locate highlights when pointer is within its boundaries.
Grab	User presses and holds left mouse button while pointer is over the plane of interest.	User interface controls on the object, including other manipulators, are removed except for the bounding box (translation planes). Selected plane is displayed in selected style. Other planes of the bounding box are displayed in guide style. Standard translation feedback is displayed at the location of the pointer. Feedback consists of a perpendicular set of arrows in the selected style; the arrows represent the two main axes defining the translation plane (see Figure 15-2). Pointer shape remains the upper left pointing arrow.
Drag	User drags plane of interest to translate object.	Object, bounding box (translation planes), and translation feedback (arrows) move along plane of interest based on user input (see Figure 15-2). All user input is interpreted to follow the plane of interest. As the object is translated, it's continuously displayed in its current state, using adaptive rendering if necessary (see "3D Viewing Performance and Scene Fidelity" in Chapter 13).
Release	User releases mouse button to stop translation motion.	Translation feedback (arrows) disappears. The user interface controls on the object that were removed in the grab phase are redisplayed in their original state.

a. See "Phases of 3D Manipulation."

b. See "3D Manipulation Feedback."

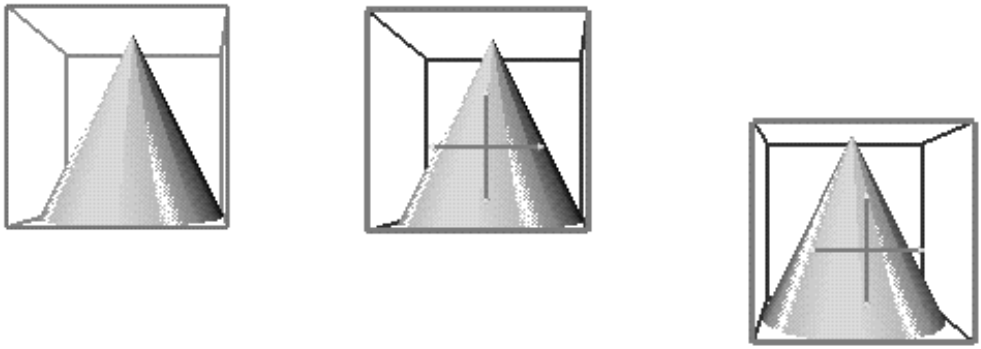


Figure 15-2 Simple Planar Translation Sequence

Dragging any side of the bounding box translates the object along that plane. Figure 15-2 shows translating an object to the right and down. The user has dragged the facing plane of the bounding box, enabling translation in the xy plane. Therefore, the object can move left and right as well as up and down. The object moves in the coordinate space defined by the bounding box and is independent of the scene coordinate space.

Make sure it's not possible to select through the translation planes. For example, in Figure 15-2 the left side of the bounding box, which represents the xz plane, is behind the front panel, which represents the xy plane. As the user moves the pointer over the front panel, this front xy plane locate highlights indicating that the user can select it. As the user moves the pointer over both the front xy plane and the side xz plane behind it, only the front xy plane locate highlights because the side xz panel cannot be selected through the front xy plane.

Constrained 3D Translation

Constrained translation gives users finer control over the position of an object. The two recommended types, discussed in this section, are:

- "Translation Constrained to One Axis of the Plane"
- "Translation Constrained to the Normal of the Selected Plane"

The user indicates one of these nondefault translations by holding down a modifier key while performing the standard translation action of dragging one of the translation planes in the manipulator (see "Free and Constrained 3D Manipulation"). Users expect to be able to switch from unconstrained translation to constrained translation and back at any point in the translation by pressing the appropriate modifer key, and expect to continue performing constrained translation until the modifier key is released.

Translation Constrained to One Axis of the Plane

Performing translation constrained to only one of the axes that define a translation plane is similar to planar translation (see Table 15-2). However, as shown in Table 15-3, the grab phase changes to allow selection of a single axis, and the drag phase has a single axis as the feedback. The user presses the <Shift> modifier key to indicate constrained translation along only one axis of the selected plane.

Table 15-3 Phases of Translation Along One Axis of the Selected Plane

Phase	User Action	Application Response
Approach	Same as for planar translation (see Table 15-2).	Same as for planar translation.
Grab	User presses and holds the <Shift> key and left mouse button while pointer is over plane of interest.	Same as for planar translation with this exception: Translation feedback is displayed in the choice style (see Figure 15-3).
Drag	User continues to press the <Shift> key and drags in the direction of one of the two choice arrows to identify the desired translation axis.	When the user has moved the pointer far enough to clearly indicate the axis for translation, the arrow representing that axis changes to selected style and the other arrow disappears (see Figure 15-3). Remainder of this phase is the same as for planar translation with this exception: translation is along the selected axis, not along the selected plane, so all user input is interpreted to follow the selected axis.
Release	Same as for planar translation.	Same as for planar translation.

Figure 15-3 illustrates this type of constrained translation. The user simultaneously presses the <Shift> key and the left mouse button. In response, the application displays the translation feedback arrows in choice style. The user then selects one of the two axes indicated by these choice arrows. After the user has dragged slightly to the right, only the arrow representing the user's choice of the left-right axis (x) remains, and translation is restricted to this axis until the user releases the <Shift> key.

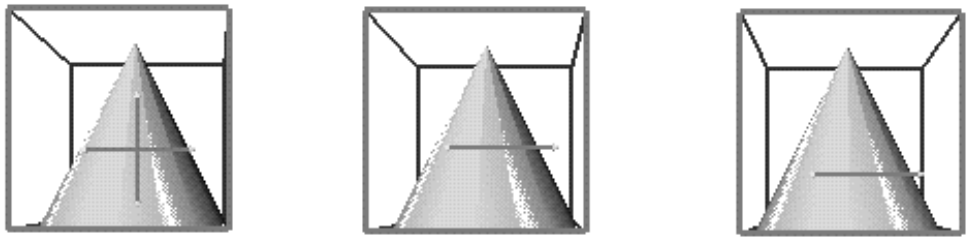


Figure 15-3 Constrained Translation Along One Axis of the Selected Plane

Translation Constrained to the Normal of the Selected Plane

Users want to perform translation along a path perpendicular to a plane primarily for these two reasons:

- Other planes of the bounding box that are orthogonal to the current plane aren't always visible or accessible.
- Some users need to rapidly translate objects along all three axes.

Performing translation constrained to the axis that's normal to the selected translation plane is similar to planar translation (see Table 15-2). However, as shown in Table 15-4, the grab phase changes to show the axis that's normal to the selected plane, and the drag phase translates the object along this normal axis. The user presses the <Ctrl> modifier key to indicate translation constrained along the normal.

Table 15-4 Phases of Translation Along the Normal to the Selected Plane

Phase	User Action	Application Response
Approach	Same as for planar translation. (see Table 15-2).	Same as for planar translation.
Grab	User presses and holds <Ctrl> key and left mouse button while pointer is over plane of interest.	Same as for planar translation with this exception: Translation feedback consists of an arrow representing the normal to the selected plane and a small pair of axes representing the main axes of the selected translation plane (see Figure 15-4). This feedback is displayed in selected style.
Drag	User continues to press <Ctrl> key and drags plane of interest to translate object.	Same as for planar translation except that object, bounding box (translation planes), and translation feedback (axes and arrow) move along the normal to plane of interest (see Figure 15-4), so all user input is interpreted to follow this normal.
Release	Same as for planar translation.	Same as for planar translation.

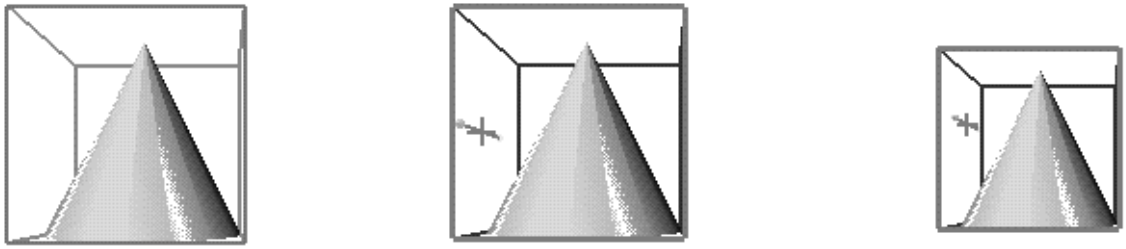


Figure 15-4 Constrained Translation Along the Normal to the Selected Plane

Figure 15-4 shows this type of constrained translation. The user simultaneously presses the <Ctrl> key and the left mouse button. In response, the application displays the translation feedback arrow and axes in the selected style. As the user drags the frontmost plane of the bounding box in this example (the xy plane), translation is restricted to the axis normal to this selected plane (z) until the user releases the <Ctrl> key.

3D Translation User Interface Guidelines

When designing a user interface for 3D translation...

- ❑ Use the standard translation manipulator either alone or in combination with the other standard manipulators. This manipulator is a set of translation planes arranged to define a bounding box around the object.
- ❑ Allow users to perform translation in the planes of the object's bounding box. These planes may or may not be aligned with the world (scene) coordinate space.
- ❑ Don't allow users to translate objects that are part of a scene into unusable locations, such as behind the camera or so far back along the z axis that they vanish from view.
- ❑ Provide planar translation as the default translation method (see Table 15-2 on page 289).
- ❑ Provide access to constrained translation along only one axis of a plane (see Table 15-3 on page 291).
- ❑ Provide access to axial (constrained) translation along the normal to a plane (see Table 15-4 on page 293).
- ❑ Allow users to switch from planar translation to constrained translation and back at any point in a translation. For example, if the user is performing planar translation by dragging a translation plane and then presses the <Shift> key, switch to axial translation along the plane until the <Shift> key is released.
- ❑ Display the appropriate translation feedback as the user switches between unconstrained and constrained translation.
- ❑ Don't allow users to select objects or controls through the translation planes.

Rotating 3D Objects

Rotating an object means turning it around an axis or around a point. Rotating objects is important in most 3D applications because users need access to all sides of the object. This section discusses and provides guidelines for the basic rotation behavior users expect in 3D applications. It discusses these topics:

- "3D Rotation Basics"
- "Constrained 3D Rotation"
- "Free 3D Rotation"

Some applications may also find it useful to allow users to change the center of rotation. This is discussed in "Changing the Center of Rotation and Scaling for 3D Objects".

3D Rotation Basics

Provide access to both constrained rotation and free rotation in your application; both are critical in providing users access to rotation functionality:

- Constrained rotation is rotation around an axis. Because free rotation makes the object difficult to control, constrained rotation is the appropriate default behavior. It's discussed in "Constrained 3D Rotation."
- Free rotation is rotation around a point. It's discussed in "Free 3D Rotation."

When users rotate an object, the default center of rotation for free rotation is the center of the object itself (the center of the object's bounding box); for constrained rotation, it's an axis that runs through the center of the object. There are situations where users need to change the center of rotation; see "Changing the Center of Rotation and Scaling for 3D Objects" for a detailed discussion of this special case.

The standard rotation manipulator, shown in Figure 15-5, is a set of rotation handles, at most one per side, that emanate from the object's center of rotation. The length of each handle is the length of the bounding box plus one-eighth of it on each side.

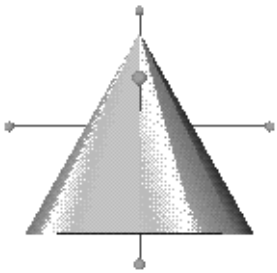


Figure 15-5 Object With Rotation Manipulator (Rotation Handles)

Constrained 3D Rotation

Constrained rotation is rotation around an axis. Because it's much easier to control than free rotation, it's the default rotation behavior. Table 15-5 illustrates user input and the resulting application behavior for constrained rotation.

Table 15-5 Phases of Constrained Rotation

Phase[a]	User Action	Application Response[b]
Approach	User moves pointer over a rotation handle.	Rotation handle of interest locate highlights.
Grab	User presses and holds left mouse button while pointer is over rotation handle of interest.	User interface controls on the object, including other manipulators, are removed except for rotation handles. Rotation handle of interest is displayed in selected style, and other handles are displayed in neutral style. Standard rotation feedback is displayed at the location of the selected handle. Feedback consists of a perpendicular set of arrows in the choice style and two rings in guide style to indicate available rotation directions (see Figure 15-6). Pointer shape remains the upper left pointing arrow.
Drag	User drags rotation handle to rotate object around the selected axis.	When user has moved the pointer far enough to clearly indicate direction and axis for rotation, the corresponding arrow changes to selected style, and the other arrow and its associated guide ring disappear (see Figure 15-6). Object, rotation handles, and rotation feedback (ring and arrow) rotate around the selected axis based on user input. All user input is interpreted to follow the ring. As the object is rotated, it's continuously displayed in its current state, using adaptive rendering if necessary (see "3D Viewing Performance and Scene Fidelity" in Chapter 13).
Release	User releases mouse button to stop rotation motion.	Rotation feedback (arrow and ring) disappears. User interface controls on object that were removed in grab phase are redisplayed in their original state.

a. See "Phases of 3D Manipulation."

b. See "3D Manipulation Feedback."

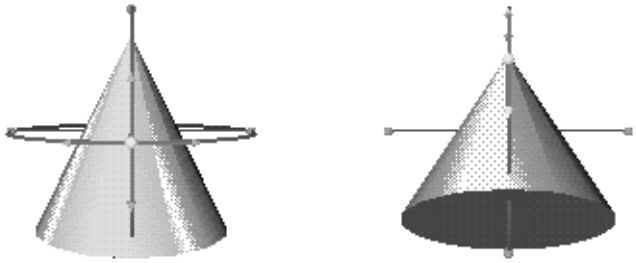


Figure 15-6 Constrained Rotation Sequence

Figure 15-6 shows the sequence of events for constrained rotation. When the user grabs a rotation handle, the rotation feedback appears indicating that the user must choose one of two axes for the rotation. The user chooses a direction and an axis for rotation by dragging along one of the two arrows. Once the axis has been chosen, all user input is interpreted to rotate the object around the selected axis in the given direction.

Free 3D Rotation

In free rotation, the object rotates around a point rather than an axis; that is, the object can rotate around all three axes simultaneously. Typically, the default center of rotation is at the center of the object's bounding box. The user presses the <Shift> modifier key to indicate free rotation.

Free rotation is similar to constrained rotation (see Table 15-5). However, as shown in Table 15-6, the grab phase displays a virtual trackball as the rotation feedback, and the drag phase rotates the object around the center of rotation.

Table 15-6 Phases of Free Rotation

Phase	User Action	Application Response
Approach	Same as for constrained rotation (see Table 15-5).	Same as for constrained rotation.
Grab	User presses and holds <Shift> key and left mouse button while pointer is over rotation handle of interest.	Same as for constrained rotation with this exception: Rotation feedback consists of three rings representing a virtual trackball (see Figure 15-7). This feedback is displayed in guide style. No choice arrows are provided.
Drag	User continues to press <Shift> key and drags rotation handle to rotate object simultaneously in all three dimensions around the center of rotation.	Same as for constrained rotation except that object, rotation handles, and rotation feedback (three rings) rotate around the center of rotation (see Figure 15-7). All user input is interpreted to follow the virtual trackball.
Release	Same as for constrained rotation.	Same as for constrained rotation.

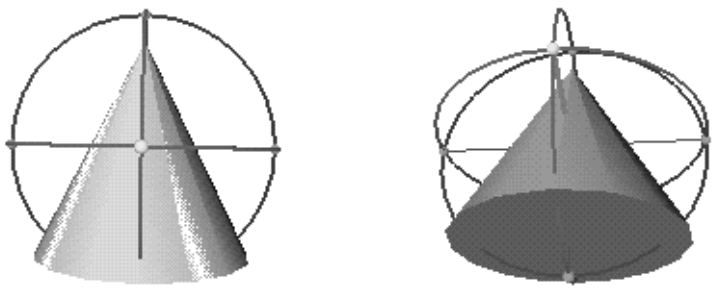


Figure 15-7 Free Rotation Sequence

Figure 15-7 illustrates free rotation. The user simultaneously presses the <Shift> key and the left mouse button. In response, the application displays the virtual trackball rotation feedback. As the user drags the pointer, the object is rotated around the point at the center of the virtual trackball.

Users expect to be able to switch between free rotation and constrained rotation at any time during a rotation operation:

- If the user is performing constrained rotation, pressing the <Shift> key immediately puts the object into free rotation. The pointer starts following a virtual trackball path.
- If the user is performing free rotation, releasing the <Shift> key immediately puts the object into constrained rotation. The object rotates around the axis the pointer is moving around.
- Users can continually press and release the <Shift> key to switch to and from free rotation as long as they continue to press the left mouse button while the pointer is over a rotation handle.

3D Rotation User Interface Guidelines

When designing a user interface for 3D rotation...

- ❑ Use the standard rotation manipulator either alone or in combination with the other standard manipulators. This manipulator is a set of handles that emanate from the center of rotation. Typically, the default center of rotation is the center of the object (the center of the object's bounding box).
- ❑ Provide constrained rotation around an axis as the default rotation method (see Table 15-5 on page 296).
- ❑ Provide access to free rotation around a point (see Table 15-6 on page 298).
- ❑ Allow users to switch between constrained and free rotation at any point in a rotation. For example, if the user is performing constrained rotation by dragging along a ring and then presses the <Shift> key, switch to free rotation and interpret pointer movements as following the virtual trackball until the <Shift> key is released.
- ❑ As the user switches between constrained and free rotation, display the appropriate rotation feedback. For example, if the user is performing free rotation and then releases the <Shift> key, switch to constrained rotation and display the appropriate rotation feedback. Determine the direction and axis for rotation based on the next pointer movement. Once the direction and axis have been determined, display the appropriate arrow and ring feedback for this direction and remove the virtual trackball.

Scaling 3D Objects

Scaling an object means enlarging or reducing it without changing its position or orientation. This section discusses and provides guidelines for the basic scaling behavior users expect in 3D applications. It includes these topics:

- "3D Scaling Basics"
- "Uniform 3D Scaling"
- "Axial 3D Scaling (Stretching)"
- "Scaling Around the Opposite Corner or Side"

Some applications may find it useful to allow users to change the center of scaling. This is discussed in "Changing the Center of Rotation and Scaling for 3D Objects."

3D Scaling Basics

There are two scaling operations, uniform scaling and axial scaling:

- Uniform scaling preserves proportions, and is the most common type of scaling so it's the default scaling behavior. Uniform scaling is discussed in detail in "Uniform 3D Scaling".
- In axial scaling (also referred to as stretching), the object is scaled or stretched in only one of the three dimensions at a time. Because axial scaling is less commonly used than uniform scaling, it's the non-default scaling operation. Axial scaling is discussed in detail in "Axial 3D Scaling (Stretching)."

Objects are scaled with respect to a fixed anchor point, which is, by default, the center of the object (the center of the object's bounding box). Users also need to occasionally scale an object around a corner or side of the object's bounding box, as discussed in "Scaling Around the Opposite Corner or Side." Some applications may find it necessary to allow users to change the center of scaling. This is discussed in "Changing the Center of Rotation and Scaling for 3D Objects."

The standard scaling manipulator, shown in Figure 15-8, is a set of cube-shaped handles located at the vertices of the object's bounding box. To scale an object, the user drags one of these cube-shaped handles. Dragging toward the center of scaling shrinks the object, dragging away from the center of scaling enlarges the object.

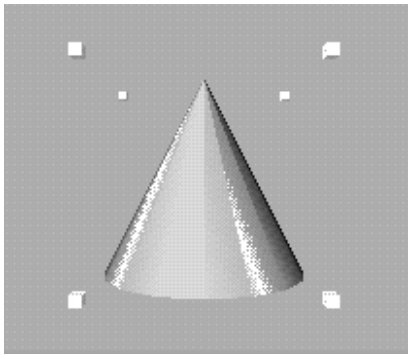


Figure 15-8 Object With Scaling Manipulator (Scaling Handles)

Uniform 3D Scaling

Uniform scaling preserves proportions; that is, all axes of the object are uniformly enlarged or reduced during the operation. It's the most common type of scaling so it's the default scaling behavior. Typically, the default center of scaling is at the center of the object (the center of the object's bounding box). Table 15-7 illustrates user input and the resulting application behavior for uniform scaling.

Table 15-7 Phases of Uniform Scaling

Phase[a]	User Action	Application Response[b]
Approach	User moves pointer over a scaling handle.	Scaling handle of interest locate highlights.
Grab	User presses and holds left mouse button while pointer is over scaling handle of interest.	User interface controls on the object, including other manipulators, are removed except for the scaling handles. Scaling handle of interest is displayed in selected style. Other handles are displayed in neutral style. Standard scaling feedback is displayed: a set of arrows in guide style that emanate from the center of scaling and go through each of the scaling handles and rotation handles displayed in neutral style (see Figure 15-9). The rotation handles provide feedback on the current location of the center of rotation and scaling. Pointer shape remains the upper left pointing arrow.
Drag	User drags scaling handle to scale object in all three dimensions around center of scaling.	Object, scaling handles, and scaling feedback are uniformly scaled in all three dimensions around the center of scaling based on user input (see Figure 15-9). All user input is interpreted to move either closer to or farther away from the center of scaling. As the user drags closer to the center of scaling, object, scaling handles, and scaling feedback shrink. As the user drags farther away from the center of scaling, the object, and so on, are enlarged. As the object is scaled it's continuously displayed in its current state, using adaptive rendering if necessary (see "3D Viewing Performance and Scene Fidelity").
Release	User releases mouse button to stop scaling.	Scaling feedback consisting of guide arrows and rotation handles disappears. User interface controls on the object that were removed in the grab phase are redisplayed in their original state.

a. See "Phases of 3D Manipulation."

b. See "3D Manipulation Feedback."

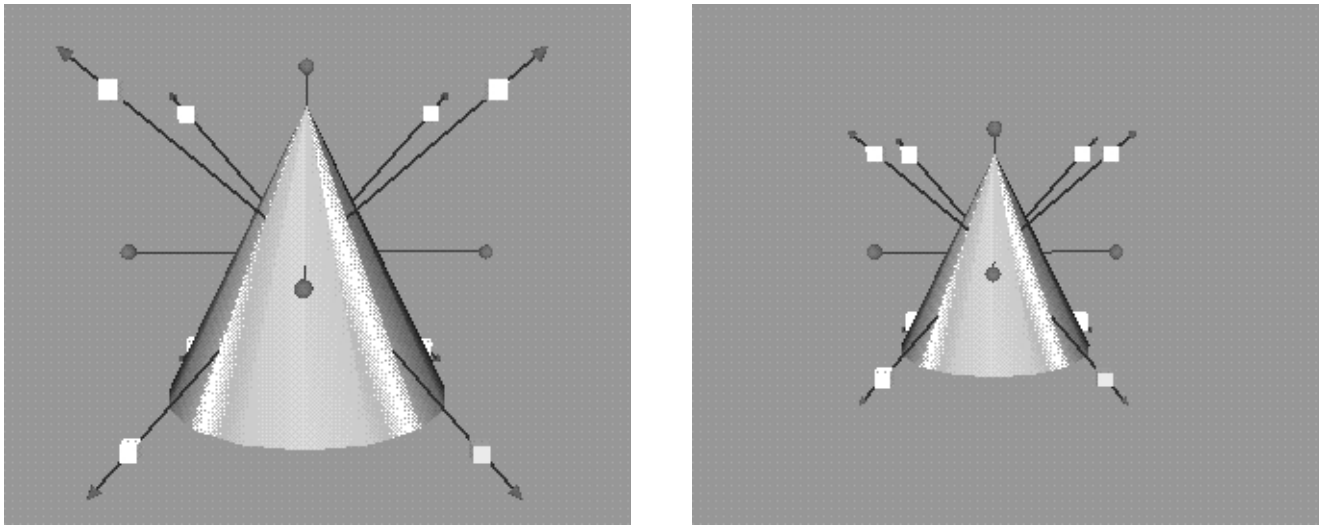


Figure 15-9 Uniform Scaling Sequence

Figure 15-9 shows the sequence of events for uniform scaling. When the user grabs the scaling handle, scaling feedback is displayed. It serves as a guide to the user in dragging to make the object smaller or larger. As the user drags the pointer toward the center of scaling, the object shrinks uniformly in all three dimensions.

Axial 3D Scaling (Stretching)

In addition to uniform scaling, users expect to be able to perform axial scaling (stretching) where the object is scaled or stretched in only one of the three dimensions at a time. Axial scaling is similar to uniform scaling (see Table 15-7). However, as shown in Table 15-8, the grab phase changes to allow selection of a single dimension (axis) for scaling, and the drag phase scales the object only along this single dimension. The user presses the <Shift> modifier key to indicate axial scaling.

Table 15-8 Phases of Axial Scaling (Stretching)

Phase	User Action	Application Response
Approach	Same as for uniform scaling (see Table 15-7).	Same as for uniform scaling.
Grab	User presses and holds <Shift> key and left mouse button while pointer is over the scaling handle of interest.	Same as for uniform scaling with this exception: Scaling feedback consists of the object's bounding box displayed in guide style, and three arrows emanating from the selected scaling handle that represent the three possible axes (dimensions) for stretching (see Figure 15-10). Arrows are displayed in choice style at the location of the selected handle. Feedback also includes rotation handles in neutral style, which provide feedback on the current location of the center of rotation and scaling. (Rotation handles are also part of scaling feedback for uniform scaling.)
Drag	User continues to press <Shift> key and drags scaling handle in the direction of one of the three choice arrows to identify the desired scaling dimension. After the user makes this choice, further dragging scales the object along this single axis.	When user has moved pointer far enough to clearly indicate the axis (dimension) for scaling, the arrow representing this axis changes to selected style and the other two arrows disappear (see Figure 15-10). The remainder of this phase is the same as for uniform scaling with this exception: Scaling is along the selected dimension, not uniformly along all three dimensions. All user input is interpreted to enlarge or shrink the object along the selected dimension.
Release	Same as for uniform scaling.	Same as for uniform scaling.

Figure 15-10 illustrates axial scaling. The user simultaneously presses the <Shift> key and the left mouse button. In response, the application displays the bounding box and choice arrows for axial scaling. As the user drags the pointer, the x axis is selected for scaling. As the user drags the pointer farther away from the center of scaling, the object is enlarged along this single dimension.

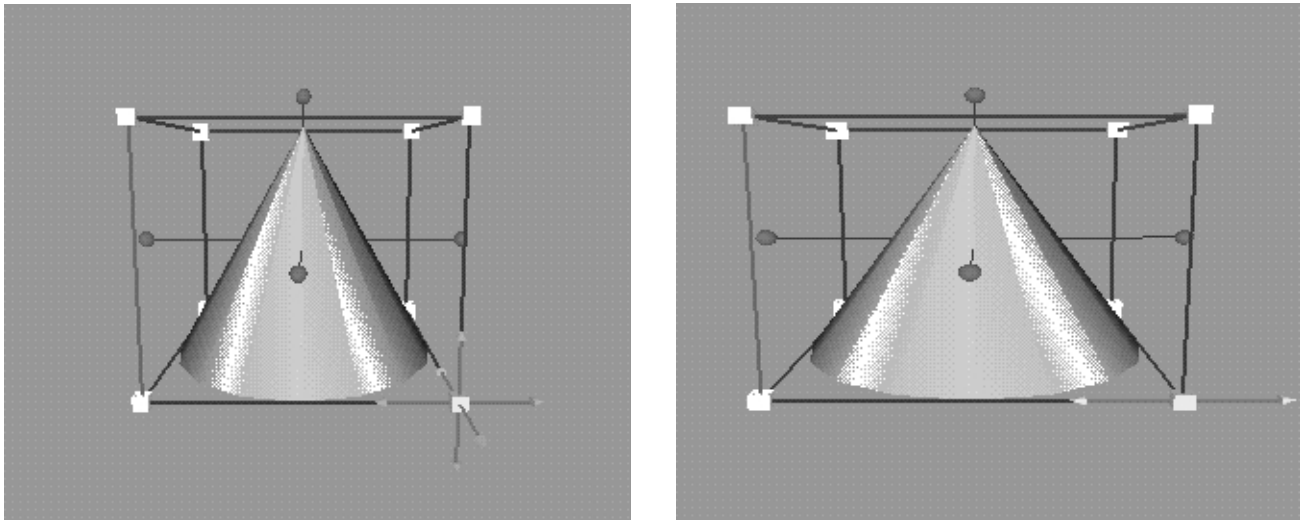


Figure 15-10 Axial Scaling Sequence

Users expect to be able to switch between uniform scaling and axial scaling at any time during a scaling operation:

- If the user is performing uniform scaling and presses the <Shift> key, the application immediately prompts the user to choose a dimension for axial scaling. Once the dimension (axis) has been selected, dragging scales the object only along this single dimension.
- If the user is performing axial scaling and releases the <Shift> key, the application immediately allows the user to uniformly scale the object. Dragging scales the object uniformly along all three dimensions.
- Users can continually press and release the <Shift> key to switch between uniform and axial scaling as long as they continue pressing the left mouse button while the pointer is over a scaling handle.

Scaling Around the Opposite Corner or Side

Users occasionally want to scale an object around a specific corner (scaling handle) for uniform scaling and around a specific plane for axial scaling. To indicate this alternate behavior for scaling, the user presses the <Ctrl> modifier key.

Table 15-9 illustrates user input and resulting application behavior for uniform scaling around a corner. Note that it's similar to uniform scaling around the center of scaling (see Table 15-7).

Table 15-9 Phases of Uniform Scaling Around a Corner

Phase	User Action	Application Response
Approach	Same as for uniform scaling (see Table 15-7).	Same as for uniform scaling.
Grab	User presses and holds <Ctrl> key and left mouse button while pointer is over scaling handle of interest.	Same as for uniform scaling with these exceptions: The set of arrows in the guide style emanate from the scaling handle directly opposite the selected handle (Figure 15-11). Rotation handles aren't shown because scaling isn't around the center of rotation and scaling.
Drag	User continues to press <Ctrl> key and drags scaling handle to scale object in all three dimensions around the corner opposite the selected handle.	Same as for uniform scaling with this exception: Scaling is done around the handle opposite the selected handle, not around the center of scaling (see Figure 15-11).
Release	Same as for uniform scaling.	Same as for uniform scaling.

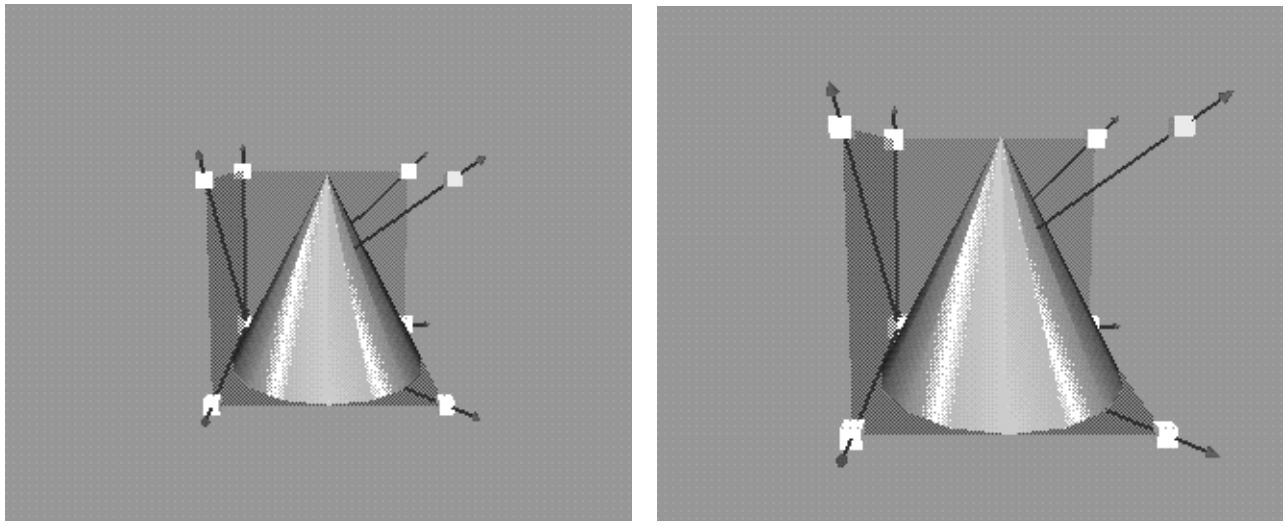


Figure 15-11 Uniform Scaling Around a Corner

Table 15-10 illustrates the same information for axial scaling around a specific side of the object's bounding box. Note that this is similar to axial scaling around the center of scaling (see Table 15-8).

Table 15-10 Phases of Axial Scaling Around a Side

Phase	User Action	Application Response
Approach	Same as for axial scaling (see Table 15-8).	Same as for axial scaling.
Grab	User presses and holds <Ctrl> and <Shift> keys and left mouse button while pointer is over the scaling handle of interest.	Same as for axial scaling with this exception: Rotation handles aren't shown because scaling isn't around the center of rotation and scaling (see Figure 15-12).
Drag	User continues to press <Ctrl> and <Shift> keys and drags scaling handle to indicate the desired scaling dimension. After the user makes this choice, further dragging scales object in this dimension around the plane normal to the selected axis and on the opposite side of the bounding box from the selected handle.	Same as for axial scaling with this exception: Scaling occurs around the side of the bounding box that's normal to the selected axis and on the opposite side of the bounding box from the selected handle (see Figure 15-12).
Release	Same as for axial scaling.	Same as for axial scaling.

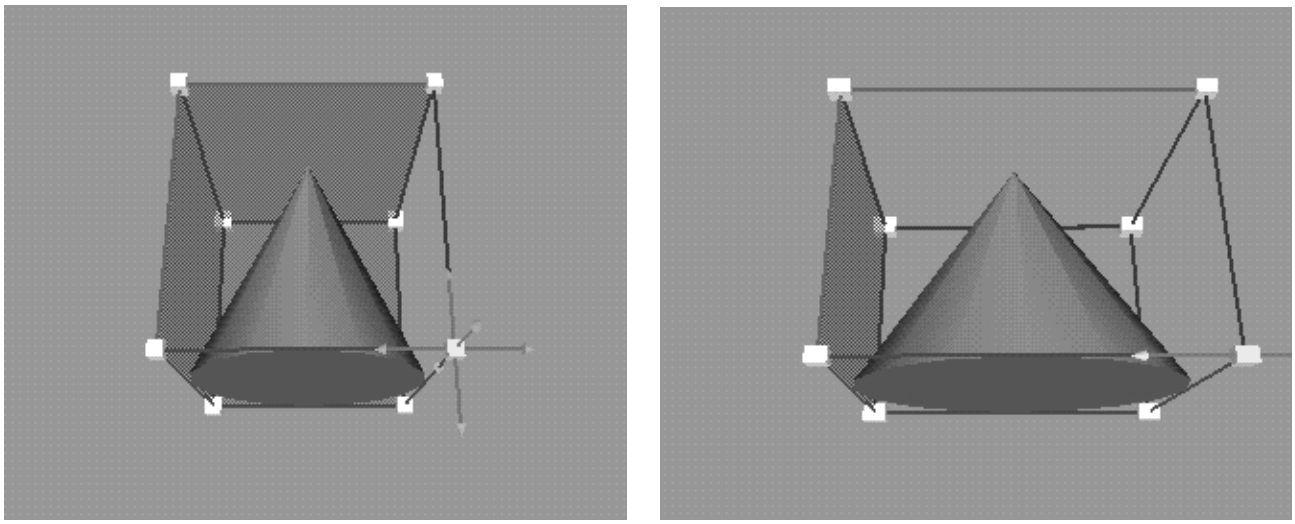


Figure 15-12 Axial Scaling Around a Side

When performing uniform or axial scaling, users expect to be able to change the point of scaling to the opposite corner or opposite side at any point during a scaling operation:

- If the user is performing uniform scaling, pressing the <Ctrl> key immediately starts uniformly scaling the object around the corner opposite to the selected handle. Releasing the <Ctrl> key immediately returns to uniform scaling around the center of scaling.
- If the user is performing axial scaling, pressing the <Ctrl> key immediately starts scaling the object around the plane normal to the selected axis and on the opposite side of the bounding box from the selected handle. Releasing the <Ctrl> key immediately returns to axial scaling around the center of scaling.
- Users can continually press and release the <Ctrl> key to switch between scaling around a corner or side and scaling around the center of scaling as long as they continue pressing the left mouse button while the pointer is over a scaling handle.

3D Scaling User Interface Guidelines

When designing a user interface for 3D scaling...

- ❑ Use the standard scaling manipulator either alone or in combination with the other standard manipulators. The manipulator consists of cube-shaped handles at the vertices of the bounding box.
- ❑ Scale objects around the center of scaling. Typically this is the center of the object (the center of the object's bounding box).
- ❑ Provide uniform scaling as the default scaling method (see Table 15-7 on page 302).
- ❑ Provide access to axial scaling (stretching) as defined in Table 15-8 on page 304.
- ❑ Allow the user to switch between uniform and axial scaling at any point in a scaling operation. For example, if the user is performing axial scaling, then releases the <Shift> key, switch to uniform scaling.
- ❑ As the user switches between uniform and axial scaling, display the appropriate scaling feedback. For example, if the user is performing uniform scaling, then presses the <Shift> key, switch to axial scaling and display the appropriate scaling feedback. Determine the single axis for stretching based on the next pointer movement. Once this axis has been determined, remove the two axes that were not selected and display the selected axis until the <Shift> key is released.
- ❑ Allow users to perform uniform scaling around a specific corner as an alternative to scaling around the center of scaling (see Table 15-9 on page 306).
- ❑ Allow users to perform axial scaling around a specific side of the object's bounding box as an alternative to axial scaling around the center of scaling (see Table 15-10 on page 307).
- ❑ Allow the user to switch between scaling around a corner or side and scaling around the center of scaling at any time during a scaling operation. For example, if the user is performing uniform scaling around the center of scaling, then presses the <Ctrl> key, switch to scaling around the corner opposite to the selected handle.
- ❑ As the user switches between scaling around a corner or side and scaling around the center of scaling, display the appropriate feedback. For example, if the user is performing uniform scaling around a corner, then releases the <Ctrl> key, switch to scaling around the center of scaling and display the appropriate guide feedback for this type of scaling.

Changing the Center of Rotation and Scaling for 3D Objects

At times, users may want to explicitly choose a center for rotation or scaling. For example, you have a graphic object that's a stick figure. An arm is part of the stick figure; its natural centers of rotation are the elbow and the wrists, not the center of the arm.

Typically, the center of rotation and the center of scaling are defined to be the same point. Table 15-11 illustrates user input and the resulting application behavior for changing the center of rotation (along a plane). The user presses the <Ctrl> key while dragging on a rotation handle to change this point. Note that for the standard manipulators recommended by Silicon Graphics, a change in the center of rotation also changes the center of scaling. If your users need to have separate centers of rotation and scaling, you need to provide a separate mechanism for changing the center of scaling.

Table 15-11 Phases of Changing the Center of Rotation and Scaling Along a Plane

Phase[a]	User Action	Application Response[b]
Approach	User moves pointer over a rotation handle.	Rotation handle of interest locate highlights.
Grab	User presses and holds <Ctrl> key and left mouse button while pointer is over rotation handle.	User interface controls on the object, including other manipulators, are removed except for rotation handles. Rotation handle of interest is displayed in selected style. Other handles are displayed in neutral style. Feedback is displayed at the location of the selected handle. Feedback consists of a perpendicular set of arrows in selected style representing the two axes of the plane in which the user can move the point of rotation, and three rings in guide style representing a virtual trackball around the center of rotation (see Figure 15-13). In addition, the object's bounding box is displayed in neutral style to further guide the user in placing the center of rotation. Pointer shape remains upper left pointing arrow.
Drag	User continues to press <Ctrl> key and drags rotation handle to move center of rotation and scaling along selected plane.	Feedback assembly (two arrows and three rings), rotation handles, and center of rotation and scaling move along plane based on user input. All input is interpreted to follow the plane represented by the two arrows. Object and bounding box don't move.
Release	User releases mouse button to stop movement of the center of rotation and scaling.	Feedback consisting of the arrows, rings, and bounding box disappears, and the user interface controls on the object that were removed in the grab phase are again displayed in their original state.

a. See "Phases of 3D Manipulation."

b. See "3D Manipulation Feedback."

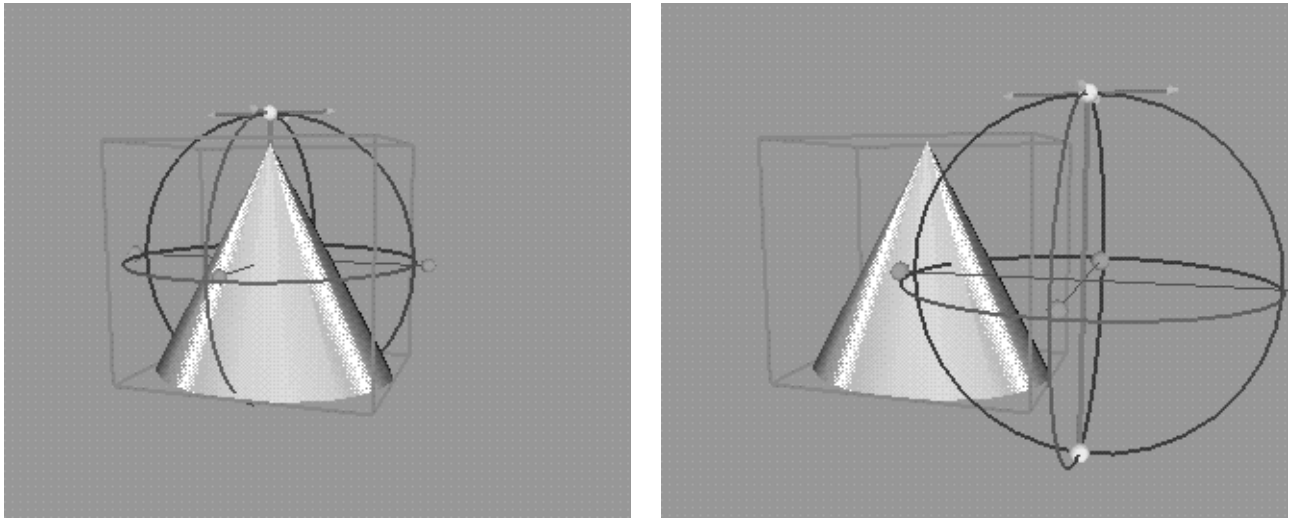


Figure 15-13 Changing the Center of Rotation and Scaling

In addition to moving the center of rotation and scaling along a plane, users may want to constrain the movement to a single axis. This is similar to moving the center along a plane (see Table 15-11). However, as shown in Table 15-12, the grab phase changes to indicate that the user must choose an axis for movement, and the drag phase changes to restrict movement to the selected axis. The user presses the <Ctrl> key while dragging on a rotation handle to change the center of rotation and scaling and also presses the <Shift> key to constrain this movement to a single axis of the plane.

Table 15-12 Phases of Changing the Center of Rotation and Scaling Along an Axis

Phase	User Action	Application Response
Approach	User moves pointer over a rotation handle.	Same as for changing center of rotation and scaling along a plane (see Table 15-11).
Grab	User presses and holds <Shift> and <Ctrl> keys and left mouse button while pointer is over rotation handle.	Same as for changing center of rotation and scaling along a plane with this exception: The perpendicular arrows included in the feedback are displayed in choice style indicating that user must choose an axis for movement (see Figure 15-14).

Table 15-12 (continued) Phases of Changing the Center of Rotation and Scaling Along an

Phase	User Action	Application Response
Drag	User continues to press <Shift> and <Ctrl> keys and drags rotation handle to move center of rotation and scaling along selected axis.	When user has moved the pointer far enough to clearly indicate the axis for movement, arrow representing this axis changes to selected style and the other arrow disappears (see Figure 15-14). The remainder of this phase is the same as for changing the center of rotation and scaling along a plane with this exception: Movement is restricted to the selected axis. All user input is interpreted to follow it.
Release	User releases mouse button to stop movement of the center of rotation and scaling.	Same as for changing the center of rotation and scaling along a plane.

When users are interacting with a rotation handle, they expect at any point to be able to switch among constrained rotation, free rotation, changing the point of rotation along a plane, and changing the point of rotation along an axis.

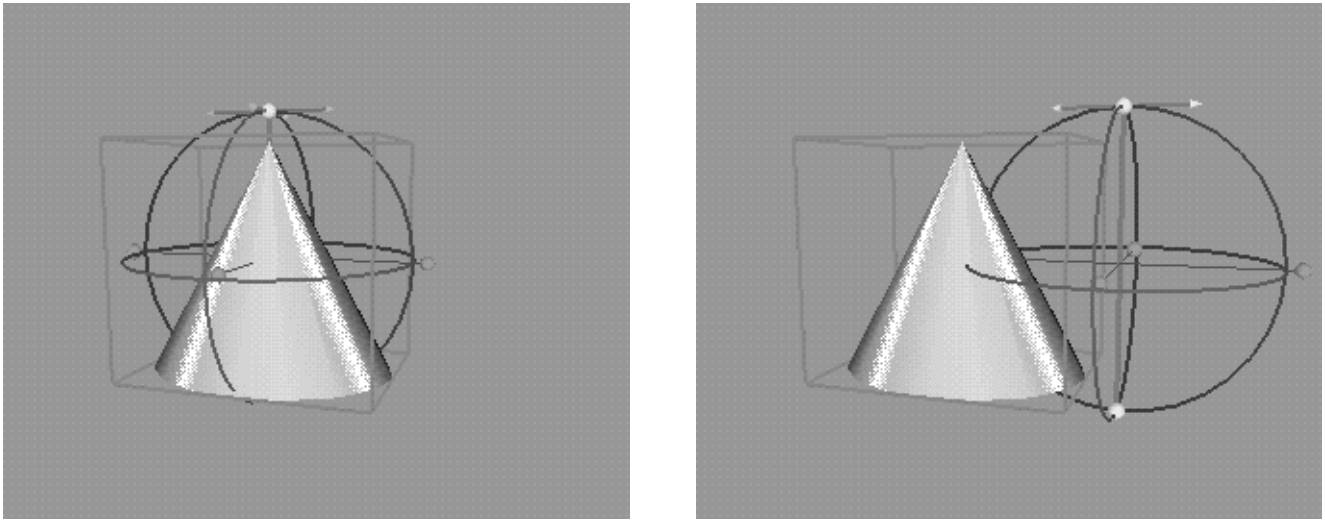


Figure 15-14 Changing the Center of Rotation and Scaling Along An Axis

Guidelines for Changing the Center of 3D Rotation

When designing a user interface for changing the center of rotation and scaling...

- ❑ If necessary, allow users to change the center of rotation and scaling using the standard rotation manipulator.
- ❑ As users move the center of rotation and scaling, make sure that they can always identify the current center.
- ❑ Allow users to change the center of rotation and scaling along a plane (see Table 15-11 on page 310).
- ❑ Allow users to change the center of rotation and scaling along an axis (see Table 15-12 on page 311).
- ❑ At any time while dragging a rotation handle, allow users to switch among constrained rotation, free rotation, changing the center of rotation and scaling along a plane, and changing the center of rotation and scaling along an axis. For example, if the user is performing constrained rotation then presses the <Ctrl> and <Shift> keys, switch to changing the center of rotation and scaling along an axis.
- ❑ As the user switches among constrained rotation, free rotation, changing the center of rotation and scaling along a plane, and changing the center of rotation and scaling along an axis, display the appropriate feedback. For example, if the user is moving the center of rotation and scaling along a plane then presses the <Shift> key, switch to moving this center along an axis and display the appropriate feedback. Determine the single axis for movement based on the next pointer movement. Once this axis has been determined, remove the axis that wasn't selected and display the selected axis until the <Shift> key is released.

Object Manipulation for Multiple Selected 3D Objects

When the current selection consists of more than one object, the standard manipulations of translation, rotation, and scaling affect the selected objects somewhat differently than if each object was manipulated by itself. To perform these manipulations, one of the objects in the selection is assigned as the "key" object; manipulations on all of the objects in the selection are done with respect to the manipulation controls of the key object. For example, all selected objects are translated along the selected plane of the key object.

If your application supports the concept of a lead object and uses the manipulator as the selection feedback only on this object as described in "Manipulator Selection Feedback" in Chapter 14, the lead object is always also the key object because it's the only object displaying the manipulators. Otherwise, the key object is the one that the user is currently interacting with via its manipulators.

The remainder of this section discusses the specific manipulations as follows:

- "Translation of Multiple Selected 3D Objects"
- "Rotation of Multiple Selected 3D Objects"
- "Scaling of Multiple Selected 3D Objects"

Translation of Multiple Selected 3D Objects

When users perform translation on a selection consisting of several objects, all selected objects move in the selected translation plane of the key object, regardless of their own orientations. The relative positional relationships of the selected objects don't change as a result of translation. The same behavior applies to constrained translation and translation along the normal. That is, translation is done with respect to the selected axis of the key object.

Translation of a single object is discussed in detail in "Translating 3D Objects."

Rotation of Multiple Selected 3D Objects

When the user selects more than one object and starts a rotation operation, all selected objects rotate around the center of rotation for the key object.

Rotation of a single object is discussed in detail in "Rotating 3D Objects."

Scaling of Multiple Selected 3D Objects

When the user selects more than one object and starts a uniform scaling operation, all objects scale proportionally with the key object and scale about the same center of scaling as the key object.

Axial scaling (stretching) is treated quite differently when the user selects more than one object. In this case, only the key object is scaled; other selected objects don't change. While this may seem like an inconsistency in light of applying other manipulations to all selected objects, it's the safest default approach. Applying stretching to objects that are not aligned with the master object can lead to shearing, which is an undesirable outcome.

Guidelines for Manipulating More Than One 3D Object

When users manipulate more than one object at a time...

- Assign one object as the key object for the manipulations. Typically, only the lead object displays the manipulators so this object is by default the key object. If your application instead displays manipulators on all selected objects, the object associated with the manipulator the user is currently interacting with is the key object.
- For translation, translate all selected objects along the chosen translation plane for the key object, regardless of the orientation of the other selected objects. Don't change the relative positional relationships of the selected objects.
- For rotation, rotate all selected objects around the center of rotation for the key object.
- For uniform scaling, scale all selected objects the same amount as the key object and about the same center of scaling as the key object.
- For axial scaling (stretching), scale only the key object (the object being manipulated) rather than all selected objects. This prevents shearing of objects that are not aligned with the key object.

Summary of Guidelines

This appendix contains an all-inclusive list of the guidelines for designing applications for the IRIX Interactive 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:

- Integrating with the IRIX Interactive Desktop
  - "Icon Appearance Design Guidelines" on page 319
  - "Icon Behavior Guidelines" on page 321
  - "Application Icon Accessibility Guidelines" on page 322
  - "IRIX Interactive Desktop Look Guidelines" on page 322
  - "Application Window Characteristic Guidelines" on page 322
  - "Guidelines for Keyboard Focus Across Windows" on page 325
  - "Minimized Window Guidelines" on page 326
  - "Desks Guidelines" on page 327
  - "Session Management Guidelines" on page 327
  - "Software Installation Guideline" on page 328
  - "Guidelines for Designing Online Help" on page 328
  - "Guidelines for Creating SGIHelp Content" on page 330
  - "Desktop Variables Guidelines" on page 332
  - "File Monitoring Guideline" on page 333
  - "Data Exchange Guidelines" on page 333

- Interface Components
  - "Application Model Guidelines" on page 334
  - "Primary Window Guidelines" on page 334
  - "Support Window Guidelines" on page 336
  - "Pointer Behavior Guidelines" on page 336
  - "Keyboard Focus and Navigation Guidelines" on page 337
  - "Selection Guidelines" on page 337
  - "Drag and Drop Guidelines" on page 338
  - "Menu Traversal and Activation Guidelines" on page 338
  - "Pull-Down Menu Guidelines" on page 339
  - "Popup Menu Guidelines" on page 342
  - "Tab Panel Guidelines" on page 343
  - "Pushbutton Guidelines" on page 343
  - "Option Button Guidelines" on page 345
  - "Checkbox Guidelines" on page 346
  - "Radio Button Guidelines" on page 347
  - "LED Button Guidelines" on page 348
  - "List Guidelines" on page 348
  - "Text Field Guidelines" on page 349
  - "Scrollbar Guidelines" on page 350
  - "IRIX Interactive Desktop Scale Guidelines" on page 352
  - "Label Guidelines" on page 352
  - "File Finder Guidelines" on page 353
  - "Thumbwheel Guidelines" on page 354
  - "Dial Guidelines" on page 354
  - "Guidelines for Using the Various Types and Modes of Dialogs" on page 355
  - "Guidelines for Designing Dialogs" on page 356
  - "Guidelines for Invoking Dialogs" on page 358
  - "General User Feedback Guidelines" on page 360
  - "Pointer Shapes and Colors Guidelines" on page 360

- 3D Style Guidelines
  - "Basic 3D Interface Design Guidelines" on page 361
  - "Pointer Feedback Guidelines for 3D Applications" on page 362
  - "Guidelines for Resizing Windows in 3D Applications" on page 362
  - "Guidelines for 3D Viewing Functions" on page 363
  - "3D Viewing Trade-Offs and Related Guidelines" on page 365
  - "3D Selection Design Guidelines" on page 367
  - "3D Selection Feedback Design Guidelines" on page 368
  - "Basic 3D Manipulation Guidelines" on page 369
  - "3D Translation User Interface Guidelines" on page 371
  - "3D Rotation User Interface Guidelines" on page 372
  - "3D Scaling User Interface Guidelines" on page 372
  - "Guidelines for Changing the Center of 3D Rotation" on page 373
  - "Guidelines for Manipulating More Than One 3D Object" on page 374

Guidelines for Integrating With the IRIX Interactive Desktop

The following guidelines are extracted from Part I, "Integrating with the IRIX Interactive Desktop," of this guide:

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 IRIX Interactive 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 moving the magic carpet from a horizontal (not running) to vertical (running) position, or by providing two different application symbols and moving the magic carpet from a horizontal to vertical 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.
- ❑ Application icons that have two separate symbols for the not running and running states should make sure that the main part of the symbol remains in the same location during both states, so that the symbol appears stationary to the user.

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. Don't use an abbreviation of the product name.
- ❑ 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.

IRIX Interactive Desktop Look Guidelines

When designing the look for your application . . .

- ❑ Use the IRIX Interactive Desktop 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 ...* .
- ❑ Don't include full pathnames unless that information is required by users to distinguish one window from another. For remote applications, don't include domain information.
- ❑ 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.

For windows without title bars . . .

- ❑ Display the "Exit" option with the right mouse button.
- ❑ Allow users to resize the window with the left mouse button.

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 IRIX Interactive 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.

Interface Component Guidelines

The following guidelines have been extracted from Part II, "Interface Components," of this guide.

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 IRIX Interactive Desktop 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 entire one-word titles for menus rather than abbreviations.
- ❑ 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.
- ❑ Use entire words rather than abbreviations.
- ❑ 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.

Tab Panel Guidelines

When binding Page Up and Page Down to tab panels . . .

- ❑ If your application is based on presenting a page at a time and the user model is that tabs are used to move through the pages, then bind Page Up and Page Down to the tabs. In this case, <Ctrl><up arrow> and <Ctrl><down arrow> moves the vertical scrollbar a window at a time, allowing the user to pan around the current page.
- ❑ If the user model is that a tab moves to another "document," and the scrollbar is used to view various pages in the current document, then bind Page Up and Page Down to the scrollbar. In this case, no keyboard accelerator exists for moving to the next/previous tab.

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 labeling 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. Don't use abbreviations in labels.
- ❑ 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 labeling 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 and is not abbreviated.

When labeling the entries in an option menu . . .

- ❑ Use nouns that indicate the possible values of the parameter being set.
- ❑ Use entire words for the entries rather than abbreviations.

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 labeling 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.
- ❑ Don't use abbreviations for either the checkbox labels or the group label.

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 labeling 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.
- ❑ Don't use abbreviations for either the radio button labels or the group label.

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.

LED Button Guidelines

When using LED buttons . . .

- ❑ Use LED buttons 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.)
- ❑ Don't use LED buttons 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 labeling LED buttons . . .

- ❑ Label each LED button with a term that describes the attribute, state, or option it controls.
- ❑ If there is a group of LED buttons, create a group label for the group, and indent the LED buttons below the label. Use a noun that describes the function of the group.
- ❑ Don't use abbreviations for either the LED button labels or the group label.

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 labeling a list . . .

- ❑ Label the list with a noun that indicates the function of the elements in the list. Don't use abbreviations in the label.
- ❑ Place the label directly above and either left-aligned with or slightly to the left of the first element of the list.

When labeling the list entries . . .

- ❑ If the elements in the list represent operations to perform, they should be active verbs. Otherwise, they should be nouns. In either case, use entire words rather than abbreviations.

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 IRIX Interactive Desktop File Finder instead.
- ❑ Use text fields for values that change over time; don't use labels.

When labeling 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. Use entire words in labels rather than abbreviations.
- ❑ 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 IRIX Interactive Desktop thumbwheel instead.
- ❑ Don’t use scrollbars to choose a value in a range; use the IRIX Interactive Desktop 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.

IRIX Interactive Desktop Scale Guidelines

When using the IRIX Interactive Desktop 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 labeling 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. Don't use abbreviations in this label.

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 entire words in labels rather than abbreviations.
- ❑ 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 labeling 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).
- ❑ Use entire words in the label rather than abbreviations.

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 IRIX Interactive Desktop 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 IRIX Interactive Desktop 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 IRIX Interactive Desktop 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 IRIX Interactive Desktop 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 IRIX Interactive Desktop 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.

3D Style Guidelines

The following guidelines have been extracted from Part III, "3D Style Guidelines," of this guide.

Basic 3D Interface Design Guidelines

To provide mouse functionality that matches user's expectations in a 3D application...

- ❑ Assign primary functionality to the left mouse button.
- ❑ Use modifier keys in conjunction with the left mouse button to make additional functionality available.
- ❑ Reserve the middle mouse button for primary copy and paste operations and to provide access to advanced user shortcuts that are also accessible via a more obvious user interface.
- ❑ Reserve the right mouse button for popup menus.

When deciding on the use of modifier keys in a 3D application...

- ❑ Use the <Shift> key to constrain or unconstrain the default behavior of the left mouse button.
- ❑ Use the <Ctrl> key to allow alternate behaviors that aren't thought of as constraining or unconstraining the default behavior assigned to the left mouse button.
- ❑ Use <Shift> and <Ctrl> together where it makes sense to provide alternate behavior that's also constrained or unconstrained.
- ❑ Reserve the <Alt> key for a view overlay (see "View Overlay" in Chapter 13).
- ❑ Reserve the <Esc> key for mode switches, for example, switching from view mode to edit mode.

Pointer Feedback Guidelines for 3D Applications

To help users stay oriented while working in your 3D application...

- ❑ Use the standard pointer shapes to indicate specific 3D functionality. Display these pointers whenever the user accesses that functionality.
- ❑ Use the standard pointers used in 2D applications as needed (for example, the watch pointer to indicate an operation is in progress).

Guidelines for Resizing Windows in 3D Applications

When designing the resizing operation for the viewports in your 3D application...

- ❑ If the user resizes the window and keeps its same aspect ratio, make the scene in the viewing area of the window proportionally bigger or smaller based on the resize action.
- ❑ If the user resizes the window and changes its aspect ratio during the resize operation, display more or less of the scene being displayed based on the resize action, but don't change the size of the objects in the scene.

Guidelines for 3D Viewing Functions

When designing the user interface for a 3D application...

- ❑ Provide a viewing interface regardless of other capabilities of the application (for example, editing).

When designing the user interface for 3D viewing...

- ❑ Decide whether your application will support inspection, navigation, or both, then provide the appropriate viewing functions. If your application supports both inspection and navigation, choose one as the primary mode for viewing.
- ❑ Use standard pointer shapes to indicate the current 3D viewing function.

When designing the user interface for inspection in a 3D application...

- ❑ Support the user model that users are manipulating a scene as though it were a single object they are holding in their hand (not the user model that users are manipulating a camera). From the user's perspective, all controls appear to manipulate the object or scene while the camera remains stationary.
- ❑ Support tumbling as the default inspection function to allow users to view all sides of the scene.
- ❑ Assign tumbling to dragging with the left mouse button.
- ❑ Display the tumble pointer while the user accesses the tumble function.
- ❑ Support dollying to allow users to move the scene closer or farther away.
- ❑ Assign dollying to dragging with the left and middle mouse buttons pressed simultaneously.
- ❑ Display the dolly pointer while the user accesses the dolly function.
- ❑ Support panning to allow users to move the scene left, right, up, or down.
- ❑ Assign panning to dragging with the middle mouse button.
- ❑ Display the pan pointer while the user accesses the panning function.
- ❑ Support seeking to allow users to change the look-at point and center the object of interest and to bring the object incrementally closer.

❑ Support seeking as follows:

- If your application needs to reserve clicking with the left mouse button for a more critical or useful function, allow users to seek by first activating a seek tool, then clicking with the left mouse button in the scene. Otherwise, support seeking without the use of a tool.
- In either case, the user seeks by clicking on a part of the scene with the left mouse button. The application centers that part of the scene in the viewing window and moves the scene closer by half the distance between the camera and the object.
- With each subsequent click on the same part of the scene, the scene again moves closer.

❑ If your application needs to reserve clicking with the left mouse button for a more critical function, allow users to seek by first activating a seek tool, then clicking with the left mouse button in the scene.

❑ Display the seek pointer while the user accesses the seek function.

When designing the user interface for navigation in a 3D application...

❑ Support the user model that the scene is stationary and the user is moving through this fixed, immovable world. From the user's perspective, all navigation controls appear to manipulate the camera (user's view into the world) while the scene remains stationary.

❑ Support roaming as the default navigation function. In roaming, the user can move forward and backward, turn left and right, and turn to move in a new direction.

❑ Assign roaming to dragging with the left mouse button.

❑ Display the roam pointer while the user accesses the roaming function.

❑ Support tilting to allow users to change their view of the scene by tilting their head up and down. Tilting doesn't move the user forward or backward.

❑ Assign tilting to dragging with the left and middle mouse buttons pressed simultaneously.

❑ Display the tilt pointer while the user accesses the tilting function.

❑ Support sidling to allows users to sidestep left and right and to move up and down as if on an elevator.

❑ Assign sidling to dragging with the middle mouse button.

❑ While the user accesses the sidling function, display the sidle pointer.

❑ Support seeking to allow users to move closer to an object in the scene by clicking instead of dragging.

❑ Support seeking as follows:

- The user clicks on an object in the scene with the left mouse button when the application is in view mode. The application automatically moves the user through the scene and closer to that object.
- With each subsequent click on the same object in the scene, the user moves closer to that object.

❑ Display the seek pointer while the user accesses the seek function.

3D Viewing Trade-Offs and Related Guidelines

To make viewing quickly and easily accessible in 3D applications...

❑ Always provide ready access to viewing no matter what the user is doing (for example editing).

When designing a viewing interface for a 3D application that also supports editing...

❑ Display the appropriate pointer depending on the task the user is performing:

- While the user is accessing editing functions, display the edit pointer.
- While the user is accessing viewing functions, display the appropriate view pointer based on the user's current viewing function (for example, the roaming pointer if the user is currently navigating a scene).

❑ Provide a modal interface to viewing and editing whenever possible.

❑ Provide an obvious mechanism for changing between the view and edit modes, such as buttons in a tool palette or entries in a pull-down menu.

❑ Reserve the <Esc> key for switching between the view and edit modes.

❑ Always provide a view overlay for quick access to viewing. That is, when the primary task is editing, the user can at any time temporarily enter a view mode by pressing and holding the <Alt> key. The user can release the <Alt> key to return the application to edit mode.

❑ Reserve the <Alt> key for providing access to a view overlay. If the user is already in view mode, the <Alt> key has no effect.

❑ Display the appropriate pointer for the current viewing function (for example, the tumble pointer or the roaming pointer) while the user is accessing a view overlay.

❑ Optionally provide additional ways to access viewing, for example, offer viewing fixtures or split viewing and editing input across separate dedicated input devices.

When deciding between a single viewport and multiple viewports...

- ❑ Use a single viewport if the user doesn't need to do much editing, performance or screen real estate is critical, you need a simple user model, or if several of these conditions are met.
- ❑ Support multiple viewports if the user needs two or more views of the data simultaneously (such as when editing complex objects or working on scene composition) and performance isn't a critical issue.

When designing a viewing interface for a single viewport...

- ❑ Use the perspective view of the scene as the default view.
- ❑ Update the single-viewport view with a new view as the user selects different cameras.

When making viewing performance design decisions...

- ❑ Support a minimum frame rate of 8 fps when the user is interacting with the view.
- ❑ Ideally, support a minimum rate of 10-12 fps for editing and a minimum frame rate of 15 fps for a realistic interactive experience.
- ❑ If the frame rate drops below 8 fps, provide at least one of the following solutions:

- Automatic adaptive rendering, where the application always maintains an acceptable frame rate at the expense of scene fidelity.
- User-controlled adaptive rendering, where the user explicitly chooses between adaptive rendering (acceptable frame rate but loss of detail) and fully rendering the contents of the scene (at a possibly unacceptably low frame rate).

- ❑ If users sometimes need fully rendered, high-fidelity scenes and the frame rate is likely to drop below 8 fps, provide user-controlled adaptive rendering.
- ❑ If you application only provides automatic adaptive rendering, provide users ready access to fully rendered scenes. At a minimum, this should happen when the user stops interacting with the view.

3D Selection Design Guidelines

When designing the selection user interface for your 3D application...

- ❑ Follow the object-action paradigm of direct manipulation: The user first selects an object (or group of objects), then chooses an action to perform on it.
- ❑ For actions that apply to objects, apply the action to all the objects in the current selection and only to those objects.
- ❑ Provide one current selection for each application at any time. The current selection may be empty (that is, no selection). Note that each window of your application can maintain a separate selection, but there is only one current selection.
- ❑ Support direct selection as the primary selection mechanism. Using the left mouse button, the user either clicks directly on an object to select it or sweeps out an area to select multiple objects.
- ❑ Support indirect selection if your users need it. For example, allow users to select an indirect representation of an object such as an item in a list, or provide a "Select All" menu item.
- ❑ If your application lets users select more than one object in a hierarchy of objects, provide at a minimum a method for selecting the highest and lowest object in the hierarchy and a method for adjusting the selection up and down the hierarchy.

When deciding on a selection model...

- ❑ If your application allows users to select only one object at a time, support the OSF/Motif single selection model as follows:

- Users directly select an object by moving the pointer over it and pressing the left mouse button. Any previously selected object is deselected.
- Users deselect an object by clicking outside the selection in an empty area.

- ❑ If your application allows users to select more than one object at a time, support the OSF/Motif discontiguous selection model as follows. Ideally, support the entire OSF/Motif discontiguous selection model.

- Users directly select an object by moving the pointer over it and pressing the left mouse button. Any previously selected objects are deselected.
- Users <Ctrl>-click an unselected object to add it to the current selection, and <Ctrl>-click an already selected object to remove it from the current selection. That is, <Ctrl>-click toggles the selection state of the object.

- If users are accustomed to using the <Shift> key in other applications to toggle the selection state of an object, allow them to add and remove objects by <Shift>-clicking an object in addition to <Ctrl>-clicking an object. In that case, <Shift>-clicking an object performs the same selection actions as <Ctrl>-clicking an object.
- Users deselect all objects by clicking outside the selection in an empty area. In addition, a "Deselect All" menu item may be useful for some applications.
- Optionally, allow users to use sweep selection to select multiple objects, allowing either rectangle-drag or lasso-drag. At the end of a sweep action, the only objects selected are those inside the sweep area.
- Optionally allow users to <Ctrl>-sweep through a collection of objects to toggle the selection state of all objects inside the sweep area. That is, objects inside the sweep area that were previously selected are deselected, and objects inside the sweep area that were previously deselected are selected. The selection state of objects outside the sweep area doesn't change.
- If your application supports the optional <Ctrl>-sweep selection action and users are accustomed to using the <Shift> key in their other applications to toggle the selection state of an object, allow users to use <Shift>-sweep in addition to <Ctrl>-sweep to toggle the selection state of all objects inside the sweep area.

3D Selection Feedback Design Guidelines

When designing selection feedback for your 3D application...

- ❑ Provide clear feedback on each object as it is selected.
- ❑ When using a bounding box for selection feedback, make sure that it is differently shaped or larger than the object itself so that it is readily visible. Using a distinct color for the bounding box is also highly recommended.
- ❑ If users don't typically select objects in your application to manipulate them (translate, rotate, scale) or can't manipulate the selected objects, use bounding boxes to indicate the selected objects.
- ❑ If users typically select objects in your application to manipulate them, use the manipulator as selection feedback. If your application allows more than one object to be selected at a time, consider displaying the manipulator only on a lead object and bounding boxes on the other selected objects.
- ❑ If your application needs to always present a realistic "experience-oriented" representation of objects in the scene, highlight selected objects in some way rather than cluttering the scene with bounding boxes or manipulators.

Lead Object Design Guidelines for 3D Applications

When designing the selection user interface for your 3D application...

- ❑ If your application allows users to select more than one object at a time, consider identifying one of those objects as the lead object.
- ❑ Clearly distinguish the lead object from other selected objects:
  - If users typically select objects to manipulate them, consider making this distinction by displaying the manipulator only on the lead object and a bounding box on all other selected objects.
  - Otherwise, if there are manipulators or bounding boxes on all selected objects, distinguish the lead object another way (color, full manipulator instead of partial, and so on.).
- ❑ If the user clicks to define a selection or <Ctrl>-clicks to add to a selection, make the lead object the last object selected. This allows users to change the lead object using click or <Ctrl>-click on an object that's currently not selected, or using <Ctrl>-click twice on a currently selected object.
- ❑ If the user employs a single action such as a sweep selection to select multiple objects at the same time, make the lead object the one that is closest to the camera and closest to the middle of the viewport.
- ❑ When the user deselects the lead object, move back through the previous lead objects making the most recent lead object that is still selected the new lead object.
- ❑ If the user groups a collection of objects, make the group the new lead object.
- ❑ When the user ungroups a grouped collection of objects, each object that was in the group becomes selected, and the object from the group that is closest to the camera and closest to the middle of the viewport becomes the new lead object.

Basic 3D Manipulation Guidelines

When designing a user interface for object manipulation in a 3D application...

- ❑ Let users manipulate objects directly whenever possible.
- ❑ If an intermediary, such as a dialog box, yields more precise results for an action and your users need precise manipulations, provide an intermediary method in addition to direct manipulation.
- ❑ Provide manipulators to allow users to use direct manipulation when editing objects.

- ❑ Use the standard manipulators provided by Silicon Graphics for translation, rotation, and scaling.
- ❑ Make sure users can readily identify a manipulator's controls and how to interact with those controls. Also, make sure the controls allow users to precisely manipulate an object.
- ❑ Provide immediate feedback on available actions during the different stages of manipulation as follows:
  - Display phase—Display the manipulator in its neutral state.
  - Approach phase—As the pointer passes over a control on the manipulator, locate highlight the control to show that it's a live, functional control.
  - Grab phase—Provide selected feedback if there is no additional choice to make; otherwise provide choice feedback. Provide guide feedback as appropriate to facilitate user interaction.
  - Drag phase—If the user was presented with a choice in the grab phase, resolve it when the user begins the drag, and replace the choice feedback with selected feedback.
  - Release phase—Return the manipulator to its neutral state.
- ❑ Use clearly distinguishable feedback styles as follows:
  - Provide neutral feedback using a style that stands out from the scene; for example, use the color green or white.
  - Provide locate highlight for manipulator controls. That is, the controls brighten (often to an orange color) as the pointer passes over them.
  - Provide selected feedback using a style that stands out strongly; for example, use the color yellow.
  - Provide choice feedback using a style that stands out, but not as strongly as selection feedback; for example, use the color orange.
  - Provide guide feedback using a style that recedes but is distinct from the other styles; for example, use the color purple.
- ❑ Provide continuous feedback on the status of an object while the user is manipulating it. For example, as the user translates an object, the object should move in the scene in a smooth and continuous fashion to keep the user updated on the location of the object. Use adaptive rendering if necessary.
- ❑ Make commonly used and critical manipulation techniques immediately available on the manipulator via the left mouse button. Make less commonly used techniques available on a secondary level, for example, though modifier keys used in conjunction with the left mouse button.

When displaying manipulators in your 3D application...

- ❑ Decide which manipulators to display by default based on the functional requirements of your users and their most common tasks.
- ❑ Allow users to change which subset of manipulators are currently displayed. At a minimum, allow users to specify this subset using entries in the Edit menu.

3D Translation User Interface Guidelines

When designing a user interface for 3D translation...

- ❑ Use the standard translation manipulator either alone or in combination with the other standard manipulators. This manipulator is a set of translation planes arranged to define a bounding box around the object.
- ❑ Allow users to perform translation in the planes of the object's bounding box. These planes may or may not be aligned with the world (scene) coordinate space.
- ❑ Don't allow users to translate objects that are part of a scene into unusable locations, such as behind the camera or so far back along the z axis that they vanish from view.
- ❑ Provide planar translation as the default translation method (see Table 15-2 on page 289).
- ❑ Provide access to constrained translation along only one axis of a plane (see Table 15-3 on page 291).
- ❑ Provide access to axial (constrained) translation along the normal to a plane (see Table 15-4 on page 293).
- ❑ Allow users to switch from planar translation to constrained translation and back at any point in a translation. For example, if the user is performing planar translation by dragging a translation plane and then presses the <Shift> key, switch to axial translation along the plane until the <Shift> key is released.
- ❑ Display the appropriate translation feedback as the user switches between unconstrained and constrained translation.
- ❑ Don't allow users to select objects or controls through the translation planes.

3D Rotation User Interface Guidelines

When designing a user interface for 3D rotation...

- ❑ Use the standard rotation manipulator either alone or in combination with the other standard manipulators. This manipulator is a set of handles that emanate from the center of rotation. Typically, the default center of rotation is the center of the object (the center of the object's bounding box).
- ❑ Provide constrained rotation around an axis as the default rotation method (see Table 15-5 on page 296).
- ❑ Provide access to free rotation around a point (see Table 15-6 on page 298).
- ❑ Allow users to switch between constrained and free rotation at any point in a rotation. For example, if the user is performing constrained rotation by dragging along a ring and then presses the <Shift> key, switch to free rotation and interpret pointer movements as following the virtual trackball until the <Shift> key is released.
- ❑ As the user switches between constrained and free rotation, display the appropriate rotation feedback. For example, if the user is performing free rotation and then releases the <Shift> key, switch to constrained rotation and display the appropriate rotation feedback. Determine the direction and axis for rotation based on the next pointer movement. Once the direction and axis have been determined, display the appropriate arrow and ring feedback for this direction and remove the virtual trackball.

3D Scaling User Interface Guidelines

When designing a user interface for 3D scaling...

- ❑ Use the standard scaling manipulator either alone or in combination with the other standard manipulators. The manipulator consists of cube-shaped handles at the vertices of the bounding box.
- ❑ Scale objects around the center of scaling, by default the center of the object (the center of the object's bounding box).
- ❑ Provide uniform scaling as the default scaling method (see Table 15-7 on page 302).
- ❑ Provide access to axial scaling (stretching) as defined in Table 15-8 on page 304.

- ❑ Allow the user to switch between uniform and axial scaling at any point in a scaling operation. For example, if the user is performing axial scaling then releases the <Shift> key, switch to uniform scaling.
- ❑ As the user switches between uniform and axial scaling, display the appropriate scaling feedback. For example, if the user is performing uniform scaling then presses the <Shift> key, switch to axial scaling and display the appropriate scaling feedback. Determine the single axis for stretching based on the next pointer movement. Once this axis has been determined, remove the two axes that were not selected and display the selected axis until the <Shift> key is released.
- ❑ Allow users to perform uniform scaling around a specific corner as an alternative to scaling around the center of scaling (see Table 15-9 on page 306).
- ❑ Allow users to perform axial scaling around a specific side of the object's bounding box as an alternative to axial scaling around the center of scaling (see Table 15-10 on page 307).
- ❑ Allow the user to switch between scaling around a corner or side and scaling around the center of scaling at any time during a scaling operation. For example, if the user is performing uniform scaling around the center of scaling then presses the <Ctrl> key, switch to scaling around the corner opposite to the selected handle.
- ❑ As the user switches between scaling around a corner or side and scaling around the center of scaling, display the appropriate feedback. For example, if the user is performing uniform scaling around a corner then releases the <Ctrl> key, switch to scaling around the center of scaling and display the appropriate guide feedback for this type of scaling.

Guidelines for Changing the Center of 3D Rotation

When designing a user interface for changing the center of rotation and scaling...

- ❑ If necessary, allow users to change the center of rotation and scaling using the standard rotation manipulator.
- ❑ As users move the center of rotation and scaling, make sure that they can always identify the current center.
- ❑ Allow users to change the center of rotation and scaling along a plane (see Table 15-11 on page 310).

- Allow users to change the center of rotation and scaling along an axis (see Table 15-12 on page 311).
- At any time while dragging a rotation handle, allow users to switch among constrained rotation, free rotation, changing the center of rotation and scaling along a plane, and changing the center of rotation and scaling along an axis. For example, if the user is performing constrained rotation then presses the <Ctrl> and <Shift> keys, switch to changing the center of rotation and scaling along an axis.
- As the user switches among constrained rotation, free rotation, changing the center of rotation and scaling along a plane, and changing the center of rotation and scaling along an axis, display the appropriate feedback. For example, if the user is moving the center of rotation and scaling along a plane then presses the <Shift> key, switch to moving this center along an axis and display the appropriate feedback. Determine the single axis for movement based on the next pointer movement. Once this axis has been determined, remove the axis that was not selected and display the selected axis until the <Shift> key is released.

Guidelines for Manipulating More Than One 3D Object

When users manipulate more than one object at a time...

- Assign one object as the key object for the manipulations. Typically, only the lead object displays the manipulators so this object is by default the key object. If your application instead displays manipulators on all selected objects, the object associated with the manipulator the user is currently interacting with is the key object.
- For translation, translate all selected objects along the chosen translation plane for the key object, regardless of the orientation of the other selected objects. Don't change the relative positional relationships of the selected objects.
- For rotation, rotate all selected objects around the center of rotation for the key object.
- For uniform scaling, scale all selected objects the same amount as the key object and about the same center of scaling as the key object.
- For axial scaling (stretching), scale only the key object (the object being manipulated) rather than all selected objects. This prevents shearing of objects that are not aligned with the key object.

Index

Numbers

3D applications
  frame rate, 263
  manipulation phases, 282
  modifier keys, 235
  mouse input, 234
  selection, 267
  viewing techniques, 241
3D object hierarchy, 269, 271
3D user interface design, 234
3D viewing
  performance, 263
  trade-offs, 259
4-directional arrow pointer, 228
4Dwm window manager
  default setting, 113
  See also window managers

A

activation
  menus, 152-154
adaptive rendering, 264
AIFF sound file icon, 25
<Alt>-<Insert> keys, 107
<Alt> key, 155
  3D applications, 235
  for view overlay, 260
appearance, Indigo Magic. *See* Indigo Magic look
application icons
  guidelines, 33
application icons (Desktop icons), 22-23
application-modal dialogs, 207
application models, 42-43, 115-119
applications
  3D, 233
  appearance, 36
  data exchange, 101-109
  data exchange guidelines, 109
  editing text, 96-97
  locations, 32
  naming, 31-32
  processing while minimized, 64, 66
  session management, and, 68-70
  windows, 35-71
approach phase
  rotation, 296, 298, 304, 306, 307
  scaling, 302
  translation, 289, 291, 293
approach phase (3D manipulation), 283
archive file icon, 25
ASCII file icon, 25
Auto Window Placement, 51-52, 93
axial scaling, 300
  phases, 304, 306, 307

B

backgrounds
  and desks, 67
bars
  path navigation, 199
  scrollbars, 194
<Begin> key, 10
binding keys, 179
BMenu, 9
bounding box, 288
bounding volume, 273
browse selection model, 139
BSelect, 9
BTransfer, 9
buttons, 182-183
  dialogs, and, 182
  dynamic, 183
  option buttons, 151-152, 184-185
  pull-down menus, and, 165
  right justification, 37

C

<Cancel> key, 10
Cancel/Close button, 183
cards, 179
cascading menus, 150-151, 169
categories
  windows, 41, 114
center of rotation
  changing, 310
changing center of rotation, 310
checkboxes, 186-187
  Indigo Magic look, 36, 224
  in menus, 168, 170
choice feedback (3D), 284
"Clear" option (in Edit menu), 163
"Click for Help" option (in Help menu), 79, 82
clipboard data, nonpersistent, 102
clipboard transfer model, 102-103
  example, 103
  primary transfer model, independence, 103, 107
"Close" option (in File menu), 160, 219
"Close" option (in window menu), 45, 46
collections, 139-141
  multiple, 142-143
color chooser, 129-130
"Color Editor" option (in Edit menu), 129, 163
colors
  Desktop icons, 17-20
  text fields, 192
color schemes, 39-40
command line input
  windows, 124
command script file icon, 25
constrained translation
  illustration, 292
  phases, 291, 293
  to normal of plane, 292
context-sensitive help, 79, 82
  writing help content, 88
"Context-Sensitive Help" option (in Help menu), 82
control areas
  co-primary windows, 123-124
  main windows, 123-124
control panels
  Desktop, 96-97, 225
  Language, 94
  Mouse Settings, 95, 132
  Window Settings, 51, 67

controls, 181-202
  buttons, 182-183
  checkboxes, 186-187
  dials, 201-202
  File Finder, 146, 199
  labels, 198
  LED buttons, 189-190
  lists, 190-191
  option buttons, 184-185
  radio buttons, 187-189
  scales, 196-197
  scrollbars, 194-196
  text fields, 192-193
  thumbwheels, 200-201
co-primary windows, 41, 114, 120-126
  See also windows
  command line input, 124
  control areas, 123-124
  decorations, 43
  Help button, 83-84
  keyboard shortcuts, 122
  menu (in title bar), 43
  menu bars, 122
  panes, 125
  popup menus, 125
  status areas, 124
  title bar labels, 49
  work areas, scrollable, 123
copying data, 102-103, 104-107
"Copy" option (in Edit menu), 102-103, 162
C program file icon, 25
cross hair pointer, 227
<Ctrl> key, 156, 171-173
  3D applications, 235
<Ctrl><Page Down> keys, 196
<Ctrl><Page Up> keys, 196
<Ctrl><Shift><Tab> keys, 137
<Ctrl><Tab> keys, 137
cursors
  location, 134
customize
  control panels, 4
  desktop, 93
cut and paste. *See* data exchange
"Cut" option (in Edit menu), 102-103, 162
cutting data, 102-103

D

data, selecting, 140-141
data exchange, 101-109
  <Alt>-<Insert> keys, 107
  application guidelines, 109
  clipboard data, nonpersistent, 102
  clipboard transfer
    example, 103
  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, 144-147
  highlighting selected data, 104-107, 224
  nonpersistent selection model, 107
  pasting data, 102-103, 104-107
  persistent always selection model, 107
  primary selection, 104-107
    reinstating, 107
  primary selection example, 105
  primary transfer example, 106
  primary transfer model, 104-107
    clipboard transfer model, independence, 103, 107
  selections, 139-143
data types
  data exchange, supported, 108
decorations
  dialogs, 43, 209
  windows, 43
"Delete" option (in Edit menu), 163

"Deselect All" option (in Edit menu), 163
deselecting data, 140-141
Desks, 66-67
Desks Overview, 4, 66
  example, 66
  window titles, and, 47
Desktop
  icons. *See* Desktop icons
  overview, 3-10
  windows. *See* windows
Desktop control panel, 96-97
  stderr messages, 225
Desktop icons, 4, 11-33, 224
  See also minimized windows
  accessibility guidelines, 33
  appearance, 12-33
  appearance design guidelines, 26
  application icons, 22-23
  behavior, programming, 27-29
  behavior guidelines, 29
  colors, 17-20, 25
  components, 14-15
  creating with IconSmith, 15
  data icons, 12
  design guidelines, 26
  designing appearance, 12-27
  examples, 23
  file icons, 24-25
  Find an Icon tool, 31-32
  generic data file symbol, 25
  generic executable symbol (magic carpet), 14, 22
  icon accessibility guidelines, 33
  icon behavior guidelines, 29
  Icon Catalog, 4, 30-31
  icon color, 17-19
  implementation hints, 28
  launching applications, 6, 28-29
  orientation, 20
  outline color, 19
  predefined templates, 15
  printing files, 28
  shadow color, 25
  size, 17
  states, 6-8, 17-18, 22-23, 224
  templates, 15
  user interaction, 6-8
Desktop services, 73-99
  Auto Window Placement, 93
  customize, 93
  desktop variables, 93
  desktop variables guideline, 98
  editor, preferred, 96-97
  File Alteration Monitor (FAM), 99
  Language control panel, 94
  mouse
    double-click speed, 95
  online documentation, 92
  online help, 76-92
    context-sensitive, 79, 82
    design guidelines, 85
    Help button, 80, 83-84
    Help button guidelines, 86
    help card guidelines, 90
    Help menu, 82-83
    index, 80, 83
    index guidelines, 91
    keyboard shortcuts, 81, 83
    keyboard shortcuts guidelines, 91
    overview, 80, 82
    product information, 81, 83
    task-oriented, 80, 83
    types of, guidelines, 85
    writing help content, 87-92
  schemes, 93
  software installation, 74-75
  user control, 93
dialogs, 41, 114, 203-222
  actions, 211-214
    choosing, 211-212
    default, 212-213
    standard, 211

button labels, 213-214
buttons, and, 182
decorations, 43, 209
designing, 209-218
desks, and, 67
error, 204, 214
file selection, 204, 219-220
Help button, 83-84
information, 204
invoking, 218-222
justification of buttons, 37
menu (in title bar), 43, 209
modes, 207-208
placement, 210
prompt, 204, 214
pull-down menus, and, 165, 168
question, 204, 215
size, 210
title bar labels, 49
types, 203-206
user feedback, 225
warning, 204, 215
working, 204, 215-216, 221
See also windows
dials, 201-202
directories
monitoring changes, 99
Directory Views, 4
direct selection, 268
disabling menu entries, 173, 179
discontiguous selection model, 139
in 3D applications, 270
displays, updating, 224-225
documentation, online, 92
dollying, 245
moving through object, 246
dolly pointer, 237
double-click speed, mouse, 95
<down arrow> key, 137, 154
drag and drop, 144-147
non-text objects, 145-146
pointers, 147
text, 146
drag icons, 147
drag phase
free rotation, 298
rotation, 296
scaling, 302
stretching, 304, 306, 307
translation, 289
drag phase (3D manipulation), 283
drag state, Desktop icons, 7
drop-accepting state, Desktop icons, 7
drop pocket, 199
dynamic buttons, 183
dynamic menu entries, 174

E

editing and viewing (3D), 259
Edit menu, 161-163
edit mode, 260
editor, preferred, 96-97
edit pointer, 237
ellipsis
in menus, 156
in pull-down menus, 168
in pushbuttons, 183
enabling menu entries, 173, 179
<Enter> key, 10, 154
error dialogs, 204, 214
<Esc> key, 235
for view mode, 260
<Escape> key, 10, 154, 193, 196, 197

"Exit" option (in File menu), 160, 219
"Exit" option (in window menu), 45, 46
explicit focus, 55, 133-134
eyepoint during tumbling, 245

F

<F1> key, 10
<F10> key, 137, 154
FAM (File Alteration Monitor), 99
feedback,3D manipulation, 284
feedback. *See* user feedback
feedback styles
  choice feedback, 284
  path feedback, 284
  selection feedback, 284
File Alteration Monitor (FAM), 99
File Finder, 146, 199
file icons (Desktop icons), 24-25
File menu, 158-160
files
  finding, 199
  monitoring changes, 99
  printing, 28
file selection dialogs, 204, 219-220
File Typing Rules (FTRs), 27-29
Find an Icon tool, 4, 31-32
focus, keyboard. *See* keyboard focus
font schemes, 39-40
frame rate
  3D applications, 263
free rotation, 297
  illustration, 298
  phases, 298
FTRs (File Typing Rules), 27-29

G

generic data file symbol (Desktop icons), 25
generic executable symbol (Desktop Icons), 14, 22
grab phase
  axial translation, 291
  free rotation, 298
  rotation, 296
  scaling, 302
  stretching, 304, 306, 307
grab phase (3D manipulation), 283
grab phase (translation), 289
graphic feedback, 224
grouping,and lead objects, 277

H

hardware description, 9-10
<Help> key, 10
Help button, 80, 83-84
  writing help content, 90
help cards, 87-88
Help menu, 82-83, 164
  "Click for Help" option, 79
  examples, 78
  "Index" option, 80
  "Keys & Shortcuts" option, 81
  "Overview" option, 80
  "Product Information" option, 81
  task options, 80
help. *See* online help
highlighting selected data, 104-107, 142, 224
history button, 199
<Home> key, 10
home button, 200
hourglass pointer, 228

I

I-beam pointer, 227
Icon Catalog, 4, 30-31
  Application page, 30
  Collaboration page, 30
  Control Panels page, 30
  Demos page, 30
  Desktop Tools page, 30
  Media Tools page, 30
  Web Tools page, 30
icon color, Desktop icons, 17-19
icons. *See* Desktop icons
images
  examples of minimized windows, 61
  minimized windows, 60-62
implicit focus, 55, 133
"Import" option (in File menu), 159
index, help topics, 80, 83
  writing help content, 89
"Index" option (in Help menu), 80, 83
indicators
  scales, 196
Indigo Magic look, 36-40
  examples, 38
indirect selection, 269
information dialogs, 204
input focus. *See* keyboard focus
inspection
  dollying, 245
  overview, 243
  panning, 248
  tumbling, 244
inspection pointer, 237
installing software, 74-75
internationalization, 94
Inventor file icon, 25

J

justification
  dialog buttons, 37

K

keyboard accelerators, 156, 171-173
keyboard binding, 179
keyboard description, 9-10
keyboard focus, 55-59, 133-138
  dialogs, 210
  guidelines, 59
keyboard navigation, 133-137
keyboard shortcuts
  co-primary windows, 122
  help, 81, 83
    writing help content, 90
  main windows, 122
  support windows, 127
keys
  special, 9-10, 79, 136-137, 154, 155, 156, 171-173, 196
  substitutes to Motif-compliant keys, 10
"Keys & Shortcuts" option (in Help menu), 81, 83

L

labels, 198
  buttons, 183
  checkboxes, 187
  dialog buttons, 213-214
  dials, 202
  ellipsis in button, 183
  LED buttons, 189-190
  lists, 191
  minimized windows, 63, 66, 225

option buttons, 185
radio buttons, 188
scales, 197
text fields, 193
Language control panel, 94
launching applications
Desktop icons, 6, 28-29
lead object
when grouping and ungrouping, 277
when selecting multiple objects, 276
LED buttons, 189-190
<left arrow> key, 137, 154
left mouse button, 9
data exchange, 104
in 3D applications, 234
lists, 190-191
located state, Desktop icons, 6
locate highlight, 36, 224
location cursor, 134
locations
applications, 32
look, Indigo Magic. *See* Indigo Magic look
look-at point during tumbling, 245
"Lower" option (in window menu), 46

M

magic carpet, icon component, 14, 22
main windows, 41, 114, 120-126
See also windows
command line input, 124
control areas, 123-124
decorations, 43
example, 44
Help button, 83-84
keyboard shortcuts, 122
menu (in title bar), 43
menu bars, 122
panes, 125
popup menus, 125
status areas, 124
title bar labels, 48
work areas, scrollable, 123
manipulation of 3D objects
feedback, 284
overview of techniques, 285
phases, 282
manipulators
rotation, 295
scaling, 301
translation, 288
maximize button (in title bar), 46
"Maximize" option (in window menu), 46
maximum window size, 49-50
<Menu> key, 10
menu bars, 149-150, 155-177
co-primary windows, 122
main windows, 122
menus
naming, 166
ordering, 168-169
menus, 149-179
activation, 152-154
cascading menus, 150-151, 169
ellipsis, 156
entries
checkboxes, 168, 170
disabling, 173, 179
dynamic, 174
enabling, 173, 179
naming, 167-168
ordering, 168-169
radio buttons, 168, 170
toggles, 168
Indigo Magic look, 37
keyboard accelerators, 156, 171-173
menu bars, 122, 149-150, 155-177

mnemonics, 155-156, 171
naming, in menu bars, 166
option menus, 151-152, 184-185
naming entries, 185
ordering, in menu bars, 168-169
popup, 125, 136, 177-179
contents, 178
popup menus, 151
posted, 153
pull-down, 149-150, 155-177
buttons, and, 165
contents, 165-170
dialogs, and, 165, 168
naming, in menu bars, 166
naming entries, 167-168
ordering, in menu bars, 168-169
ordering entries, 168-169
support windows, and, 166
spring-loaded, 153
standard, 157-164
submenus, 150-151, 169
tear-off, 166
traversal, 152-154
types, 149-152
window (in title bar), 43, 209
messages, user, 225
middle mouse button, 9
data exchange, 104
in 3D applications, 234
minimize button (in title bar), 46
minimized windows, 60-65
See also Desktop icons
behavior guidelines, 65
images, 60-62
images examples, 61
images guidelines, 64
labels, 63, 66, 225
labels guidelines, 65
processing while minimized, 64, 66
showing status, 64
status example, 64
"Minimize" option (in window menu), 46
minimum window size, 49-50
mnemonics, menus, 155-156, 171
modeless dialogs, 207
modes
dialogs, 207-208
supports windows, and, 128
modifier keys, 235
monitoring, files and directories, 99
mouse
3D applications, 234
buttons, 9
data exchange, 104
description, 9
double-click speed, 95
movement, 132
mouse navigation, 138
Mouse Settings control panel, 95, 132
"Move" option (in window menu), 45
movie file icon, 25
multiple-action pointer grab, example, 59
multiple-action pointer grab model, 56, 58-59
multiple collections, 142-143
"multiple document, no visible main" application model, 43, 119
"multiple document, visible main" application model, 43, 117-118
multiple objects
rotation, 314
scaling, 315
translation, 314
multiple selection model, 140
multi-viewport viewing, 262
adaptive rendering, 264
advantages, 263

N

naming
  applications, 31-32
navigation
  overview, 251
  sidling, 256
navigation. *See* keyboard navigation, mouse navigation
neutral state, Desktop icons, 6
"New" option (in File menu), 159, 219
nonpersistent selection model, 107, 142

O

object-action paradigm, 268
object hierarchy
  selecting up and down, 269, 271
online documentation, 92
online help, 76-92
  context-sensitive, 79, 82
    writing help content, 88
  design guidelines, 85
  Help button, 80, 83-84
    writing help content, 90
  Help button guidelines, 86
  help card guidelines, 90
  Help menu, 82-83
  index, 80, 83
    writing help content, 89
  index guidelines, 91
  keyboard shortcuts, 81, 83
    writing help content, 90
  keyboard shortcuts guidelines, 91
  overview, 80, 82
    writing help content, 88
  product information, 81, 83
  task-oriented, 80, 83
    writing help content, 88-89
  types of, guidelines, 85
  writing help content, 87-92
"Open" option (in File menu), 159, 219
open state, Desktop icons, 7
option buttons, 151-152, 184-185
option menus, 151-152, 184-185
  naming entries, 185
Options menu, 164
orange feedback, 284
orientation
  Desktop icons, 20
outline color, Desktop icons, 19
overlay *See* view overlay, 260
overview help, 80, 82
  writing help content, 88
"Overview" option (in Help menu), 80, 82

P

packaging software for installation, 74-75
<Page Down> key, 196
pages, 179
<Page Up> key, 196
palettes, 179
panes
  co-primary windows, 125
  main windows, 125
panning, 248
pan pointer, 237
"Paste" option (in Edit menu), 102-103, 163, 221
pasting data, 102-103, 104-107
path feedback (3D), 284
path navigation bar, 199

percent-done indicator, 196
performance, 263
peripherals
  3D input device, 262
persistent always selection model, 107, 142
phases
  3D manipulation, 282
  axial scaling, 304, 306, 307
  changing center of rotation, 310, 311
  constrained rotation, 296
  constrained translation, 291, 293
  free rotation, 298
  planar translation, 289
placement
  dialog, 210
  windows, 51-52
planar translation
  illustration, 290
  phases, 289
planar translation (3D), 289
pointer, 132
  designing, 228
  drag and drop, 147
  shapes, 3D applications, 236
  shapes, state and, 132, 224, 226-228
pointer grab, 56-59
  multiple-action, example, 59
  multiple-action model, 56, 58-59
  single-action, example, 57
  single-action model, 56-57
popup menus, 136, 151, 177-179
  contents, 178
  co-primary windows, 125
  disabling entries, 179
  main windows, 125
posted menus, 153
PostScript file icon, 25
preferred editor, 96-97
primary-modal dialogs, 207
primary selection, 104-107, 139
  reinstating, 107
primary transfer model, 104-107
  clipboard transfer model, independence, 103, 107
primary windows, 114
primary windows. *See* main windows, co-primary windows, windows
printing files
  Desktop icons, 28
"Print" option (in File menu), 160
processing while minimized, 64, 66
product information, 81, 83, 221
"Product Information" option (in Help menu), 81, 83, 221
"Promote" option (in Edit menu), 107, 163
prompt dialogs, 204, 214
pull-down menus, 149-150, 155-177
  buttons, and, 165
  contents, 165-170
  dialogs, and, 165, 168
  ellipsis, 168
  entries
    naming, 167-168
    ordering, 168-169
  naming, in menu bars, 166
  ordering, in menu bars, 168-169
  support windows, and, 166
pushbuttons. *See* buttons

Q

question dialogs, 204, 215
question mark pointer, 227

R

radio buttons, 187-189
  Indigo Magic look, 36, 224
  in menus, 168, 170
"Raise" option (in window menu), 45, 46
range selection model, 139
"Redo" option (in Edit menu), 162
release phase
  rotation, 296, 298
  scaling, 302
  translation, 289, 291, 293
release phase (3D manipulation), 283
"Reopen" option (in File menu), 159, 219
resize handles (in title bar), 45
resize pointer, 228
restarting applications
  session management, 68-70
"Restore" option (in window menu), 45
<Return> key, 10
"Revert" option (in File menu), 160, 220
rgb file icon, 25
<right arrow> key, 137, 154
right mouse button, 9
  in 3D applications, 234
rotation
  changing center, 310
  free, 297
  multiple objects, 314
  phases, 296
rotation manipulator, 295

S

"Save As" option (in File menu), 160, 219
"Save" option (in File menu), 160, 220
scales, 196-197
scaling, 300
  changing center, 310
  multiple objects, 315
scaling manipulators, 301
schemes, 39-40, 93
screen backgrounds
  and desks, 67
scrollable lists, 190-191
scrollable work areas
  co-primary windows, 123
  main windows, 123
scrollbars, 194-196
  Indigo Magic look, 37, 224
Search tool, 4
secondary windows. *See* support windows, dialogs, windows
seeking, 249
"Select All" option (in Edit menu), 163
Selected menu, 161
selected state, Desktop icons, 7
selecting data, 140-141
selection feedback (3D), 284
selections, 139-143
  collections, 139-141
    multiple, 142-143
  direct selection (3D), 268
  highlighting, 142
  in 3D applications, 267
  in 3D object hierarchy, 269, 271
  indirect selection, 269
  models, 139-141

session management, 67-71
  guidelines, 71
  logging out, 70-71
  logging out guidelines, 71
SGIHelp system. *See* online help
SGML, 92
shading
  in Indigo Magic look, 36
shadow color, Desktop icons, 25
shared library file icon, 25
<Shift><F1> keys, 79, 82
<Shift><F10> key, 136
<Shift><F10> keys, 10, 154
<Shift> key, 235
  for constrained (axial) translation, 292, 293
  free rotation, 297
<Shift><Tab> keys, 137
sidle pointer, 237
sidling, 256
single-action pointer grab, example, 57
single-action pointer grab model, 56-57
"single document, multiple primaries" application model, 42, 116-117
"single document, one primary" application model, 42, 116
single selection model, 139
  in 3D applications, 270
single-viewport viewing, 262
  advantages, 262
size
  Desktop icons, 17
  dialogs, 210
  windows, 49-50
"Size" option (in window menu), 45
slider bars, 194
sliding scalebars
  scale, 196
software installation, 74-75
Software Manager, 74-75
  advantages, 75
  example, 74
Software Packager, 74-75
spaceball, 262
spring-loaded menus, 153
standard menus, 157-164
states
  applications and sessions management, 68-70
  Desktop icons, 6-8, 17-18, 22-23, 224
  pointers, shapes and, 132, 224, 226-228
status areas, 225
  co-primary windows, 124
  main windows, 124
stderr, 225
stdout, 225
stretching *See* axial scaling, 300
submenus, 150-151, 169
support windows, 41, 114, 127-131
  See also windows
  color chooser, 129-130
  decorations, 43
  design, 127-128
  desks, and, 67
  Help button, 83-84
  keyboard shortcuts, 127
  menu (in title bar), 43
  parent windows, 127
  pull-down menus, and, 166
  title bar labels, 49
  Tools menu, and, 127
system-modal dialogs, 207

T

<Tab> key, 137
task-oriented help, 80, 83
  writing help content, 88-89
tear-off menus, 166
text fields, 192-193
  enhanced, 192
thumbwheels, 200-201
tilt pointer, 237
title bar labels, 46-49, 66
Toolchest, 4
tool palettes, 123, 182
Tools menu, 164
  support windows, and, 127
trackball path
  rotation, 299
trade-offs in 3D viewing, 259
translation
  constrained to normal of plane, 292
  multiple objects, 314
translation manipulator, 288
traversal
  menus, 152-154
tumbling, 244

U

"Undo" option (in Edit menu), 162
ungrouping,and lead objects, 277
uniform scaling, 300
<up arrow> key, 137, 154
updating displays, 224-225
upper-left arrow pointer, 227
upper-right arrow pointer, 227
user feedback, 223-229
  graphic, 224
  messages, 225
  updating displays, 224-225
user messages, 225

V

variables
  desktop, 93
viewing
  3D definition, 241
  overview of techniques, 242
  See Also inspection, navigation, 241
  trade-offs in 3D, 259
viewing and editing (3D), 259
viewing controls
  sliders, 261
  thumbwheels, 261
viewing peripherals
  3D input device, 262
View menu, 164
view mode, 260
view overlay, 260
views, 179

W

warning dialogs, 204, 215
watch pointer, 227
widgets
  outlines in Indigo Magic look, 37
  shading in Indigo Magic look, 36

windows, 35-71, 113-132
  categories, 41, 114
  characteristics guidelines, 52
  co-primary windows, 114
  decorations, 43
  decorations guidelines, 52
  default setting, 113
  dialogs, 114
  Indigo Magic environment, 35
  keyboard focus, 55-59
  main windows, 114, 120-126
  menu (in title bar), 43
  menu bars, 122
  menus guidelines, 53
  minimized, 60-65
  minimized, guidelines, 64
  "multiple document, no visible main" application model, 119
  "multiple document, visible main" application model, 117-118
  no title bar, 55
  placement, 51-52
  placement guidelines, 55
  primary windows, 114
  resize, 49
  scrollable work areas, 123
  "single document, multiple primaries" application model, 116-117
  "single document, one primary" application model, 116
  size, 49-50
  size guidelines, 55
  support windows, 114
  title bar label guidelines, 54
  title bar labels, 46-49, 66
Window Settings control panel, 51, 67
work areas, scrollable
  co-primary windows, 123
  main windows, 123
working dialogs, 204, 215-216, 221

X

X pointer, 228

Y

yellow feedback, 284

Z

zooming, 246